欢迎使用 CodeIgniter

CodeIgniter 是一套给 PHP 网站开发者使用的应用程序开发框架和工具包。它提供一套丰富的标准库以及简单的接口和逻辑结构，其目的是使开发人员更快速地进行项目开发。使用 CodeIgniter 可以减少代码的编写量，并将你的精力投入到项目的创造性开发上。

请阅读下面用户指南的介绍部分，它能让你对 CodeIgniter 有一个大概的了解。接下来就可以从开始页展开 CodeIgniter 之旅了！

CodeIgniter 是为谁准备的？

CodeIgniter 就是你所需要的，如果...

你想要一个小巧的框架。

你需要出色的性能。
你需要广泛兼容标准主机上的各种 PHP 版本和配置(例如 PHP4)。
你想要一个几乎只需 0 配置的框架。
你想要一个不需使用命令行的框架。
你想要一个不需坚守限制性编码规则的框架。
你对 PEAR 这种大规模集成类库不感兴趣。
你不希望被迫学习一门模板语言(虽然可以选择你要求的模板解析器)。
你不喜欢复杂，热爱简单。
你需要清晰、完整的文档。

欢迎使用 CodeIgniter

CodeIgniter 是一套给 PHP 网站开发者使用的应用程序开发框架和工具包。它提供一套丰富的标准库以及简单的接口和逻辑结构，其目的是使开发人员更快速地进行项目开发。使用 CodeIgniter 可以减少代码的编写量，并将你的精力投入到项目的创造性开发上。

请阅读下面用户指南的介绍部分，它能让你对 CodeIgniter 有一个大概的了解。接下来就可以从开始页展开 CodeIgniter 之旅了！

CodeIgniter 是为谁准备的？

CodeIgniter 就是你所需要的，如果...

你想要一个小巧的框架。

你需要出色的性能。
你需要广泛兼容标准主机上的各种 PHP 版本和配置(例如 PHP4)。
你想要一个几乎只需 0 配置的框架。
你想要一个不需使用命令行的框架。
你想要一个不需坚守限制性编码规则的框架。
你对 PEAR 这种大规模集成类库不感兴趣。

你不希望被迫学习一门模板语言(虽然可以选择你要求的模板解析器)。
你不喜欢复杂，热爱简单。
你需要清晰、完整的文档。

目录

服务器要求

PHP version 4.3.2或更新的版本

数据库。支持的数据库包含：MySQL, MySQLi, MS SQL, Postgre, Oracle, SQLite, 和 ODBC

CodeIgniter 许可协议

版权所有 (c) 2006, EllisLab, Inc.

变更记录

Version 1.5.4.............

关于 **CodeIgniter**

CodeIgniter 是由 Ellislab 公司的 CEO Rick Ellis 开发的。其核心框架是为这个程序特别编写的，而其他很多类库、辅助函数和子系统则来自于 Rick Ellis 和 Paul Burdick 编写的内容管理系统 ExpressEngine。

来自 Ruby on Rails 的灵感启发我们创造了一个 PHP 框架，并且将框架的概念引入到网络社区的一般意识中。

CodeIgniter 的 logo 和 icons 是由 Rick Ellis 设计的。

手册目录的下拉菜单使用了 moo.fx 库来编写。

下载 CodeIgniter

CodeIgniter V 1.5.4 (最新版本)

Subversion 服务器

公开的 Subversion 存取目前通过以下地址：http://dev.ellislab.com/svn/CodeIgniter/trunk

请注意通过这个代码仓库取得的文件，我们不能保证代码完全实现功能。

Subversion 是一个版本控制系统。

安装指导

CodeIgniter 安装分为四个步骤：

解压缩安装包。

把 CodeIgniter 文件夹和里面的文件上传到你的服务器。通常 index.php 在根目录。

用任何文本编辑器打开 application/config/config.php 去设置你的基本URL。

如果你打算使用数据库，用任何文本编辑器打开 application/config/database.php 去设置你的数据库参数。

如果你希望通过隐藏 CodeIgniter 文件的位置来增加安全性，你可以修改 system 目录的名字，把它改成任何你想改的。如果已经修改了名字，你必须打开你主目录下面的 index.php 文件设置里面的 $system_folder 变量，把它设成你新改的名字。

以上就是全部安装过程！

如果你刚刚接触 CodeIgniter，请阅读用户指南的入门指引部分，开始学习如何构造动态的 PHP 应用。让我们享受这个过程吧！

疑难解答

如果你发现不管你在 URL 里面写什么都只是出现缺省页面的话，有可能是你的服务器不支持 PATH_INFO 变量，它被用来提供搜索引擎友好的 URL。解决这个问题的第一步是打开 application/config/config.php 文件，查找 URI Protocol 信息。在那里推荐你去尝试一些其他的设置方法。如果这些方法都无效，你就需要让 CodeIgniter 去强行加一个问号去标记你的 URL。为了做到这点，打开你的 application/config/config.php 文件把里面的:

```
$config['index_page'] = "index.php";
```

修改成这样:

$config['index_page'] = "index.php?";

CodeIgniter 是什么？

CodeIgniter 是一个应用程序框架

CodeIgniter 是一个为用 PHP 编写网络应用程序的人员提供的工具包。它的目标是实现让你比从零开始编写代码更快速地开发项目，为此，CI 提供了一套丰富的类库来满足通常的任务需求，并且提供了一个简单的接口和逻辑结构来调用这些库。CodeIgniter 可以将需要完成的任务代码量最小化，这样你就可以把更多的精力放到项目的开发上了。

CodeIgniter 是免费的

CodeIgniter 是经过 Apache/BSD-style 开源许可授权的，只要你愿意就可以使用它。阅读 许可协议 可获得更多的信息。

CodeIgniter 在 PHP 4 上运行

CodeIgniter 的编写完全兼容 PHP 4。Although we would have loved to take advantage of the better object handling in PHP 5 since it would have simplified some things we had to find creative solutions for (looking your way, multiple inheritance), at the time of this writing PHP 5 is not in widespread use, which means we would be alienating most of our potential audience. Major OS vendors like RedHat have yet to support PHP 5, and they are unlikely to do so until 2007, so we felt that it did not serve the best interests of the PHP community to write CodeIgniter in PHP 5.

Note: CodeIgniter will run on PHP 5. It simply does not take advantage of any native features that are only available in that version.

CodeIgniter 是轻量级的

真正的轻量级。我们的核心系统只需要一些非常小的库，这与那些需要更多资源的框架完全相反。额外的库文件只在请求的时候加载，依需求而定，所以核心系统是非常快而且轻的。

CodeIgniter 是快速的

速度非常快。你要找到一个比 CodeIgniter 表现更优的框架应该很难吧。

CodeIgniter 使用 M-V-C 模型

CodeIgniter 使用了模型（Model）- 视图（View）- 控制器（Controllers）的方法，这样可以更好地使表现层和逻辑层分离。这对项目的模板设计者来说是非常有用的，它最小化了模板中的程序代码量。我们在 MVC 各自的页面中对此做了更多的介绍。

CodeIgniter 生成干净的 URLs

CodeIgniter 生成的 URL 非常干净而且是对搜索引擎友好化的。不同于标准的“字符串查询”方法，CodeIgniter使用了 segment-based 这样的方法：

www.your-site.com/news/article/345

注意：index.php 文件是被默认包含在 URL 中的，但是可以通过更改 .htaccess 文件来改变这个设置。

CodeIgniter 功能强大

CodeIgniter 拥有全范围的类库，可以完成大多数通常需要的网络开发任务，包括： 读取数据库、发送电子邮件、数据确认、保存 session 、对图片的操作，以及支持 XML-RPC 数据传输等。

CodeIgniter 是可扩展的

这个系统可以非常简单的通过插件和 helper 类库来进行扩展，或者也可以通过扩展类、系统钩子来实现。

CodeIgniter 不需要模板引擎

Although CodeIgniter *does* come with a simple template parser that can be optionally used, it does not force you to use one. Template engines simply can not match the performance of native PHP, and the syntax that must be learned to use a template engine is usually only marginally easier than learning the basics of PHP. Consider this block of PHP code:

```
<ul>
<?php foreach ($addressbook as $name):?>
<li><?=$name?></li>
<?php endforeach; ?>
</ul>
```

Contrast this with the pseudo-code used by a template engine:

```
<ul>
{foreach from=$addressbook item="name"}
<li>{$name}</li>
{/foreach}
</ul>
```

Yes, the template engine example is a bit cleaner, but it comes at the price of performance, as the pseudo-code must be converted back into PHP to run. Since one of our goals is *maximum performance*, we opted to not require the use of a template engine.

CodeIgniter 已彻底文档化

程序员都喜欢写代码讨厌写文档。当然我们也一样，但是既然文档和代码本身一样重要，我们就要完成它了。况且我们代码资源极其干净而且方便注释。

CodeIgniter 拥有一个友好的用户社区

你可以在我们的 社区论坛 中看到一个成长中的积极活跃的用户社区。

CodeIgniter 特性

Features in and of themselves are a very poor way to judge an application since they tell you nothing about the user experience, or how intuitively or intelligently it is designed. Features don't reveal anything about the quality of the code, or the performance, or the attention to detail, or security practices. The only way to really judge an app is to try it and get to know the code. Installing CodeIgniter is child's play so we encourage you to do just that. In the mean time here's a list of CodeIgniter's main features.

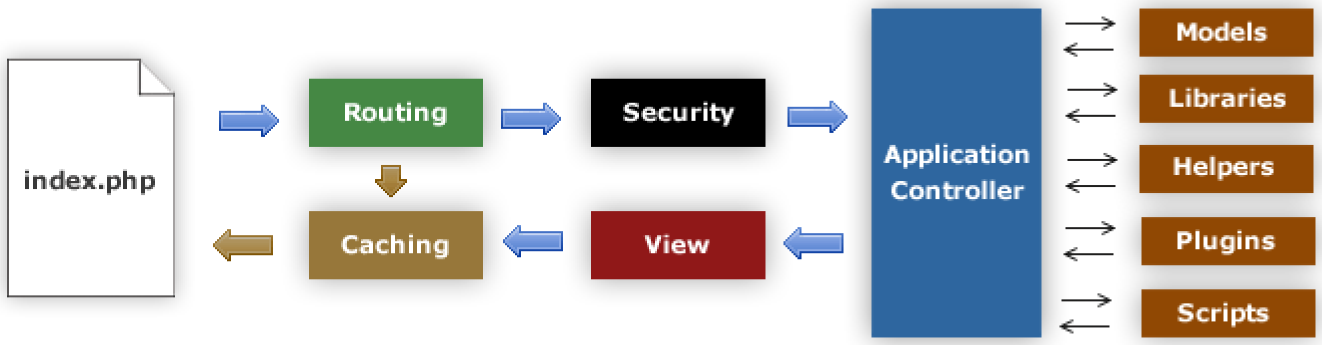
Model-View-Controller Based System

PHP 4 Compatible
Extremely Light Weight
Full Featured database classes with support for several platforms.
Active Record Database Support
Form and Data Validation
Security and XSS Filtering
Session Management
Email Sending Class. Supports Attachments, HTML/Text email, multiple protocols (sendmail, SMTP, and Mail) and more.
Image Manipulation Library (cropping, resizing, rotating, etc.). Supports GD, ImageMagick, and NetPBM
File Uploading Class
FTP Class
Localization
Pagination
Data Encryption
Benchmarking
Full Page Caching
Error Logging
Application Profiling
Scaffolding
Calendaring Class
User Agent Class
Zip Encoding Class
Template Engine Class

Trackback Class
XML-RPC Library
Unit Testing Class
Search-engine Friendly URLs
Flexible URI Routing
Support for Hooks, Class Extensions, and Plugins
Large library of "helper" functions

应用程序流程图

下图说明数据流如何贯穿整个系统：



index.php 作为前端控制器，初始化运行 CodeIgniter 所需要的基本资源。

Router 检查 HTTP 请求，以确定谁来处理请求。
如果缓存(Cache)文件存在，它将绕过通常的系统执行顺序，被直接发送给浏览器。
安全(Security)。应用程序控制器(Application Controller)装载之前，HTTP 请求和任何用户提交的数据将被过滤。
控制器(Controller)装载模型、核心库、插件、助手类，以及任何处理特殊请求所需的其它资源。
最终视图(View)渲染发送到 Web 浏览器中的内容。如果开启缓存(Caching)，视图首先被缓存，所以将可用于以后的请求。

模型-视图-控制器

CodeIgniter 是基于模型-视图-控制器这一设计模式的。MVC 是一种将应用程序的逻辑层和表现层进行分离的方法。在实践中，由于表现层从 PHP 脚本中分离了出来，所以它允许你的网页中只包含很少的脚本。

模型 代表你的数据结构。通常来说，你的模型类将包含取出、插入、更新你的数据库资料这些功能。

视图 是展示给用户的信息。 一个视图通常是一个网页，但是在 CodeIgniter 中， 一个视图也可以是一个页面片段，如头部、顶部。它还可以是一个 RSS 页面，或其他任一页面。

控制器 是一个*中介*，联接视图和模型，以及其他任何处理 HTTP 请求和产生网页的资源。

CodeIgniter 在 MVC 使用上非常宽松，因此模型不是必需的。如果你不需要使用这种分离方式，或是发觉维护模型比你想象中的复杂很多，你可以不用理会它们而创建自己的应用程序，并最少化使用控制器和视图。CodeIgniter 也可以和你现有的脚本合并使用，或者允许自行开发此系统的核心库，其目的是，使你可以以最适合你的方式工作。

设计和架构目标

CodeIgniter的目标是在最小化，最轻量级的开发包中得到最大的执行效率、功能和灵活性。

为了达到这个目标，我们在开发过程的每一步都致力于基准测试、重构和简化工作，拒绝加入任何不助于目标的东西。

从技术和架构角度看，CodeIgniter 按照下列目标创建：

动态实例化. 在 CodeIgniter 中，元件的导入和函数的执行只有在被要求的时候才执行，而不是在全局范围。除了最小的核心资源外，不假设系统需要任何资源，因此缺省的系统非常轻量级。被HTTP请求所触发的事件，以及你设计的控制器和视图将决定它们什么时候被引用。

松耦合. 耦合是指一个系统的元件之间的相关程度。越少的元件互相依赖那么那个系统的重用性和灵活性就越好。我们的目标是一个非常松耦合的系统。

组件单一性. 单一是指组件有一个非常小的专注目标。在 CodeIgniter 里面，为了达到最大的用途最大的每个类和它的功能都是高度自治的。

CodeIgniter 是一个动态实例化，高度元件奇异性的松耦合系统。它在一个小型包里面力求做到简单、灵活和高性能。

CodeIgniter 将从这里开始

任何应用软件的学习都需要花费一定的精力。我们尽了自己最大的努力使大家在学习中少走弯路，并且充满乐趣。

第一步就是安装 CodeIgniter，然后阅读**介绍**部分的所有主题。

接下来按顺序阅读**常规主题**(请点击右上角的“目录” 或点击目录页)里面的每一个内容。那里的每个内容都是承接前一个的，并且提供了方便实践的例子代码。

一旦你掌握了这些基础概念，你就可以浏览**类库参考**和**辅助函数参考**，学习如何使用类库和辅助函数文件了。

如果遇到了问题可以到我们的中国开发者社区寻找解答，同时，英文的 Wiki 里面也提供了其他用户所写的例子代码。

CodeIgniter URL

默认情况下，CodeIgniter 中的 URL 被设计成对搜索引擎和人类友好。不同于使用标准“查询

字符串”方法的是，CodeIgniter 使用**基于段**的方法：

www.your-site.com/news/article/my_article

注意：查询字符串形式的 URL 是可选的，分述如下。

URI 段

根据模型-视图-控制器模式，在此 URL 段一般以如下形式表示：

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

第一段表示调用控制器**类**。

第二段表示调用类中的**函数**或方法。

第三及更多的段表示的是传递给控制器的参数，如 ID 或其他各种变量。

URI 类和 URL 辅助函数中的函数可以使你的 URI 更简单的工作。另外，使用 URI 路由特性可以将你的 URL 重定向，以获得更大的灵活性。

删除 index.php 文件

默认情况下，**index.php** 文件将被包含在你的 URL 中：

www.your-site.com/index.php/news/article/my_article

你可以很容易的通过 .htaccess 文件来设置一些简单的规则删除它。下面是一个例子，使用“negative”方法将非指定内容进行重定向：

```
RewriteEngine on

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

在上面的例子中，可以实现任何非 index.php、images 和 robots.txt 的 HTTP 请求都被指向 index.php。

添加 URL 后缀

通过设置 config/config.php 文件，你可以为 CodeIgniter 生成的 URL 添加一个指定的文件后缀。举例来说，如果 URL 是这样的：

www.your-site.com/index.php/products/view/shoes

你可以随意添加一个后缀，例如 .html，使其显示为：

www.your-site.com/index.php/products/view/shoes.html

启用查询字符串

在一些情况下你需要在 URL 中使用查询字符串：

index.php?c=products&m=view&id=345

CodeIgniter 支持这个功能是可选的，可以在 application/config.php 文件中进行设置。如果你打开 config 文件可以看到如下内容：

```
$config['enable_query_strings'] = FALSE;

$config['controller_trigger'] = 'c';
$config['function_trigger'] = 'm';
```

如果你将 enable_query_strings 更改为 TRUE ，那么这个功能就被激活了。此时，你就可以通过关键字来调用需要的控制器和方法了：

```
index.php?c=controller&m=method
```

请注意：如果你使用查询字符串，那么就必须使用自己建立的 URL ，而且不能使用URL 辅助函数（或是其他生成 URL 的辅助函数，例如表单辅助函数），因为这些都是根据分段 URL 设计的。

控制器

控制器是应用程序的心脏，因为它们决定如何处理 HTTP 请求。

什么是控制器？

简而言之，一个控制器就是一个类文件，是以一种能够和 URI 关联在一起的方式来命名的。

假设这个 URI:

```
www.your-site.com/index.php/blog/
```

在上面的例子中，CodeIgniter 将尝试寻找并装载一个名为 blog.php 的控制器。

当控制器的名字匹配 URI 的第一段时，它将被装载。

让我们试试看：你好，世界！

我们来创建一个简单的控制器，以便更直观地了解其工作原理。使用你的文本编辑器，创建一个名为 blog.php 的文件，然后输入下列代码：

```
<?php class Blog extends Controller { function index() { echo '你好，世界！'; } } ?>
```

然后保存文件到 application/controllers/ 文件夹。

现在使用类似这样的 URL 访问你的站点：

www.your-site.com/index.php/blog/

如果你做的没错，你应该看到你好，世界！.

注意：类名必须以大写字母开头。换句话说，这是有效的：

```
<?php

class Blog extends Controller {
}
?>
```

这是**无效**的：

```
<?php

class blog extends Controller {
}
?>
```

同时，始终确保你的控制器扩展自父控制器类，以便它能够继承其所有的方法。

方法

上面的例子中用到的方法名是 index()。如果 URI 的**第二部分**为空的话，会默认载入 “index” 方法。也可以将地址写成这样来访问 “Hello World”：

www.your-site.com/index.php/blog/index/

URI 的第二部分是用来决定调用控制器中哪个方法的。

我们再来试试。在你的控制器中加入一个新的方法：

```
<?php class Blog extends Controller { function index() { echo 'Hello World!'; } function comments() { echo 'Look at this!'; } } ?>
```

现在在地址栏中输入下面的内容来访问 comment 方法：

www.your-site.com/index.php/blog/comments/

你应该看到新的信息了：Look at this!

将 URI 片段传递给方法

如果你的 URI 超过两个部分，那么超过的将被作为参数传递给方法。

举例来说，如果你的 URI 是这样的：

www.your-site.com/index.php/products/shoes/sandals/123

URI 的第3和第4部分会被传递给你的方法（“sandals” 和 “123”）：

```
<?php
class Products extends Controller {
    function shoes($sandals, $id)
    {
        echo $sandals;
        echo $id;
    }
}
?>
```

注意：如果你使用 URI 路由特性，则传递到方法中的 URI 片段将被重新路由一次。

定义默认控制器

当你的网站没有设置 URI 或者直接从根目录访问的时候，CodeIgniter 会加载默认控制器。打开 application/config/routes.php 文件来设置默认控制器：

```
$route['default_controller'] = 'Blog';
```

这里的 Blog 就是你希望使用的控制器的名字。如果此时你不指定任何 URI 片段来访问你的主页就会看到默认的“Hello World”信息。

重新定义方法的调用规则

如上所述，URI 的第二片段决定会调用控制器中的哪个方法。CodeIgniter 允许你使用 _remap() 方法来废除这种规则：

```
function _remap()
{
    // Some code here...
}
```

注意：如果你的控制器中包含一个名为 _remap() 的方法，那么不管你的 URI 中包含什么，它**总会**被忽略掉。这个方法会废除掉由 URI 片段来决定哪个方法被调用的规则，允许你重新定义调用方法的规则（方法的路由规则）。

被重新定义的方法调用方式（一般是 URI 中的第二片段）将作为一个参数传递给 _remap()：

```
function _remap($method)
{
    if ($method == 'some_method')
    {
        $this->$method();
    }
    else
    {
        $this->default_method();
    }
}
```

处理输出

CodeIgniter 拥有一个输出类用来确保你修改的数据会自动被传递给浏览器。关于这个的更多信息可以在视图和输出类里找到。有些时候，你可能想要自己发布修改一些最终的数据或是自己把它传递给浏览器。CodeIgniter 允许你给你的控制器增加一个名为 **_output()** 的方法来接收最终的数据。

注意： 如果你的控制器包含一个 _output() 方法，那么它将**总是**被调用，而不是直接输出最终的数据。这个方法的第一个参数即为最终的输出。

例如：

```
function _output($output)
{
    echo $output;
}
```

请注意，你的 _output() 将接收最终的数据。 Benchmark and memory usage data will be rendered, cache files written (if you have caching enabled), and headers will be sent (if you use that feature) before it is handed off to the _output() function. If you are using this feature the page execution timer and memory usage stats might not be perfectly accurate since they will not take into acccount any further processing you do. For an alternate way to control output *before* any of the final processing is done, please see the available methods in the Output Class.

私有方法

在某些情况下，你可能想要隐藏一些方法使之无法对外查阅。将方法私有化很简单，只要在方法名字前面加一个下划线（“_”）做前缀就无法通过 URL 访问到了。例如，如果你有一个像这样的方法：

```
function _utility()
{
  // some code
}
```

那么，通过下面这样的 URL 进行访问是无法访问到的：

www.your-site.com/index.php/blog/_utility/

如何将控制器放入子文件夹中

如果你在建立一个大型的应用程序，你会发现 CodeIgniter 可以很方便的将控制器放到一些子文件夹中。

只要在 application/controllers 目录下创建文件夹并放入你的控制器就可以了。

注意： 如果你要使用某个子文件夹下的功能，就要保证 URI 的第一个片段是用于描述这个文件夹的。例如说你有一个控制器在这里：

application/controllers/products/shoes.php

调用这个控制器的时候你的 URI 要这么写：

www.your-site.com/index.php/products/shoes/123

你的每个子文件夹中需要包含一个默认的控制器，这样如果 URI 中只有子文件夹而没有具体功能的时候它将被调用。只要将你作为默认的控制器名称在 application/config/routes.php 文件中指定就可以了。

CodeIgniter 也允许你使用 URI 路由 功能来重新定向 URI。

构造函数

如果要在你的任意控制器中使用构造函数的话，那么**必须**在里面加入下面这行代码：

parent::Controller();

这行代码的必要性在于，你此处的构造函数会覆盖掉这个父控制器类中的构造函数，所以我们要手动调用它。

如果你对构造函数不熟悉可以看这里，在 PHP 4 中，一个构造函数就是一个拥有和类名完全相同的名字的简单函数：

```
<?php

class Blog extends Controller {
        function Blog()
        {
              parent::Controller();
        }
}
?>
```

在 PHP 5 中，构造函数的语法是这样的：

```
<?php

class Blog extends Controller {
        function __construct()
        {
                parent::Controller();
        }
}
?>
```

如果你需要设定某些默认的值或是在实例化类的时候运行一个默认的程序，那么构造函数在这方面就非常有用了。

构造函数并不能返回值，但是可以用来设置一些默认的功能。

已保留的方法名称

因为你添加的控制器类继承了主要的应用程序控制器，所以你要小心你的方法名不要和那个类中的方法名一样了，否则你的方法会覆盖原有的。下面列出了已经保留的名称，请不要将你的控制器方法命名为这些：

Controller

CI_Base
_ci_initialize
_ci_scaffolding

如果你使用的是 PHP 4 这里有一些附加的名字。这些只在 PHP 4 下会被使用。

CI_Loader

config
database
file
helper
helpers
language
library
model
plugin
plugins
scaffolding
script
view
vars
_ci_assign_to_models
_ci_autoloader
_ci_init_class
_ci_init_scaffolding
_ci_is_instance
_ci_load

_ci_load_class
_ci_object_to_array

就这样了！

OK，总的来说，这就是关于控制器的所有内容了。

视图

简而言之，一个视图就是一个网页，或是网页的部分，如头部，底部，侧边栏等等。事实上，如果你需要这种层次类型，视图可以很舒服的载入其他视图。

视图从不直接调用，必须被一个控制器来调用。记住，在一个 MVC 框架中，控制器扮演着交通警察的角色，那么，他有责任去取回某一特定的视图。如果你还没有阅读过控制器页面的话，你应该事先阅读控制器页面。

下面使用你在控制器页面已经创建过的示例控制器,让我们来给他添加个视图。

创建视图

使用你的文本编辑器，创建一个名为 blogview.php 的文件，写入以下代码：

<html> <head> <title>My Blog</title> </head> <body> <h1>Welcome to my Blog!</h1> </body> </html>

然后保存文件到 application/views/ 文件夹。

载入视图

你必须使用下面的函数来载入一个视图文件:

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

上面的 name 便是你的视图文件的名字. 注意: .php 文件的扩展名(后缀名)没有必要专门写出,除非你使用了其他的.

现在, 打开你先前写的名为 blog.php 控制器文件,并且使用视图载入函数替换echo段代码:

<?php class Blog extends Controller { function index() { $this->load->view('blogview'); } } ?>

如果你使用先前你用的 URL 浏览你的网站,你将会看到你的新视图..URL与下面的类似:I

www.your-site.com/index.php/blog/

用子文件夹存储视图

如果你想让文件更有组织性,你也可以用子文件夹来存储你的视图文件.. 当你在载入视图时,必须加上子文件夹的名字. 示例如下:

$this->load->view('folder_name/file_name');

给视图添加动态数据

数据通过控制器以一个**数组**或是**对象**的形式传入视图 ，这个数组或对象作为视图载入函数的第二个参数 .下面便是使用数组的示例:

```
$data = array(
                    'title' => 'My Title',
                    'heading' => 'My Heading',
                    'message' => 'My Message'
             );

$this->load->view('blogview', $data);
```

这里是使用对象的示例:

```
$data = new Someclass();
$this->load->view('blogview', $data);
```

注意：如果你使用一个对象，那么类变量将转换为数组元素。

好了，让我们用你的控制器试试。打开控制器并添加以下代码：

<?php class Blog extends Controller { function index() { $data['title'] = "My Real Title"; $data['heading'] = "My Real Heading"; $this->load->view('blogview', $data); } } ?>

现在，打开你的视图文件，将其中的文本替换成与数组对应的变量：

<html> <head> <title><?php echo $title;?></title> </head> <body> <h1><?php echo $heading;?></h1> </body> </html>

然后使用你先前用过的URL载入页面，你将看到变量已经被替换。

注意: 你可能已经注意到，在上面的示例中，我们使用 PHP 的替代语法。如果，你对此不是太熟悉，你可以阅读这篇文章。

创建循环

你传入视图文件的数据，不仅仅局限于简单的变量。你可以传递多维数组。例如：你从数据库里面取出数据就是典型的多维数据。

这里是个简单的示例。添加以下代码到你的控制器：

<?php class Blog extends Controller { function index() { $data['todo_list'] = array('Clean House', 'Call Mom', 'Run Errands'); $data['title'] = "My Real Title"; $data['heading'] = "My Real Heading"; $this->load->view('blogview', $data); } } ?>

现在打开你的视图文件，创建一个循环：

<html> <head> <title><?=$title;?></title> </head> <body> <h1><?=$heading;?></h1> <h3>My Todo List</h3> <ul> <?php foreach($todo_list as $item):?> <li><?=$item;?></li> <?php

endforeach;?> </ul> </body> </html>

模型

模型对于那些想用传统MVC方式的人来说是**可选**的。

什么是模型?

模型是专门用来和数据库打交道的PHP类。例如，假设你想用CodeIgniter来做一个Blog。你可以写一个模型类，里面包含插入、更新、删除Blog数据的方法。下面的例子将向你展示一个普通的模型类:

```
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']));
    }
}
```

注意: 上面用到的函数是 Active Record 数据库函数.

剖析模型

模型类文件存放在 application/models/ 文件夹。 如果你愿意，可以在里面建立子文件夹。

最基本的模型类必须像这样:

```
class Model_name extends Model {
    function Model_name()
    {
        parent::Model();
    }
}
```

Model_name 是模型类的名字。 类名的首字母**必须**大写。并且确保你的类继承了基本模型类(Base Model Class)。

文件名应该是模型类名的小写版。比如，如果你的类是:

```
class User_model extends Model {
    function User_model()
    {
        parent::Model();
    }
}
```

类的文件名应该是：

application/models/user_model.php

载入模型

模型可以在 controller 中被引用。就像这样:

$this->load->model('Model_name');

如果模型文件在子文件夹下，引用的时候要带上相对路径名。例如：如果你有一个模型 application/models/blog/queries.php。下面的代码可以引用它:

$this->load->model('blog/queries');

模型一旦被载入，你就能通过下面的方法使用它:

$this->load->model('Model_name');

$this->Model_name->function();

If you would like your model assigned to a different object name you can specify it via the second parameter of the loading function:

$this->load->model('Model_name', 'fubar');

$this->fubar->function();

Here is an example of a controller, that loads a model, then serves a view:

```
class Blog_controller extends Controller {
    function blog()
    {
        $this->load->model('Blog');
        $data['query'] = $this->Blog->get_last_ten_entries();
        $this->load->view('blog', $data);
    }
}
```

Connecting to your Database

When a model is loaded it does **NOT** connect automatically to your database. The following options for connecting are available to you:

You can connect using the standard database methods described here, either from within your Controller class or your Model class.

You can tell the model loading function to auto-connect by passing TRUE (boolean) via the third parameter, and connectivity settings, as defined in your database config file will be used: $this->load->model('Model_name', '', TRUE);

You can manually pass database connectivity settings via the third parameter: $config['hostname'] = "localhost";

$config['username'] = "myusername";
$config['password'] = "mypassword";
$config['database'] = "mydatabase";
$config['dbdriver'] = "mysql";
$config['dbprefix'] = "";
$config['pconnect'] = FALSE;
$config['db_debug'] = TRUE;
$config['active_r'] = TRUE;
$this->load->model('Model_name', '', $config);

辅助函数

辅助函数，顾名思义，是帮助我们完成特定任务的函数。每个辅助函数文件仅仅是一些函数的集合。例如，URL Helpers 可以帮助我们创建链接，Form Helpers 可以帮助我们穿件表单，Text

Helpers 提供一系列的格式化输出方式，Cookie Helpers 能帮助我们设置和读取COOKIE， File Helpers 能帮助我们处理文件，等等。

跟其他部分不同的是，辅助函数不是用类的方式来实现的。它们仅仅是一些简单的过程处理函数。每个辅助函数处理一个特定的任务，并且不必依靠其他函数。

CodeIgniter 默认是没有载入辅助函数文件的，所以如果你想用辅助函数，就必须先载入它。一旦被载入，辅助函数将全局可用(globally available)，你可以在 controller 和 views 中使用它们。

辅助函数文件一般保存在 system/helpers 文件夹。 但是你仍然可以选择在你的application文件夹下建立一个叫helpers的文件夹来存放它们。CodeIgniter 将会先在system/application/helpers寻找对应的辅助函数文件，如果目录不存在或者目录下没有对应的辅助函数文件，CI 才会载入 system/helpers 下的辅助函数文件。

载入辅助函数

载入辅助函数是非常简单的:

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

name 是辅助函数文件的名字(不带.php后缀 和"helper" 部分)。

例如，要载入文件名为url_helper.php的URL Helper，你将会用到下面的语句：

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

辅助函数可以在你的控制器(controller)的任何地方被载入，甚至可以在视图(View)文件中被载入(我们并不建议你这么做)。请在使用辅助函数之前载入他们。你可以在你的控制器构造函数中载入它们，以便辅助函数能自动在其他函数之前被载入。你也可以在要用到辅助函数的地方当场载入。

注意: 辅助函数载入函数并不返回值，所以不要尝试将它付给一个变量，直接像这样用就可以了。

载入多个辅助函数

如果你想一次载入多个辅助函数，你可以这样做:

$this->load->helper(array('helper1', 'helper2', 'helper3'));

自动载入辅助函数

如果你想要的话， CodeIgniter 可以自动为你载入辅助函数。你可以通过打开 application/config/autoload.php ，并往自动载入数组(autoload array)中增加辅助函数来实现。

使用辅助函数

一旦你载入了想要用到辅助函数文件，你就可以用标准的函数调用方法来使用里面的函数。

例如，要使用anchor() 函数来建立一个链接，在视图(View)文件里面你可以这样做:

<?=anchor('blog/comments', 'Click Here');?>

"Click Here" 是链接的名字，"blog/comments" 是链接的URI。

现在可以做什么?

在目录里面有所有的辅助函数列表，你可以打开每个文件看看他们都能做些什么。

插件

插件的工作方式几乎和辅助函数一模一样。它们最主要的区别在于插件文件一般只有一个函数，而辅助函数文件里面通常是一系列函数。辅助函数被看作系统核心的一部分，而插件通常是网友制作和分享的。

插件文件一般保存在system/plugins 文件夹。但是你仍然可以选择在你的application文件夹下建立一个叫plugins的文件夹来存放它们。CodeIgniter 将会先在system/application/plugins 寻找对应的插件文件，如果目录不存在或者目录下没有对应的插件文件，CI 才会载入 system/plugins下的插件文件。

载入插件

载入插件是非常简单的:

$this->load->plugin('name');

name 是插件文件的名字(不带.php后缀 和"plugin" 部分)。

例如，要载入文件名为captcha_pi.php的Captcha 插件，你将会用到下面的语句:

$this->load->plugin('captcha');

插件可以在你的控制器(controller)的任何地方被载入，甚至可以在视图(View)文件中被载入(我们并不建议你这么做)。请在使用插件之前载入他们。你可以在你的控制器构造函数中载入它们，以便插件能自动在其他函数之前被载入。你也可以在要用到插件的地方当场载入。

注意: 插件载入函数并不返回值，所以不要尝试将它付给一个变量，直接像这样用就可以了。

载入多个插件

如果你想一次载入多个插件，你可以这样做:

$this->load->plugin(array('plugin1', 'plugin2', 'plugin3'));

Auto-loading Plugins

如果你想要的话，CodeIgniter 可以自动为你载入插件。你可以通过打开 application/config/autoload.php ，并往自动载入数组(autoload array)中增加插件来实现。

使用插件

一旦你载入了想要用到插件文件，你就可以用标准的函数调用方法来使用里面的函数。

使用 **CodeIgniter** 类库

所有的类库文件存放在system/libraries 文件夹。大多数情况下你需要预先在controller中初始化后才能使用它们:

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

class name是你想要使用的类名。例如，要载入'确认类'，你可以这样做：

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

一旦类库被载入，你就可以按照用户手册中的方法来使用它们。

创建类库

请阅读用户手册中关于 创建你自己的类库 的部分。

Creating Libraries

When we use the term "Libraries" we are normally referring to the classes that are located in the libraries directory and described in the Class Reference of this user guide. In this case, however, we will instead describe how you can create your own libraries within your application/libraries directory in order to maintain separation between your local resources and the global framework resources.

As an added bonus, CodeIgniter permits your libraries to extend native classes if you simply need to add some functionality to an existing library. Or you can even replace native libraries just by placing identically named versions in your application/libraries folder.

In summary:

You can create entirely new libraries.

You can extend native libraries.
You can replace native libraries.

The page below explains these three concepts in detail.

Note: The Database classes can not be extended or replaced with your own classes, nor can the main Controller class. All other classes are able to be replaced/extended.

Storage

Your library classes should be placed within your application/libraries folder, as this is where CodeIgniter will look for them when they are initialized.

Naming Conventions

File names must be capitalized. For example: Myclass.php

Class declarations must be capitalized. For example: class Myclass

Class names and file names must match.

The Class File

Classes should have this basic prototype (Note: We are using the name Someclass purely as an example):

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

class Someclass {
        function some_function()
        {
        }
}
?>
```

Using Your Class

From within any of your Controller functions you can initialize your class using the standard:

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

Where *someclass* is the file name, without the ".php" file extension. You can submit the file name capitalized or lower case. CodeIgniter doesn't care.

Once loaded you can access your class using the lower case version:

```
$this->someclass->some_function(); // Object instances will always be lower case
```

Passing Parameters When Initializing Your Class

In the library loading function you can dynamically pass data via the second parameter and it will be passed to your class constructor:

```
$params = array('type' => 'large', 'color' => 'red');

$this->load->library('Someclass', $params);
```

If you use this feature you must set up your class constructor to expect data:

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

class Someclass {
        function Someclass($params)
        {
                // Do something with $params
        }
}
?>
```

You can also pass parameters stored in a config file. Simply create a config file named identically to the class file name and store it in your application/config/ folder. Note that if you dynamically pass parameters as described above, the config file option will not be available.

Utilizing CodeIgniter Resources within Your Library

To access CodeIgniter's native resources within your library use the get_instance() function. This function returns the CodeIgniter super object.

Normally from within your controller functions you will call any of the available CodeIgniter functions using the $this construct:

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

$this->load->library('session');
$this->config->item('base_url');
etc.

$this, however, only works directly within your controllers, your models, or your views. If you would like to use CodeIgniter's classes from within your own custom classes you can do so as follows:

First, assign the CodeIgniter object to a variable:

$CI =& get_instance();

Once you've assigned the object to a variable, you'll use that variable *instead* of $this:

$CI =& get_instance();

$CI->load->helper('url');
$CI->load->library('session');
$CI->config->item('base_url');
etc.

Note: You'll notice that the above get_instance() function is being passed by reference:

$CI =& get_instance();
This is very important. Assigning by reference allows you to use the original CodeIgniter object rather than creating a copy of it.
Also, please note: If you are running PHP 4 it's usually best to avoid calling get_instance() from within your class constructors. PHP 4 has trouble referencing the CI super object within application constructors since objects do not exist until the class is fully instantiated.

Replacing Native Libraries with Your Versions

Simply by naming your class files identically to a native library will cause CodeIgniter to use it instead of the native one. To use this feature you must name the file and the class declaration exactly the same as the native library. For example, to replace the native Email library you'll create a file named application/libraries/Email.php, and declare your class with:

class CI_Email {

```
}
```

Note that most native classes are prefixed with CI_.

To load your library you'll see the standard loading function:

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

Note: At this time the Database classes can not be replaced with your own versions.

Extending Native Libraries

If all you need to do is add some functionality to an existing library - perhaps add a function or two - then it's overkill to replace the entire library with your version. In this case it's better to simply extend the class. Extending a class is nearly identical to replacing a class with a couple exceptions:

The class declaration must extend the parent class.

Your new class name and filename must be prefixed with MY_ (this item is configurable. See below.).

For example, to extend the native Email class you'll create a file named application/libraries/MY_Email.php, and declare your class with:

```
class MY_Email extends CI_Email {

}
```

Note: If you need to use a constructor in your class make sure you extend the parent constructor:

```
class MY_Email extends CI_Email {
    function My_Email()
    {
        parent::CI_Email();
    }
}
```

Loading Your Sub-class

To load your sub-class you'll use the standard syntax normally used. DO NOT include your prefix. For example, to load the example above, which extends the Email class, you will use:

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

Once loaded you will use the class variable as you normally would for the class you are extending. In the case of the email class all calls will use:

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

Setting Your Own Prefix

To set your own sub-class prefix, open your application/config/config.php file and look for this item:

$config['subclass_prefix'] = 'MY_';

Please note that all native CodeIgniter libraries are prefixed with CI_ so DO NOT use that as your prefix.

创建核心系统类

Every time CodeIgniter runs there are several base classes that are initialized automatically as part of the core framework. It is possible, however, to swap any of the core system classes with your own versions or even extend the core versions.

Most users will never have any need to do this, but the option to replace or extend them does exist for those who would like to significantly alter the CodeIgniter core.

Note: Messing with a core system class has a lot of implications, so make sure you know what you are doing before attempting it.

System Class List

The following is a list of the core system files that are invoked every time CodeIgniter runs:

Benchmark

Input
Config
Hooks
Router
URI
Language
Loader
Controller
Output

Replacing Core Classes

To use one of your own system classes instead of a default one simply place your version inside your local application/libraries directory:

application/libraries/some-class.php

If this directory does not exist you can create it.

Any file named identically to one from the list above will be used instead of the one normally used.

Please note that your class must use CI as a prefix. For example, if your file is named Input.php the class will be named:

```
class CI_Input {

}
```

Extending Core Class

If all you need to do is add some functionality to an existing library - perhaps add a function or two - then it's overkill to replace the entire library with your version. In this case it's better to simply extend the class. Extending a class is nearly identical to replacing a class with a couple exceptions:

The class declaration must extend the parent class.

Your new class name and filename must be prefixed with MY_ (this item is configurable. See below.).

For example, to extend the native Input class you'll create a file named application/libraries/MY_Input.php, and declare your class with:

```
class MY_Input extends CI_Input {

}
```

Note: If you need to use a constructor in your class make sure you extend the parent constructor:

```
class MY_Input extends CI_Input {
    function My_Input()
    {
        parent::CI_Input();
    }
}
```

Tip: Any functions in your class that are named identically to the functions in the parent class will be used instead of the native ones (this is known as "method overriding"). This allows you to substantially alter the CodeIgniter core.

Setting Your Own Prefix

To set your own sub-class prefix, open your application/config/config.php file and look for this item:

```
$config['subclass_prefix'] = 'MY_';
```

Please note that all native CodeIgniter libraries are prefixed with CI_ so DO NOT use that as your prefix.

钩子 - 扩展框架的核心

CodeIgniter's Hooks feature provides a means to tap into and modify the inner workings of the framework without hacking the core files. When CodeIgniter runs it follows a specific execution process, diagramed in the Application Flow page. There may be instances, however, where you'd like to cause some action to take place at a particular stage in the execution process. For example, you might want to run a script right before your controllers get loaded, or right after, or you might want to trigger one of your own scripts in some other location.

启用钩子

The hooks feature can be globally enabled/disabled by setting the following item in the application/config/config.php file:

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

Defining a Hook

Hooks are defined in application/config/hooks.php file. Each hook is specified as an array with this prototype:

```
$hook['pre_controller'] = array(
                                'class'    => 'MyClass',
                                'function' => 'Myfunction',
                                'filename' => 'Myclass.php',
                                'filepath' => 'hooks',
                                'params'   => array('beer', 'wine', 'snacks')
                                );
```

Notes:

The array index correlates to the name of the particular hook point you want to use. In the above example the hook point is pre_controller. A list of hook points is found below. The following items should be defined in your associative hook array:

class The name of the class you wish to invoke. If you prefer to use a procedural function instead of a class, leave this item blank.

function The function name you wish to call.

filename The file name containing your class/function.

filepath The name of the directory containing your script. Note: Your script must be located in a directory INSIDE your application folder, so the file path is relative to that folder. For example, if your script is located in application/hooks, you will simply use hooks as your filepath. If your script is located in application/hooks/utilities you will use hooks/utilities as your filepath. No trailing slash.

params Any parameters you wish to pass to your script. This item is optional.

Multiple Calls to the Same Hook

If want to use the same hook point with more then one script, simply make your array declaration multi-dimensional, like this:

```
$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')
                                );
```

Notice the brackets after each array index:

$hook['pre_controller'][]

This permits you to the same hook point with multiple scripts. The order you define your array will be the execution order.

挂勾点

挂勾点可用于下面列表里的东东. :)

pre_system

系统执行的早期调用.仅仅在benchmark 和 hooks 类 加载完毕的时候. 没有执行路由或者其它的东西.

pre_controller

Called immediately prior to any of your controllers being called. All base classes, routing, and security checks have been done.

post_controller_constructor

Called immediately after your controller is instantiated, but prior to any method calls happening.

post_controller

Called immediately after your controller is fully executed.

display_override

Overrides the _display() function, used to send the finalized page to the web browser at the end of system execution. This permits you to use your own display methodology. Note that the finalized data will be available by calling $this->output->get_output()

cache_override

Enables you to call your own function instead of the _display_cache() function in the output class. This permits you to use your own cache display mechanism.

scaffolding_override

Permits a scaffolding request to trigger your own script instead.

post_system

Called after the final rendered page is sent to the browser, at the end of system execution after the finalized data is sent to the browser.

自动加载资源

CodeIgniter有一个自动加载属性它允许在系统运行过程中初始的自动加载库(libraries),帮助函数(helpers),插件(plugins),如果你想要特定的资源全局在你的程序中全局使用,你会发现将他们自动加载是很方便的。

以下列表项目可以自动加载：

核心类在libraries文件夹中

帮助函数文件在helper文件夹中
插件在plugins文件夹中
自定义配置文件在config文件夹中
语言包文件在"system/language"文件夹中

要自动加载资源，先打开application/config/autoload.php文件，然后增加你想要自动加载的项目在自动加载数组中，你能看见文件中的有与上边项目对应的指示。

注意： 在增加项目到自动加载数组中的时候，不要包括文件扩展名(.php)。

脚手架**(Scaffolding)**

CodeIgniter's Scaffolding feature provides a fast and very convenient way to add, edit, or delete information in your database during development.

Very Important: Scaffolding is intended for development use only. It provides very little security other than a "secret" word, so anyone who has access to your CodeIgniter site can potentially edit or delete your information. If you use scaffolding make sure you disable it immediately after you are through using it. DO NOT leave it enabled on a live site. And please, set a secret word before you use it.

Why would someone use scaffolding?

Here's a typical scenario: You create a new database table during development and you'd like a quick way to insert some data into it to work with. Without scaffolding your choices are either to write some inserts using the command line or to use a database management tool like phpMyAdmin. With CodeIgniter's scaffolding feature you can quickly add some data using its browser interface. And when you are through using the data you can easily delete it.

Setting a Secret Word

Before enabling scaffolding please take a moment to set a secret word. This word, when encountered in your URL, will launch the scaffolding interface, so please pick something obscure that no one is likely to guess.

To set a secret word, open your application/config/routes.php file and look for this item:

```
$route['scaffolding_trigger'] = '';
```

Once you've found it add your own unique word.

Note: The scaffolding word can **not** start with an underscore.

Enabling Scaffolding

Note: The information on this page assumes you already know how controllers work, and that you have a working one available. It also assumes you have configured CodeIgniter to auto-connect to your database. If not, the information here won't be very relevant, so you are encouraged to go through those sections first. Lastly, it assumes you understand what a class constructor is. If not, read the last section of the controllers page.

To enable scaffolding you will initialize it in your constructor like this:

```
<?php
class Blog extends Controller {
        function Blog()
        {
                parent::Controller();
                $this->load->scaffolding('table_name');
        }
}
?>
```

Where table_name is the name of the table (table, not database) you wish to work with.

Once you've initialized scaffolding, you will access it with this URL prototype:

www.your-site.com/index.php/class/secret_word/

For example, using a controller named Blog, and abracadabra as the secret word, you would access scaffolding like this:

www.your-site.com/index.php/blog/abracadabra/

The scaffolding interface should be self-explanatory. You can add, edit or delete records.

A Final Note:

The scaffolding feature will only work with tables that contain a primary key, as this is information is needed to perform the various database functions.

URI Routing

Typically there is a one-to-one relationship between a URL string and its corresponding controller class/method. The segments in a URI normally follow this pattern:

www.your-site.com/class/function/id/

In some instances, however, you may want to remap this relationship so that a different class/function can be called instead of the one corresponding to the URL.

For example, lets say you want your URLs to have this prototype:

www.your-site.com/product/1/

www.your-site.com/product/2/
www.your-site.com/product/3/
www.your-site.com/product/4/

Normally the second segment of the URL is reserved for the function name, but in the example above it instead has a product ID. To overcome this, CodeIgniter allows you to remap the URI handler.

Setting your own routing rules

Routing rules are defined in your application/config/routes.php file. In it you'll see an array called $route that permits you to specify your own routing criteria. Routes can either be specified using wildcards or Regular Expressions

Wildcards

A typical wildcard route might look something like this:

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

In a route, the array key contains the URI to be matched, while the array value contains the destination it should be re-routed to. In the above example, if the literal word "product" is found in the first segment of the URL, and a number is found in the second segment, the "catalog" class and the "product_lookup" method are instead used.

You can match literal values or you can use two wildcard types:

:num

:any

:num will match a segment containing only numbers.

:any will match a segment containing any character.

Note: Routes will run in the order they are defined. Higher routes will always take precedence over lower ones.

Examples

Here are a few routing examples:

$route['journals'] = "blogs";

Any URL containing the word "journals" in the first segment will be remapped to the "blogs" class.

$route['blog/joe'] = "blogs/users/34";

Any URL containing the segments blog/joe will be remapped to the "blogs" class and the "users" method. The ID will be set to "34".

$route['product/:any'] = "catalog/product_lookup";

Any URL with "product" as the first segment, and anything in the second will be remapped to the "catalog" class and the "product_lookup" method.

Important: Do not use leading/trailing slashes.

Regular Expressions

If you prefer you can use regular expressions to define your routing rules. Any valid regular expression is allowed, as are back-references.

Note: If you use back-references you must use the dollar syntax rather then the double backslash syntax.

A typical RegEx route might look something like this:

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

In the above example, a URI similar to products/shirts/123 would instead call the shirts controller class and the id_123 function.

You can also mix and match wildcards with regular expressions.

Reserved Routes

There are two reserved routes:

$route['default_controller'] = 'welcome';

This route indicates which controller class should be loaded if the URI contains no data, which will be the case when people load your root URL. In the above example, the "welcome" class would be loaded. You are encouraged to always have a default route otherwise a 404 page will appear by default.

$route['scaffolding_trigger'] = 'scaffolding';

This route lets you set a secret word, which when present in the URL, triggers the scaffolding feature. Please read the Scaffolding page for details.

Important: The reserved routes must come before any wildcard or regular expression routes.

Error Handling

CodeIgniter lets you build error reporting into your applications using the functions described below. In addition, it has an error logging class that permits error and debugging messages to be saved as text files.

Note: By default, CodeIgniter displays all PHP errors. You might wish to change this behavior once your development is complete. You'll find the error_reporting() function located at the top of your main index.php file. Disabling error reporting will NOT prevent log files from being written if there are errors.

Unlike most systems in CodeIgniter, the error functions are simple procedural interfaces that are available globally throughout the application. This approach permits error messages to get triggered without having to worry about class/function scoping.

The following functions let you generate errors:

show_error('message')

This function will display the error message supplied to it using the following error template:

application/errors/error_general.php

show_404('page')

This function will display the 404 error message supplied to it using the following error template:

application/errors/error_404.php

The function expects the string passed to it to be the file path to the page that isn't found. Note that CodeIgniter automatically shows 404 messages if controllers are not found.

log_message('level', 'message')

This function lets you write messages to your log files. You must supply one of three "levels" in the first parameter, indicating what type of message it is (debug, error, info), with the message itself in the second parameter. Example:

```
if ($some_var == "")
{
     log_message('error', 'Some variable did not contain a value.');
}
else
{
     log_message('debug', 'Some variable was correctly set');
}
log_message('info', 'The purpose of some variable is to provide some value.');
```

There are three message types:

Error Messages. These are actual errors, such as PHP errors or user errors.

Debug Messages. These are messages that assist in debugging. For example, if a class has been initialized, you could log this as debugging info.

Informational Messages. These are the lowest priority messages, simply giving information regarding some process. CodeIgniter doesn't natively generate any info messages but you may want to in your application.

Note: In order for the log file to actually be written, the "logs" folder must be writable. In addition, you must set the "threshold" for logging. You might, for example, only want error messages to be logged, and not the other two types. If you set it to zero logging will be disabled.

Web Page Caching

CodeIgniter lets you cache your pages in order to achieve maximum performance.

Although CodeIgniter is quite fast, the amount of dynamic information you display in your pages will correlate directly to the server resources, memory, and processing cycles utilized, which affect your page load speeds. By caching your pages, since they are saved in their fully rendered state, you can achieve performance that nears that of static web pages.

How Does Caching Work?

Caching can be enabled on a per-page basis, and you can set the length of time that a page should remain cached before being refreshed. When a page is loaded for the first time, the cache file will be written to your system/cache folder. On subsequent page loads the cache file will be retrieved and sent to the requesting user's browser. If it has expired, it will be deleted and refreshed before being sent to the browser.

Note: The Benchmark tag is not cached so you can still view your page load speed when caching is enabled.

Enabling Caching

To enable caching, put the following tag in any of your controller functions:

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

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

The above tag can go anywhere within a function. It is not affected by the order that it appears, so place it wherever it seems most logical to you. Once the tag is in place, your pages will begin being cached.

Note: Before the cache files can be written you must set the file permissions on your system/cache folder such that it is writable (666 is usually appropriate).

Deleting Caches

If you no longer wish to cache a file you can remove the caching tag and it will no longer be refreshed when it expires. Note: Removing the tag will not delete the cache immediately. It will have to expire normally. If you need to remove it earlier you will need to manually delete it from your cache folder.

Profiling Your Application

The Profiler Class will display benchmark results, queries you have run, and $_POST data at the bottom of your pages. This information can be useful during development in order to help with debugging and optimization.

Initializing the Class

Important: This class does NOT need to be initialized. It is loaded automatically by the Output Class if profiling is enabled as shown below.

Enabling the Profiler

To enable the profiler place the following function anywhere within your Controller functions:

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

When enabled a report will be generated and inserted at the bottom of your pages.

To disable the profiler you will use:

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

Setting Benchmark Points

In order for the Profiler to compile and display your benchmark data you must name your mark points using specific syntax. Please read the information on setting Benchmark points in Benchmark Class page.

Managing your Applications

By default it is assumed that you only intend to use CodeIgniter to manage one application, which you will build in your system/application/ directory. It is possible, however, to have multiple sets of applications that share a single CodeIgniter installation, or even to rename or relocate your application folder.

Renaming the Application Folder

If you would like to rename your application folder you may do so as long as you open your main index.php file and set its name using the $application_folder variable:

```
$application_folder = "application";
```

Relocating your Application Folder

It is possible to move your application folder to a different location on your server than your system folder. To do so open your main index.php and set a *full server path* in the $application_folder variable.

$application_folder = "/Path/to/your/application";

Running Multiple Applications with one CodeIgniter Installation

If you would like to share a common CodeIgniter installation to manage several different applications simply put all of the directories located inside your application folder into their own sub-folder.

For example, let's say you want to create two applications, "foo" and "bar". You will structure your application folder like this:

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/

To select a particular application for use requires that you open your main index.php file and set the $application_folder variable. For example, to select the "foo" application for use you would do this:

$application_folder = "application/foo";

Note: Each of your applications will need its own index.php file which calls the desired application. The index.php file can be named anything you want.

视图文件的 **PHP** 替代语法

如果你不使用 CodeIgniter 的模板引擎，则你可以在视图文件中使用原始 PHP 代码。要使 PHP 代码达到最精简并使其更容易辨认，因此建议你使用 PHP 替代语法控制结构及短标记的 echo 语句。 建议您使用 PHP 的语法为您的控制结构和简短的输出标签。如果你不熟悉这个语法，

它允许你从你的代码中消灭大括号，并且消灭“echo”语句。

自动短标记支持

注：如果你发现本页描述的语法在你的服务器上不工作，它可能是“短标记”，并且在你的 PHP ini 文件中禁用了。CodeIgniter 可以重写所有短标记，让你使用这个语法即使你的服务器不支持它。这个特性可以在你的 config/config.php 文件中打开。

请注意，如果你使用这个特性，如果 PHP 错误发生在你的**视图文件**中，则错误信息和行号将无法准确显示。相反，所有的错误将显示为 eval () 的错误。

替代 Echo

正常的 echo 和 print 输出一般是这样的形式：

```
<?php echo $variable; ?>
```

使用替代语法，你能改成这样的形式：

```
<?=$variable?>
```

替代控制结构

控制结构，像 if，for，foreach，和 while 也可以写成简化的形式。这里是一个用 foreach 的例子：

```
<ul>

<?php foreach($todo as $item): ?>
<li><?=$item?></li>
<?php endforeach; ?>
</ul>
```

注意，这里没有大括号。相反，结束大括号被替换成了 endforeach。上面列出的每一个控制结构也有相似的关闭语法：endif，endfor，endforeach，和 endwhile

并且在每个结构以后注意不要使用分号(除了最后一个)，用冒号。这是很重要的！

这有另一个例子，使用 if/elseif/else。注意冒号：

```
<?php if ($username == 'sally'): ?>

    <h3>Hi Sally</h3>
<?php elseif ($username == 'joe'): ?>
    <h3>Hi Joe</h3>
<?php else: ?>
    <h3>Hi unknown user</h3>
<?php endif; ?>
```

Security

This page describes some "best practices" regarding web security, and details CodeIgniter's internal security features.

URI Security

CodeIgniter is fairly restrictive regarding which characters it allows in your URI strings in order to help minimize the possibility that malicious data can be passed to your application. URIs may only contain the following:

- Alpha-numeric text
- Tilde: ~
- Period: .
- Colon: :
- Underscore: _
- Dash: -

GET, POST, and COOKIE Data

GET data is simply disallowed by CodeIgniter since the system utilizes URI segments rather than traditional URL query strings (unless you have the query string option enabled in your config file). The global GET array is **unset** by the Input class during system initialization.

Register_globals

During system initialization all global variables are unset, except those found in the $_POST and $_COOKIE arrays. The unsetting routine is effectively the same as register_globals = off.

magic_quotes_runtime

The magic_quotes_runtime directive is turned off during system initialization so that you don't have to remove slashes when retrieving data from your database.

Best Practices

Before accepting any data into your application, whether it be POST data from a form submission, COOKIE data, URI data, XML-RPC data, or even data from the SERVER array, you are encouraged to practice this three step approach:

1. Filter the data as if it were tainted.
2. Validate the data to ensure it conforms to the correct type, length, size, etc. (sometimes this step can replace step one)
3. Escape the data before submitting it into your database.

CodeIgniter provides the following functions to assist in this process:

XSS Filtering

CodeIgniter comes with a Cross Site Scripting filter. This filter looks for commonly used techniques to embed malicious Javascript into your data, or other types of code that attempt to hijack cookies or do other malicious things. The XSS Filter is described here.

Validate the data

CodeIgniter has a Validation Class that assists you in validating, filtering, and prepping your data.

Escape all data before database insertion

Never insert information into your database without escaping it. Please see the section that discusses queries for more information.

基准测试类

CodeIgniter 有一个总是有效的基准测试类,能够用来计算两个标记点的时间差.

注意:这个类会被系统自动初始化,因此不需要手动初始化.

另外,基准测试类在框架被调用的时候开始,在最终视图被output类送给浏览器之前结束,可以显示整个系统执行的精确定时.

目录

使用基准测试类

测试基准类可以在 控制器, 视图,或者 模型.中使用,用法如下:

标记一个开始点

标记一个结束点
运行elapsed_time函数显示结果

下面是一个代码示例:

```
$this->benchmark->mark('code_start');

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

注意:单词 "code_start" and "code_end" 是任意的,他们是简单的单词用来做为两个标记.你可以使用你想用的任意单词,并且你可以设置多个标记,参考下面的这些代码:

```
$this->benchmark->mark('dog');
// Some code happens here
$this->benchmark->mark('cat');
// More code happens here
$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');
```

自定义你的基准测试类

如果你想你的基准数据对评测有效,你的标记点必须设置成对,并且每个标记点必须用_start 和_end结束.每一对标记点的前部必须相同.例如:

```
$this->benchmark->mark('my_mark_start');
// Some code happens here...
$this->benchmark->mark('my_mark_end');
$this->benchmark->mark('another_mark_start');
// Some more code happens here...
$this->benchmark->mark('another_mark_end');
```

更多信息请参考 Profiler page .

显示总的执行时间

如果你想显示从CodeIgniter启动到浏览器最终输出的时间消耗,简单把这段代码放到你的一个视图模板中:

```
<?=$this->benchmark->elapsed_time();?>
```

你会注意到这个函数和上面例子中计算两个标记点时间差的函数是同一个, 不同的是这里你没有使用参数.当参数为空的时候,CodeIgniter一直会到最终页面被送往浏览器之前才停止benchmark .它不管你是在哪里调用的,计时器会持续到最终结束.

如果不喜欢使用纯PHP,另外一种备用的来显示时间消耗的方式是在视图文件中使用这个伪变量:

```
{elapsed_time}
```

注意:如果你想在你的控制器函数中benchmark(基准测试)一些东西, 你必须设置你自己的开始/结束点.

显示内存消耗

如果你的PHP在安装的时候被配置成--enable-memory-limit,你可以将下面的代码包含到视图文

件中显示整个系统的内存使用量:

<?=$this->benchmark->memory_usage();?>

注意:这个函数只能在视图文件中使用.这个消耗量就是应用程序的全部内存消耗.

如果不喜欢使用纯PHP,另外一种备用的来显示内存使用量的方式是在视图文件中使用这个伪变量:

{memory_usage}

Calendaring Class

日历类可以让你动态创建创建日历控件.并且用日历模板对创建的日历控件格式化,100%的控制它的样式.另外,你可以传送数据到你的日历单元格中(比如创建一个链接).

初始化类

和大多数其他CI中的类一样,在控制器中初始化日历类用$this->load->library函数:

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

导入之后,日历类可以这样使用: $this->calendar

显示一个日历控件

这是一个简单的例子告诉你如何去显示一个日历控件:

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

echo $this->calendar->generate();

上面的代码将根据你服务器时间创建一个当前月/年的日历控件. 要显示一个指定月和年的日历控件，你要传递这些信息到日历生成函数:

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

echo $this->calendar->generate(2006, 6);

上面的代码将创建一个显示2006年6月的控件.第一个参数指定了年，第二个参数指定了月.

传数据到单元格

增加数据到日历的单元格就要创建一个关联数组,在这个数组中索引是你想链接的天数value值包含你要传入的值.数组通过日历创建函数的第三个参数被传入. 参考下面这个例子:

```
$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);
```

使用上面的例子，天数3,7,13和26将变成链接指向你提供的URLs.

注意: 默认情况,系统假定你的数组中已经包含了链接. 在下面解释日历模板部分你会看到你可以自定义数据如何被传入日历单元格以便你可以传不同类型的信息.

设置偏好

有7中偏好可以让你设置日历控件的各个方面. 偏好被设置成数组通过导入函数的第二个参数被导入. 下面是一个例子:

```
$prefs = array (
                'start_day'     => 'saturday',
                'month_type'    => 'long',
                'day_type'      => 'short'
            );

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

上面的代码将从礼拜六开始,用"长"月标题和"短"天数格式.更多关于偏好的信息请看下面.

Preference	Default Value	Options	Description
template	None	None	A string containing your calendar template. See the template section below.
local_time	time()	None	A Unix timestamp corresponding to the current time.
start_day	sunday	Any week day (sunday, monday, tuesday, etc.)	Sets the day of the week the calendar should start on.
month_type	long	long, short	Determines what version of the month name to use in the header. long = January, short = Jan.
day_type	abr	long, short, abr	Determines what version of the weekday names to use in the column headers. long = Sunday, short = Sun, abr = Su.
show_next_prev	FALSE	TRUE/FALSE (boolean)	Determines whether to display links allowing you to toggle to next/previous months. See information on this feature below.
next_prev_url	None	A URL	Sets the basepath used in the next/previous calendar links.

Showing Next/Previous Month Links显示Next/Previous链接

要让你的日历控件通过next/previous链接动态的减少/增加,可以仿照下面的例子建立你的日历控件:

```
$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));
```

在上面的例子中，你会注意到这几点:

- 你必须把"show_next_prev"设置成TRUE.
- 你必须在"next_prev_url"偏好中向包含你日历的控件提供URL.
- 你必须向日历创建函数提供"年"和"月"通过他们应该出现的URI段 (注意: 日历类自动添加年/月到你提供的base URL上.).

创建一个日历模板

通过创建一个日历模板你能够100%的控制界面设计. 日历的每一部分都要被放在一对伪变量中,像下面这样:

```
$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="&lt;&lt;</a></th>{/heading_previous_cell}
    {heading_title_cell}<th colspan="{heading}</th>{/heading_title_cell}
    {heading_next_cell}<th><a href="&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="{day}</a>{/cal_cell_content}
    {cal_cell_content_today}
      <div class="highlight"><a href="{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();
```

配置类

该配置类提供一种方法来进行偏好设置。该偏好来源于默认的配置文件 (application/config/config.php) 或您自己定制的配置文件。

注意：该类已经自动加载而无需手动启用。

分析配置文件

默认情况下，CodeIgniter已经有一个主要的配置文件，位于application/config/config.php。如果你用文本编辑器打开你会看到配置项目被存储在一个叫$config的数组里。

您可以添加您自己的配置项目到这个文件里，或者您更愿意让您自己的配置项目与原配置项目分开（assuming you even need config items），简单的创建一个文件并保存到config这个文件夹里就行了。

提示: 如果你想建立一个和主要配置文件一样格式的配置文件，把你的配置项目建立在一个名为$config的数组中. CodeIgniter will intelligently manage these files so there will be no conflict even though the array has the same name (assuming an array index is not named the same as another).

Loading a Config File

Note: CodeIgniter automatically loads the primary config file (application/config/config.php), so you will only need to load a config file if you have created your own.

There are two ways to load a config file:

Manual Loading

To load one of your custom config files you will use the following function within the controller that needs it:

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

Where filename is the name of your config file, without the .php file extension.

If you need to load multiple config files normally they will be merged into one master config array. Name collisions can occur, however, if you have identically named array indexes in different config files. To avoid collisions you can set the second parameter to TRUE and each config file will be stored in an array index corresponding to the name of the config file. Example:

```
// Stored in an array with this prototype: $this->config['blog_settings'] = $config

$this->config->load('blog_settings', TRUE);
```

Please see the section entitled Fetching Config Items below to learn how to retrieve config items set this way.

The third parameter allows you to suppress errors in the event that a config file does not exist:

```
$this->config->load('blog_settings', FALSE, TRUE);
```

Auto-loading

If you find that you need a particular config file globally, you can have it loaded automatically by the system. To do this, open the **autoload.php** file, located at application/config/autoload.php, and add your config file as indicated in the file.

Fetching Config Items

To retrieve an item from your config file, use the following function:

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

Where item name is the $config array index you want to retrieve. For example, to fetch your language choice you'll do this:

```
$lang = $this->config->item('language');
```

The function returns FALSE (boolean) if the item you are trying to fetch does not exist.

If you are using the second parameter of the $this->config->load function in order to assign your config items to a specific index you can retrieve it by specifying the index name in the second parameter of the $this->config->item() function. Example:

```
// Loads a config file named blog_settings.php and assigns it to an index named "blog_settings"

$this->config->load('blog_settings', 'TRUE');
// Retrieve a config item named site_name contained within the blog_settings array
$site_name = $this->config->item('site_name', 'blog_settings');
// An alternate way to specify the same item:
$blog_config = $this->config->item('blog_settings');
$site_name = $blog_config['site_name'];
```

Setting a Config Item

If you would like to dynamically set a config item or change an existing one, you can so using:

```
$this->config->set_item('item_name', 'item_value');
```

Where item_name is the $config array index you want to change, and item_value is its value.

Helper Functions

The config class has the following helper functions:

$this->config->site_url();

This function retrieves the URL to your site, along with the "index" value you've specified in the config file.

$this->config->system_url();

This function retrieves the URL to your system folder.

数据库类

CodeIgniter 内置了速度快、功能强大的数据库类作为数据库的中间抽象层。数据库类支持传统架构以及 Active Record 架构。类中的数据库函数使用简单明了的语法。

数据库快速入门例子代码

下面的内容将简单说明怎样使用数据库。更详细的信息请阅读各个函数的单独介绍页面。

初始化数据库类

下面的代码将依据你的数据库配置载入并初始化数据库类:

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

一旦被载入，你可以在任何地方像这样使用它：

注意: 如果你的所有页面均要求初始化数据库类，你可以让它自动加载。详见 数据库连接。

多结果标准查询（对象形式）

```
$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 'Total Results: ' . $query->num_rows();
```

上面的result()函数返回一个对象的数组。例如：$row->title

多结果标准查询（数组形式）

```
$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'];
}
```

上面的result_array()函数返回一个带下标的数组。例如：$row['title']

测试查询结果

如果你的查询可能不返回结果，我们建议你先使用 num_rows()函数来测试:

```
$query = $this->db->query("YOUR QUERY");
if ($query->num_rows() > 0)
{
    foreach ($query->result() as $row)
    {
        echo $row->title;
        echo $row->name;
        echo $row->body;
    }
}
```

单结果标准查询（对象形式）

```
$query = $this->db->query('SELECT name FROM my_table LIMIT 1');
$row = $query->row();
echo $row->name;
```

上面的row()函数返回一个 对象。例如：$row->name

单结果标准查询（数组形式）

```
$query = $this->db->query('SELECT name FROM my_table LIMIT 1');

$row = $query->row_array();
echo $row['name'];
```
上面的row_array()函数返回一个 数组。例如：$row['name']

标准插入(insert)

```
$sql = "INSERT INTO mytable (title, name)

            VALUES (".$this->db->escape($title).", ".$this->db->escape($name).")";
$this->db->query($sql);
echo $this->db->affected_rows();
```

快捷查询

快捷查询类能为我们提供快速取得数据的途径:

```
$query = $this->db->get('table_name');

foreach ($query->result() as $row)
{
    echo $row->title;
}
```
上面的get()函数返回数据表中所有的结果。 快捷查询类 提供所有数据库操作的快捷函数。

快捷插入(insert)

```
$data = array(

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

Database Configuration

CodeIgniter 有一个配置文件让你存放数据库连接值（username:用户名，password:密码，database name:数据库名，等等..）. 配置文件位于以下路径:

application/config/database.php

配件文件存放在一个如下格式的一个多维数组里:

```
$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']['active_r'] = TRUE;
```

我们使用多维数组的原因是为了让你随意的存储多个连接值的设置。举例：如果你运行多个环境（development：开发、production：制作、test：测试 等等..），你能为每个环境建立独立的连接组，并在组直接进行切换。举例，设置一个"test"环境，你可以这样做：

```
$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']['active_r'] = TRUE;
```

那么，告诉系统使用"test"组，你可以设置位于配置文件中的变量：

```
$active_group = "test";
```

注意: "test"的名字是任意的，这可以让你自由设置，我们的主要连接默认使用"default"这个名字，当然，您可以基于您的项目为它起一个更有意义的名字。

参数解析:

hostname - 数据库的主机名，通常位于本机，可以表示为 "localhost".

username - 需要连接到数据库的用户名.
password - 登陆数据库的密码.
database - 你需要连接的数据库名.
dbdriver - 数据库类型. 如: mysql, postgre, obdc, 等等. 按规定必须指定.
dbprefix - 当运行Active Record查询时数据表的前缀,它允许在一个数据库上安装多个CodeIgniter程序.
pconnect - TRUE/FALSE (boolean) - 使用持续连接.
db_debug - TRUE/FALSE (boolean) - 显示数据库错误信息.
active_r - TRUE/FALSE (boolean) - 加载Active Record Class. 为了在初始化database classes 时使用更多的资源,如果你不使用 active record class 可以忽略.

port - 数据库端口号. Currently only used with the Postgre driver.

提示: 并不是所有的值都是必须的,这取决与您所使用的数据库平台,如(MySQL, Postgre, 等.) 例如, 当你使用SQLite时,你不需要提供username 或 password, 数据库名字就是您数据库文件的路径. 以上内容假定您使用的是 MySQL 数据库.

Connecting to your Database

There are two ways to connect to a database:

Automatically Connecting

The "auto connect" feature will load and instantiate the database class with every page load. To enable "auto connecting", add the word database to the core array, as indicated in the following file:

application/config/autoload.php

Manually Connecting

If only some of your pages require database connectivity you can manually connect to your database by adding this line of code in any function where it is needed, or in your class constructor to make the database available globally in that class.

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

If the above function does **not** contain any information in the first parameter it will connect to the group specified in your database config file. For most people, this is the preferred method of use.

The first parameter of this function can **optionally** be used to specify a particular database group from your config file, or you can even submit connection values for a database that is not specified in your config file. Examples:

To choose a specific group from your config file you can do this:

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

Where group_name is the name of the connection group from your config file.

To connect manually to a desired database you can pass an array of values:

```
$config['hostname'] = "localhost";

$config['username'] = "myusername";
$config['password'] = "mypassword";
$config['database'] = "mydatabase";
$config['dbdriver'] = "mysql";
$config['dbprefix'] = "";
$config['pconnect'] = FALSE;
$config['db_debug'] = TRUE;
$config['active_r'] = TRUE;
```

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

For information on each of these values please see the configuration page.

Or you can submit your database values as a Data Source Name. DSNs must have this prototype:

$dsn = 'dbdriver://username:password@hostname/database';

$this->load->database('$dsn');

Note that if you use a DSN you will not be able to specify some of the default values like you can if you use a connection array.

Connecting to Multiple Databases

If you need to connect to more than one database simultaneously you can do so as follows:

$DB1 = $this->load->database('group_one', TRUE);

$DB2 = $this->load->database('group_two', TRUE);

Note: Change the words "group_one" and "group_two" to the specific group names you are connecting to (or you can pass the connection values as indicated above).

By setting the second parameter to TRUE (boolean) the function will return the database object.

When you connect this way, you will use your object name to issue commands rather than the syntax used throughout this guide. In other words, rather than issuing commands with:

$this->db->query();

$this->db->result();
etc...

You will instead use:

$DB1->query();

$DB1->result();
etc...

Queries

$this->db->query();

To submit a query, use the following function:

$this->db->query('YOUR QUERY HERE');

The query() function returns a database result **object** when "read" type queries are run, which you can use to show your results. When "write" type queries are run it simply returns TRUE or FALSE depending on success or failure. When retrieving data you will typically assign the query to your own variable, like this:

$query = $this->db->query('YOUR QUERY HERE');

$this->db->simple_query();

This is a simplified version of the $this->db->query() function. It ONLY returns TRUE/FALSE on success or failure. It DOES NOT return a database result set, nor does it set the query timer, or compile bind data, or store your query for debugging. It simply lets you submit a query. Most users will rarely use this function.

Escaping Queries

It's a very good security practice to escape your data before submitting it into your database. CodeIgniter has two functions that help you do this:

$this->db->escape() This function determines the data type so that it can escape only string data. It also automatically adds single quotes around the data so you don't have to: $sql = "INSERT INTO table (title) VALUES(".$this->db->escape($title).")";

$this->db->escape_str() This function escapes the data passed to it, regardless of type. Most of the time you'll use the above function rather then this one. Use the function like this: $sql = "INSERT INTO table (title) VALUES('".$this->db->escape_str($title)."')";

Query Bindings

Bindings enable you to simplify your query syntax by letting the system put the queries together for you. Consider the following example:

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

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

The question marks in the query are automatically replaced with the values in the array in the second parameter of the query function.

The secondary benefit of using binds is that the values are automatically escaped, producing safer queries. You don't have to remember to manually escape data; the engine does it automatically for you.

Generating Query Results

There are several ways to generate query results:

result()

This function returns the query result as an array of **objects**, or **an empty array** on failure. Typically you'll use this in a foreach loop, like this:

$query = $this->db->query("YOUR QUERY");

```
foreach ($query->result() as $row)
{
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

The above function is an alias of result_object().

If you run queries that might **not** produce a result, you are encouraged to test the result first:

```
$query = $this->db->query("YOUR QUERY");

if ($query->num_rows() > 0)
{
    foreach ($query->result() as $row)
    {
        echo $row->title;
        echo $row->name;
        echo $row->body;
    }
}
```

result_array()

This function returns the query result as a pure array, or an empty array when no result is produced. Typically you'll use this in a foreach loop, like this:

```
$query = $this->db->query("YOUR QUERY");

foreach ($query->result_array() as $row)
{
    echo $row['title'];
    echo $row['name'];
    echo $row['body'];
}
```

row()

This function returns a single result row. If your query has more than one row, it returns only the first row. The result is returned as an **object**. Here's a usage example:

```
$query = $this->db->query("YOUR QUERY");

if ($query->num_rows() > 0)
{
    $row = $query->row();
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

If you want a specific row returned you can submit the row number as a digit in the first parameter:

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

row_array()

Identical to the above row() function, except it returns an array. Example:

```
$query = $this->db->query("YOUR QUERY");

if ($query->num_rows() > 0)
{
    $row = $query->row_array();

    echo $row['title'];
    echo $row['name'];
    echo $row['body'];
}
```

If you want a specific row returned you can submit the row number as a digit in the first parameter:

```
$row = $query->row_array(5);
```

In addition, you can walk forward/backwards/first/last through your results using these variations:

$row = $query->first_row()

$row = $query->last_row()
$row = $query->next_row()
$row = $query->previous_row()

By default they return an object unless you put the word "array" in the parameter:

$row = $query->first_row('array')

$row = $query->last_row('array')
$row = $query->next_row('array')
$row = $query->previous_row('array')

Result Helper Functions

$query->num_rows()

The number of rows returned by the query. Note: In this example, $query is the variable that the query result object is assigned to:

```
$query = $this->db->query('SELECT * FROM my_table');

echo $query->num_rows();
```

$query->num_fields()

The number of FIELDS (columns) returned by the query. Make sure to call the function using your query result object:

```
$query = $this->db->query('SELECT * FROM my_table');

echo $query->num_fields();
```

$query->free_result()

It frees the memory associated with the result and deletes the result resource ID. Normally PHP frees its memory automatically at the end of script execution. However, if you are running a lot of queries in a particular script you might want to free the result after each query result has been generated in order to cut down on memory consumptions. Example:

```
$query = $this->db->query('SELECT title FROM my_table');

foreach ($query->result() as $row)
{
    echo $row->title;
}
$query->free_result(); // The $query result object will no longer be available
$query2 = $this->db->query('SELECT name FROM some_table');
$row = $query2->row();
echo $row->name;
$query2->free_result(); // The $query2 result object will no longer be available
```

Query Helper Functions

$this->db->insert_id()

The insert ID number when performing database inserts.

$this->db->affected_rows()

Displays the number of affected rows, when doing "write" type queries (insert, update, etc.).

Note: In MySQL "DELETE FROM TABLE" returns 0 affected rows. The database class has a small hack that allows it to return the correct number of affected rows. By default this hack is enabled but it can be turned off in the database driver file.

$this->db->count_all();

Permits you to determine the number of rows in a particular table. Submit the table name in the first parameter. Example:

echo $this->db->count_all('my_table');

// Produces an integer, like 25

$this->db->platform()

Outputs the database platform you are running (MySQL, MS SQL, Postgre, etc...):

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

$this->db->version()

Outputs the database version you are running:

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

$this->db->last_query();

Returns the last query that was run (the query string, not the result). Example:

$str = $this->db->last_query();

// Produces: SELECT * FROM sometable....

The following two functions help simplify the process of writing database INSERTs and UPDATEs.

$this->db->insert_string();

This function simplifies the process of writing database inserts. It returns a correctly formatted SQL insert string. Example:

$data = array('name' => $name, 'email' => $email, 'url' => $url);

$str = $this->db->insert_string('table_name', $data);

The first parameter is the table name, the second is an associative array with the data to be inserted. The above example produces:

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

Note: Values are automatically escaped, producing safer queries.

$this->db->update_string();

This function simplifies the process of writing database updates. It returns a correctly formatted SQL update string. Example:

```
$data = array('name' => $name, 'email' => $email, 'url' => $url);
```

```
$where = "author_id = 1 AND status = 'active'";
$str = $this->db->update_string('table_name', $data, $where);
```

The first parameter is the table name, the second is an associative array with the data to be inserted, and the third parameter is the "where" clause. The above example produces:

```
UPDATE exp_weblog SET name = 'Rick', email = 'rick@your-site.com', url = 'www.your-site.com'
WHERE author_id = 1 AND status = 'active'
```

Note: Values are automatically escaped, producing safer queries.

Active Record 类

CodeIgniter uses a modified version of the Active Record Database Pattern. This pattern allows information to be retrieved, inserted, and updated in your database with minimal scripting. In some cases only one or two lines of code are necessary to perform a database action. CodeIgniter does not require that each database table be its own class file. It instead provides a more simplified interface.

Beyond simplicity, a major benefit to using the Active Record features is that it allows you to create database independent applications, since the query syntax is generated by each database adapter. It also allows for safer queries, since the values are escaped automatically by the system.

Note: If you intend to write your own queries you can disable this class in your database config file, allowing the core database library and adapter to utilize fewer resources.

选择数据

The following functions allow you to build SQL **SELECT** statements.

Note: If you are using PHP 5 you can use method chaining for more compact syntax. This is described at the end of the page.

$this->db->get();

Runs the selection query and returns the result. Can be used by itself to retrieve all records from a table:

```
$query = $this->db->get('mytable');
```

```
// Produces: SELECT * FROM mytable
```

The second and third parameters enable you do set a limit and offset clause:

```
$query = $this->db->get('mytable', 10, 20);
```

// Produces: SELECT * FROM mytable LIMIT 20, 10 (in MySQL. Other databases have slightly different syntax)

You'll notice that the above function is assigned to a variable named $query, which can be used to show the results:

```
$query = $this->db->get('mytable');

foreach ($query->result() as $row)
{
    echo $row->title;
}
```

Please visit the result functions page for a full discussion regarding result generation.

$this->db->getwhere();

Identical to the above function except that it permits you to add a "where" clause in the second parameter, instead of using the db->where() function:

```
$query = $this->db->getwhere('mytable', array('id' => $id), $limit, $offset);
```

Please read the about the where function below for more information.

$this->db->select();

Permits you to write the SELECT portion of your query:

```
$this->db->select('title, content, date');

$query = $this->db->get('mytable');
// Produces: SELECT title, content, date FROM mytable
```

Note: If you are selecting all (*) from a table you do not need to use this function. When omitted, CodeIgniter assumes you wish to SELECT *

$this->db->from();

Permits you to write the FROM portion of your query:

```
$this->db->select('title, content, date');

$this->db->from('mytable');
$query = $this->db->get();
// Produces: SELECT title, content, date FROM mytable
```

Note: As shown earlier, the FROM portion of your query can be specified in the $this->db->get() function, so use whichever method you prefer.

$this->db->join();

Permits you to write the JOIN portion of your query:

```
$this->db->select('*');
$this->db->from('blogs');
$this->db->join('comments', 'comments.id = blogs.id');
$query = $this->db->get();
// Produces:
// SELECT * FROM blogs
// JOIN comments ON comments.id = blogs.id
```

Multiple function calls can be made if you need several joins in one query.

If you need something other than a natural JOIN you can specify it via the third parameter of the function. Options are: left, right, outer, inner, left outer, and right outer.

$this->db->join('comments', 'comments.id = blogs.id', **'left'**);

// Produces: LEFT JOIN comments ON comments.id = blogs.id

$this->db->where();

This function enables you to set **WHERE** clauses using one of four methods:

Note: All values passed to this function are escaped automatically, producing safer queries.

Simple key/value method: $this->db->where('name', $name);

// Produces: WHERE name = 'Joe'

Notice that the equal sign is added for you.

If you use multiple function calls they will be chained together with AND between them:

```
$this->db->where('name', $name);
$this->db->where('title', $title);
$this->db->where('status', $status);
// WHERE = 'Joe' AND title = 'boss' AND status = 'active'
```

Custom key/value method:

You can include an operator in the first parameter in order to control the comparison:

```
$this->db->where('name !=', $name);
$this->db->where('id <', $id);
// Produces: WHERE name != 'Joe' AND id < 45
```

Associative array method: $array = array('name' => $name, 'title' => $title, 'status' => $status);
$this->db->where($array);
// Produces: WHERE name = 'Joe' AND title = 'boss' AND status = 'active'

You can include your own operators using this method as well:

$array = array('name !=' => $name, 'id <' => $id, 'date >' => $date);

$this->db->where($array);

Custom string:

You can write your own clauses manually:

$where = "name='Joe' AND status='boss' OR status='active'";

$this->db->where($where);

$this->db->orwhere();

This function is identical to the one above, except that multiple instances are joined by OR:

$this->db->where('name !=', $name);

$this->db->orwhere('id >', $id);
// Produces: WHERE name != 'Joe' OR id > 50

$this->db->like();

This function enables you to generate **LIKE** clauses, useful for doing searches.

Note: All values passed to this function are escaped automatically.

Simple key/value method: $this->db->like('title', $match);

// Produces: WHERE title LIKE '&#xma;tch%'

If you use multiple function calls they will be chained together with AND between them:

$this->db->like('title', $match);

$this->db->like('body', $match);
// WHERE title LIKE '&#xma;tch%' AND body LIKE '&#xma;tch%'

Associative array method: $array = array('title' => $match, 'page1' => $match, 'page2' => $match);
$this->db->like($array);
// WHERE title LIKE '&#xma;tch%' AND page1 LIKE '&#xma;tch%' AND page2 LIKE '&#xma;tch%'

$this->db->orlike();

This function is identical to the one above, except that multiple instances are joined by OR:

$this->db->like('title', $match);

$this->db->orlike('body', $match);
// WHERE title LIKE '&#xma;tch%' OR body LIKE '&#xma;tch%'

$this->db->groupby();

Permits you to write the GROUP BY portion of your query:

```
$this->db->groupby("title");

// Produces: GROUP BY title
```

You can also pass an array of multiple values as well:

```
$this->db->groupby(array("title", "date");

// Produces: GROUP BY title, date
```

$this->db->having();

Permits you to write the HAVING portion of your query:

```
$this->db->having('user_id = 45');

// Produces: HAVING 'user_id = 45'
```

You can also pass an array of multiple values as well:

```
$this->db->having(array('title =' => 'My Title', 'id <' => $id));

// Produces: HAVING title = 'My Title', 'id < 45'
```

$this->db->orderby();

Lets you set an ORDER BY clause. The first parameter contains the name of the column you would like to order by. The second parameter lets you set the direction of the result. Options are asc or desc or RAND()

```
$this->db->orderby("title", "desc");

// Produces: ORDER BY title DESC
```

You can also pass your own string in the first parameter:

```
$this->db->orderby('title desc, name asc');

// Produces: ORDER BY title DESC, name ASC
```

$this->db->limit();

Lets you limit the number of rows you would like returned by the query:

```
$this->db->limit(10);

// Produces: LIMIT 10
```

The second parameter lets you set a result offset.

```
$this->db->limit(10, 20);

// Produces: LIMIT 20, 10 (in MySQL. Other databases have slightly different syntax)
```

$this->db->count_all();

Permits you to determine the number of rows in a particular table. Submit the table name in the first parameter. Example:

```
echo $this->db->count_all('my_table');

// Produces an integer, like 25
```

插入数据

$this->db->insert();

Generates an insert string based on the data you supply, and runs the query. You can either pass an **array** or an **object** to the function. Here is an example using an array:

```
$data = array(
               'title' => 'My title' ,
               'name' => 'My Name' ,
               'date' => 'My date'
            );
$this->db->insert('mytable', $data);
// Produces: INSERT INTO mytable (title, name, date) VALUES ('My title', 'My name', 'My date')
```

The first parameter will contain the table name, the second is an associative array of values.

Here is an example using an object:

```
/*
    class Myclass {
        var $title = 'My Title';
        var $content = 'My Content';
        var $date = 'My Date';
    }
*/
$object = new Myclass;
$this->db->insert('mytable', $object);
// Produces: INSERT INTO mytable (title, content, date) VALUES ('My Title', 'My Content', 'My Date')
```

The first parameter will contain the table name, the second is an associative array of values.

Note: All values are escaped automatically producing safer queries.

$this->db->set();

This function enables you to set values for inserts or updates.

It can be used instead of passing a data array directly to the insert or update functions:

```
$this->db->set('name', $name);
```

```
$this->db->insert('mytable');
// Produces: INSERT INTO mytable (name) VALUES ('{$name}')
```

If you use multiple function called they will be assembled properly based on whether you are doing an insert or an update:

```
$this->db->set('name', $name);
$this->db->set('title', $title);
$this->db->set('status', $status);
$this->db->insert('mytable');
```

You can also pass an associative array to this function:

```
$array = array('name' => $name, 'title' => $title, 'status' => $status);
$this->db->set($array);
$this->db->insert('mytable');
```

Or an object:

```
/*
    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');
```

更新数据

$this->db->update();

Generates an update string and runs the query based on the data you supply. You can pass an **array** or an **object** to the function. Here is an example using an array:

```
$data = array(
                'title' => $title,
                'name' => $name,
                'date' => $date
            );
$this->db->where('id', $id);
$this->db->update('mytable', $data);
// Produces:
```

```
// UPDATE mytable
// SET title = '{$title}', name = '{$name}', date = '{$date}'
// WHERE id = $id
```

Or you can supply an object:

```
/*
    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
```

Note: All values are escaped automatically producing safer queries.

You'll notice the use of the $this->db->where() function, enabling you to set the WHERE clause. You can optionally pass this information directly into the update function as a string:

```
$this->db->update('mytable', $data, "id = 4");
```

Or as an array:

```
$this->db->update('mytable', $data, array('id' => $id));
```

You may also use the $this->db->set() function described above when performing updates.

删除数据

$this->db->delete();

Generates a delete SQL string and runs the query.

```
$this->db->delete('mytable', array('id' => $id));
```

```
// Produces:
// DELETE FROM mytable
// WHERE id = $id
```

The first parameter is the table name, the second is the where clause. You can also use the where() or orwhere() functions instead of passing the data to the second parameter of the function:

```
$this->db->where('id', $id);
$this->db->delete('mytable');
// Produces:
// DELETE FROM mytable
// WHERE id = $id
```

Note: All values are escaped automatically producing safer queries.

Method Chaining

Method chaining allows you to simplify your syntax by connecting multiple functions. Consider this example:

```
$this->db->select('title')->from('mytable')->where('id', $id)->limit(10, 20);
$query = $this->db->get();
```

Note: Method chaining only works with PHP 5.

Transactions

CodeIgniter's database abstraction allows you to use transactions with databases that support transaction-safe table types. In MySQL, you'll need to be running InnoDB or BDB table types rather then the more common MyISAM. Most other database platforms support transactions natively.

If you are not familiar with transactions we recommend you find a good online resource to learn about them for your particular database. The information below assumes you have a basic understanding of transactions.

CodeIgniter's Approach to Transactions

CodeIgniter utilizes an approach to transactions that is very similar to the process used by the popular database class ADODB. We've chosen that approach because it greatly simplifies the process of running transactions. In most cases all that is required are two lines of code.

Traditionally, transactions have required a fair amount of work to implement since they demand that you to keep track of your queries and determine whether to commit or rollback based on the success or failure of your queries. This is particularly cumbersome with nested queries. In contrast, we've implemented a smart transaction system that does all this for you automatically (you can also manage your transactions manually if you choose to, but there's really no benefit).

Running Transactions

To run your queries using transactions you will use the $this->db->trans_start() and $this->db->trans_complete() functions as follows:

```
$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();
```

You can run as many queries as you want between the start/complete functions and they will all be committed or rolled back based on success or failure of any given query.

Managing Errors

If you have error reporting enabled in your config/database.php file you'll see a standard error message if the commit was unsuccessful. If debugging is turned off, you can manage your own errors like this:

```
$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)
{
    // generate an error... or use the log_message() function to log your error
}
```

Enabling Transactions

Transactions are enabled automatically the moment you use $this->db->trans_start(). If you would like to disable transactions you can do so using $this->db->trans_off():

```
$this->db->trans_off()

$this->db->trans_start();
$this->db->query('AN SQL QUERY...');
$this->db->trans_complete();
```

When transactions are disabled, your queries will be auto-commited, just as they are when running queries without transactions.

Test Mode

You can optionally put the transaction system into "test mode", which will cause your queries to be rolled back -- even if the queries produce a valid result. To use test mode simply set the first parameter in the $this->db->trans_start() function to TRUE:

```
$this->db->trans_start(TRUE); // Query will be rolled back

$this->db->query('AN SQL QUERY...');
$this->db->trans_complete();
```

Running Transactions Manually

If you would like to run transactions manually you can do so as follows:

```
$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();
}
```

Note: Make sure to use $this->db->trans_begin() when running manual transactions, **NOT** $this->db->trans_start().

Table Data

These functions let you fetch table information.

$this->db->list_tables();

Returns an array containing the names of all the tables in the database you are currently connected to. Example:

```
$tables = $this->db->list_tables()

foreach ($tables as $table)
{
   echo $table;
}
```

$this->db->table_exists();

Sometimes it's helpful to know whether a particular table exists before running an operation on it. Returns a boolean TRUE/FALSE. Usage example:

```
if ($this->db->table_exists('table_name'))

{
   // some code...
}
```

Note: Replace *table_name* with the name of the table you are looking for.

Field Data

$this->db->list_fields()

Returns an array containing the field names. This query can be called two ways:

1. You can supply the table name and call it from the $this->db-> object:

```
$fields = $this->db->list_fields('table_name')

foreach ($fields as $field)
{
     echo $field;
}
```

2. You can gather the field names associated with any query you run by calling the function from your query result object:

```
$query = $this->db->query('SELECT * FROM some_table')

foreach ($query->list_fields() as $field)
{
     echo $field;
}
```

$this->db->field_exists()

Sometimes it's helpful to know whether a particular field exists before performing an action. Returns a boolean TRUE/FALSE. Usage example:

```
if ($this->db->field_exists('field_name', 'table_name'))

{
   // some code...
}
```

Note: Replace *field_name* with the name of the column you are looking for, and replace *table_name* with the name of the table you are looking for.

$this->db->field_data()

Returns an array of objects containing field information.

Sometimes it's helpful to gather the field names or other metadata, like the column type, max length, etc.

Note: Not all databases provide meta-data.

Usage example:

```
$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;
}
```

If you have run a query already you can use the result object instead of supplying the table name:

```
$query = $this->db->query("YOUR QUERY")

$fields = $query->field_data()
```

The following data is available from this function if supported by your database:

- name - column name
- max_length - maximum length of the column
- primary_key - 1 if the column is a primary key
- type - the type of the column

Custom Function Calls

$this->db->call_function();

This function enables you to call PHP database functions that are not natively included in CodeIgniter, in a platform independent manner. For example, lets say you want to call the mysql_get_client_info() function, which is **not** natively supported by CodeIgniter. You could do so like this:

```
$this->db->call_function('get_client_info');
```

You must supply the name of the function, **without** the mysql_ prefix, in the first parameter. The prefix is added automatically based on which database driver is currently being used. This permits you to run the same function on different database platforms. Obviously not all function calls are identical between platforms, so there are limits to how useful this function can be in terms of portability.

Any parameters needed by the function you are calling will be added to the second parameter.

```
$this->db->call_function('some_function', $param1, $param2, etc..);
```

Often, you will either need to supply a database connection ID or a database result ID. The connection ID can be accessed using:

```
$this->db->conn_id;
```

The result ID can be accessed from within your result object, like this:

```
$query = $this->db->query("SOME QUERY");
```

$query->result_id;

Database Caching Class

The Database Caching Class permits you to cache your queries as text files for reduced database load.

Important: This class is initialized automatically by the database driver when caching is enabled. Do NOT load this class manually.

Also note: Not all query result functions are available when you use caching. Please read this page carefully.

Enabling Caching

Caching is enabled in three steps:

- Create a writable directory on your server where the cache files can be stored.
- Set the path to your cache folder in your application/config/database.php file.
- Enable the caching feature, either globally by setting the preference in your application/config/database.php file, or manually as described below.

Once enabled, caching will happen automatically whenever a page is loaded that contains database queries.

How Does Caching Work?

CodeIgniter's query caching system happens dynamically when your pages are viewed. When caching is enabled, the first time a web page is loaded, the query result object will be serialized and stored in a text file on your server. The next time the page is loaded the cache file will be used instead of accessing your database. Your database usage can effectively be reduced to zero for any pages that have been cached.

Only read-type (SELECT) queries can be cached, since these are the only type of queries that produce a result. Write-type (INSERT, UPDATE, etc.) queries, since they don't generate a result, will not be cached by the system.

Cache files DO NOT expire. Any queries that have been cached will remain cached until you delete them. The caching system permits you clear caches associated with individual pages, or you can delete the entire collection of cache files. Typically you'll to use the housekeeping functions described below to delete cache files after certain events take place, like when you've added new information to your database.

Will Caching Improve Your Site's Performance?

Getting a performance gain as a result of caching depends on many factors. If you have a highly optimized database under very little load, you probably won't see a performance boost. If your database is under heavy use you probably will see an improved response, assuming your file-system is not overly taxed. Remember that caching simply changes how your information is retrieved, shifting it from being a database operation to a file-system one.

In some clustered server environments, for example, caching may be detrimental since file-system operations are so intense. On single servers in shared environments, caching will probably be beneficial. Unfortunately there is no single answer to the question of whether you should cache your database. It really depends on your situation.

How are Cache Files Stored?

CodeIgniter places the result of EACH query into its own cache file. Sets of cache files are further organized into sub-folders corresponding to your controller functions. To be precise, the sub-folders are named identically to the first two segments of your URI (the controller class name and function name).

For example, let's say you have a controller called blog with a function called comments that contains three queries. The caching system will create a cache folder called blog+comments, into which it will write three cache files.

If you use dynamic queries that change based on information in your URI (when using pagination, for example), each instance of the query will produce its own cache file. It's possible, therefore, to end up with many times more cache files than you have queries.

Managing your Cache Files

Since cache files do not expire, you'll need to build deletion routines into your application. For example, let's say you have a blog that allows user commenting. Whenever a new comment is submitted you'll want to delete the cache files associated with the controller function that serves up your comments. You'll find two delete functions described below that help you clear data.

Not All Database Functions Work with Caching

Lastly, we need to point out that the result object that is cached is a simplified version of the full result object. For that reason, some of the query result functions are not available for use.

The following functions ARE NOT available when using a cached result object:

num_fields()

field_names()
field_data()
free_result()

Also, the two database resources (result_id and conn_id) are not available when caching, since result resources only pertain to run-time operations.

Function Reference

$this->db->cache_on() / $this->db->cache_off()

Manually enables/disables caching. This can be useful if you want to keep certain queries from being cached. Example:

```
// Turn caching on
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM mytable");
// Turn caching off for this one query
$this->db->cache_off();
$query = $this->db->query("SELECT * FROM members WHERE member_id = '$current_user'");
// Turn caching back on
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM another_table");
```

$this->db->cache_delete()

Deletes the cache files associated with a particular page. This is useful if you need to clear caching after you update your database.

The caching system saves your cache files to folders that correspond to the URI of the page you are viewing. For example, if you are viewing a page at www.your-site.com/index.php/blog/comments, the caching system will put all cache files associated with it in a folder called blog+comments. To delete those particular cache files you will use:

```
$this->db->cache_delete('blog', 'comments');
```

If you do not use any parameters the current URI will be used when determining what should be cleared.

$this->db->cache_delete_all()

Clears all existing cache files. Example:

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

Database Utility Class

The Database Utility Class contains functions that help you manage your database.

Table of Contents

Initializing the Utility Class

Important: In order to initialize the Utility class, your database driver must already be running, since the utilities class relies on it.

Load the Utility Class as follows:

```
$this->load->dbutil()
```

Once initialized you will access the functions using the $this->dbutil object:

```
$this->dbutil->some_function()
```

$this->dbutil->create_database('db_name')

Permits you to create the database specified in the first parameter. Returns TRUE/FALSE based on success or failure:

```
if ($this->dbutil->create_database('my_db'))
{
    echo 'Database created!';
}
```

$this->dbutil->drop_database('db_name')

Permits you to drop the database specified in the first parameter. Returns TRUE/FALSE based on success or failure:

```
if ($this->dbutil->drop_database('my_db'))
{
    echo 'Database deleted!';
}
```

$this->dbutil->list_databases()

Returns an array of database names:

```
$dbs = $this->dbutil->list_databases();

foreach($dbs as $db)
```

```
{
    echo $db;
}
```

$this->dbutil->optimize_table('table_name');

Note: This features is only available for MySQL/MySQLi databases.

Permits you to optimize a table using the table name specified in the first parameter. Returns TRUE/FALSE based on success or failure:

```
if ($this->dbutil->optimize_table('table_name'))
{
    echo 'Success!';
}
```

Note: Not all database platforms support table optimization.

$this->dbutil->repair_table('table_name');

Note: This features is only available for MySQL/MySQLi databases.

Permits you to repair a table using the table name specified in the first parameter. Returns TRUE/FALSE based on success or failure:

```
if ($this->dbutil->repair_table('table_name'))
{
    echo 'Success!';
}
```

Note: Not all database platforms support table repairs.

$this->dbutil->optimize_database();

Note: This features is only available for MySQL/MySQLi databases.

Permits you to optimize the database your DB class is currently connected to. Returns an array containing the DB status messages or FALSE on failure.

```
$result = $this->dbutil->optimize_database();
if ($result !== FALSE)
{
    print_r($result);
}
```

Note: Not all database platforms support table optimization.

$this->dbutil->csv_from_result($db_result)

Permits you to generate a CSV file from a query result. The first parameter of the function must contain the result object from your query. Example:

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

$query = $this->db->query("SELECT * FROM mytable");
echo $this->dbutil->csv_from_result($query);
```

The second and third parameters allows you to set the delimiter and newline character. By default tabs are used as the delimiter and "\n" is used as a new line. Example:

```
$delimiter = ",";

$newline = "\r\n";
echo $this->dbutil->csv_from_result($query, $delimiter, $newline);
```

Important: This function will NOT write the CSV file for you. It simply creates the CSV layout. If you need to write the file use the File Helper.

$this->dbutil->xml_from_result($db_result)

Permits you to generate an XML file from a query result. The first parameter expects a query result object, the second may contain an optional array of config parameters. Example:

```
$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);
```

Important: This function will NOT write the XML file for you. It simply creates the XML layout. If you need to write the file use the File Helper.

$this->dbutil->backup()

Permits you to backup your full database or individual tables. The backup data can be compressed in either Zip or Gzip format.

Note: This features is only available for MySQL/MySQLi databases.

Note: Due to the limited execution time and memory available to PHP, backing up very large databases may not be possible. If your database is very large you might need to backup directly from your SQL server via the command line, or have your server admin do it for you if you do not have root privileges.

Usage Example

```
// Load the DB utility class

$this->load->dbutil();
// Backup your entire database and assign it to a variable
$backup =& $this->dbutil->backup();
// Load the file helper and write the file to your server
$this->load->helper('file');
write_file('/path/to/mybackup.gz', $backup);
// Load the download helper and send the file to your desktop
$this->load->helper('download');
force_download('mybackup.gz', $backup);
```

Setting Backup Preferences

Backup preferences are set by submitting an array of values to the first parameter of the backup function. Example:

```
$prefs = array(
                'tables'      => array('table1', 'table2'),    // Array of tables to backup.
                'ignore'      => array(),                  // List of tables to omit from the backup
                'format'      => 'txt',                    // gzip, zip, txt
                'filename'    => 'mybackup.sql',           // File name - NEEDED ONLY WITH ZIP
FILES
                'add_drop'    => TRUE,                        // Whether to add DROP TABLE
statements to backup file
                'add_insert'  => TRUE,                        // Whether to add INSERT data to
backup file
                'newline'     => "\n"                      // Newline character used in backup file
              );
$this->dbutil->backup($prefs);
```

Description of Backup Preferences

Preference	Default Value	Options	Description
tables	empty array	None	An array of tables you want backed up. If left blank all tables will be exported.
ignore	empty array	None	An array of tables you want the backup routine to ignore.
format	gzip	gzip, zip, txt	The file format of the export file.
filename	the current date/time	None	The name of the backed-up file. The name is needed only if you are using zip compression.

add_drop	TRUE	TRUE/FALSE	Whether to include DROP TABLE statements in your SQL export file.
add_insert	TRUE	TRUE/FALSE	Whether to include INSERT statements in your SQL export file.
newline	"\n"	"\n","\r", "\r\n"	Type of newline to use in your SQL export file.

Email Class

CodeIgniter's robust Email Class supports the following features:

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

Sending Email

Sending email is not only simple, but you can configure it on the fly or set your preferences in a config file.

Here is a basic example demonstrating how you might send email. Note: This example assumes you are sending the email from one of your controllers.

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

$this->email->from('your@your-site.com', 'Your Name');
$this->email->to('someone@some-site.com');
$this->email->cc('another@another-site.com');
$this->email->bcc('them@their-site.com');
$this->email->subject('Email Test');
$this->email->message('Testing the email class.');
$this->email->send();
echo $this->email->print_debugger();
```

Setting Email Preferences

There are 17 different preferences available to tailor how your email messages are sent. You can either set them manually as described here, or automatically via preferences stored in your config file, described below:

Preferences are set by passing an array of preference values to the email initialize function. Here is an example of how you might set some preferences:

```
$config['protocol'] = 'sendmail';
$config['mailpath'] = '/usr/sbin/sendmail';
$config['charset'] = 'iso-8859-1';
$config['wordwrap'] = TRUE;
$this->email->initialize($config);
```

Note: Most of the preferences have default values that will be used if you do not set them.

Setting Email Preferences in a Config File

If you prefer not to set preferences using the above method, you can instead put them into a config file. Simply create a new file called the Email.php, add the $config array in that file. Then save the file at config/Email.php and it will be used automatically. You will NOT need to use the $this->email->initialize() function if you save your preferences in a config file.

Email Preferences

The following is a list of all the preferences that can be set when sending email.

Preference	Default Value	Options	Description
useragent	CodeIgniter	None	The "user agent".
protocol	mail	mail, sendmail, or smtp	The mail sending protocol.
mailpath	/usr/sbin/sendmail	None	The server path to Sendmail.
smtp_host	No Default	None	SMTP Server Address.
smtp_user	No Default	None	SMTP Username.
smtp_pass	No Default	None	SMTP Password.
smtp_port	25	None	SMTP Port.
smtp_timeout	5	None	SMTP Timeout (in seconds).
wordwrap	TRUE	TRUE or FALSE (boolean)	Enable word-wrap.
wrapchars	76		Character count to wrap at.
mailtype	text	text or html	Type of mail. If you send HTML email you must send it as a complete web page. Make sure you don't have any relative links or relative image paths otherwise they will not work.
charset	utf-8		Character set (utf-8, iso-8859-1, etc.).

validate	FALSE	TRUE or FALSE (boolean)	Whether to validate the email address.
priority	3	1, 2, 3, 4, 5	Email Priority. 1 = highest. 5 = lowest. 3 = normal.
newline	\n	"\r\n" or "\n"	Newline character. (Use "\r\n" to comply with RFC 822).
bcc_batch_mode	FALSE	TRUE or FALSE (boolean)	Enable BCC Batch Mode.
bcc_batch_size	200	None	Number of emails in each BCC batch.

Email Function Reference

$this->email->from()

Sets the email address and name of the person sending the email:

$this->email->from('you@your-site.com', 'Your Name');

$this->email->reply_to()

Sets the reply-to address. If the information is not provided the information in the "from" function is used. Example:

$this->email->reply_to('you@your-site.com', 'Your Name');

$this->email->to()

Sets the email address(s) of the recipient(s). Can be a single email, a comma-delimited list or an array:

$this->email->to('someone@some-site.com'); $this->email->to('one@some-site.com', 'two@some-site.com', 'three@some-site.com'); $list = array('one@some-site.com', 'two@some-site.com', 'three@some-site.com');

$this->email->to($list);

$this->email->cc()

Sets the CC email address(s). Just like the "to", can be a single email, a comma-delimited list or an array.

$this->email->bcc()

Sets the BCC email address(s). Just like the "to", can be a single email, a comma-delimited list or an array.

$this->email->subject()

Sets the email subject:

```
$this->email->subject('This is my subject');
```

$this->email->message()

Sets the email message body:

```
$this->email->message('This is my message');
```

$this->email->set_alt_message()

Sets the alternative email message body:

```
$this->email->set_alt_message('This is the alternative message');
```

This is an optional message string which can be used if you send HTML formatted email. It lets you specify an alternative message with no HTML formatting which is added to the header string for people who do not accept HTML email. If you do not set your own message CodeIgniter will extract the message from your HTML email and strip the tags.

$this->email->clear()

Initializes all the email variables to an empty state. This function is intended for use if you run the email sending function in a loop, permitting the data to be reset between cycles.

```
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();
}
```

If you set the parameter to TRUE any attachments will be cleared as well:

```
$this->email->clear(TRUE);
```

$this->email->send()

The Email sending function. Returns boolean TRUE or FALSE based on success or failure, enabling it to be used conditionally:

```
if ( ! $this->email->send())
{
	// Generate error
}
```

$this->email->attach()

Enables you to send an attachment. Put the file path/name in the first parameter. Note: Use a file path, not a URL. For multiple attachments use the function multiple times. For example:

```
$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()

Returns a string containing any server messages, the email headers, and the email messsage. Useful for debugging.

Overriding Word Wrapping

If you have word wrapping enabled (recommended to comply with RFC 822) and you have a very long link in your email it can get wrapped too, causing it to become un-clickable by the person receiving it. CodeIgniter lets you manually override word wrapping within part of your message like this:

```
The text of your email that
gets wrapped normally.
{unwrap}http://www.some-site.com/a_long_link_that_should_not_be_wrapped.html{/unwrap}
More text that will be
wrapped normally.
```

Place the item you do not want word-wrapped between: {unwrap} {/unwrap}

Encryption Class

The Encryption Class provides two-way data encryption. It uses a scheme that pre-compiles the message using a randomly hashed bitwise XOR encoding scheme, which is then encrypted using the Mcrypt library. If Mcrypt is not available on your server the encoded message will still provide a reasonable degree of security for encrypted sessions or other such "light" purposes. If Mcrypt is available, you'll effectively end up with a double-encrypted message string, which should provide a very high degree of security.

这个加密类提供了两种数据加密的方法。首先它使用一些随机的无顺序的逐位异或xor直接交换

数值的编码方案预先编译一些信息，然后再将它们放在隐蔽的类似Mcrypt的地方。如果这个Mcrypt 不是可利用的，会给您的服务器编码消息轻巧的加密，将会提供合理安全程度。如果这个Mcrypt 是可以使用的，最终你可以获得一个被双重加密的字符串，这样加密的话，会使你的信息非常安全

Setting your Key

A *key* is a piece of information that controls the cryptographic process and permits an encrypted string to be decoded. In fact, the key you chose will provide the **only** means to decode data that was encrypted with that key, so not only must you chose the key carefully, you must never change it if you intend use it for persistent data.

钥匙实际上是一些会控制密码加密过程并且允许被加密的字串被解码的信息片段。实际上，你选择的钥匙会提供一个唯一的方法来解密一些被加密的数据，所以你需要非常谨慎的设置你的钥匙，如果你想给一些固定的数据加密的话，你最好不要更改这个钥匙。

It goes without saying that you should guard your key carefully. Should someone gain access to your key, the data will be easily decoded. If your server is not totally under your control it's impossible to ensure key security so you may want to think carefully before using it for anything that requires high security, like storing credit card numbers.

很自然，你需要非常小心的保守你的钥匙。如果某人对您的钥匙能够存取，那么数据将会很容易地被解码。如果您的服务器不完全在的您的控制之下而想保证数据安全是不可能的，因此您可以在使用它之前仔细地想一下要求高安全存放信用卡数字对象的方法。

To take maximum advantage of the encryption algorithm, your key should be 32 characters in length (128 bits). The key should be as random a string as you can concoct, with numbers and uppercase and lowercase letters. Your key should **not** be a simple text string. In order to be cryptographically secure it needs to be as random as possible.

为了发挥加密算法的最大优势，你的解密钥匙需要被设置为32个字符长度为128比特。你可以设置一个编造的随机字符串作为你的钥匙，最好包括数字、大写字母、小写字母。你的钥匙不能设置为一个简单的文本字符串。为了被安全可靠的加密，它也有一个随机的可能性。

Your key can be either stored in your application/config/config.php, or you can design your own storage mechanism and pass the key dynamically when encoding/decoding.

你的钥匙可以放在**application/config/config.php**文件中，你也可以自己设置一个存储机制用于数据的加密和解密。

To save your key to your application/config/config.php, open the file and set:

为了在**application/config/config.php文件中**保存你的钥匙，打开文件设置一下：

```
$config['encryption_key'] = "YOUR KEY";
```

Message Length

It's important for you to know that the encoded messages the encryption function generates will be approximately 2.6 times longer than the original message. For example, if you encrypt the string "my super secret data", which is 21 characters in length, you'll end up with an encoded string that is roughly 55 characters (we say "roughly" because the encoded string length increments in 64 bit clusters, so it's not exactly linear). Keep this information in mind when selecting your data storage mechanism. Cookies, for example, can only hold 4K of information.

知道加密信息的长度会是原来函数长度的2.6倍是很重要的。如果你加密这个字符串“my super secret data”,它的长度是21个字符，所以你加密后的字符串的长度大概是55个字符（我们说它是粗糙的，因为编码的字符串长度增量64比特并非是线性增长的），当你选择你的数据存储机制的时候一定要记住这一点。例如，Cookies可以占用4k的数据空间。

Initializing the Class

Like most other classes in CodeIgniter, the Encryption class is initialized in your controller using the $this->load->library function:

在Codeigniter中，像大多数其他的类一样，加密类也需要在你的控制器函数中加载

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

Once loaded, the Encrypt library object will be available using: $this->encrypt

一旦被加载，这个类库就可以这样使用：$this->encrypt

$this->encrypt->encode()

Performs the data encryption and returns it as a string. Example:

执行数据加密并返回一个字符串。例如：

$msg = 'My secret message';

$encrypted_string = $this->encrypt->encode($msg);

You can optionally pass your encryption key via the second parameter if you don't want to use the one in your config file:

如果你不想在你的配置文件中使用一个钥匙，你可以通过第二个参数随意设置你的钥匙。

$msg = 'My secret message';

$key = 'super-secret-key';
$encrypted_string = $this->encrypt->encode($msg, $key);

$this->encrypt->decode()

Decrypts an encoded string. Example:

$encrypted_string = 'APANtByIGI1BpVXZTJgcsAG8GZl8pdwwa84';

$plaintext_string = $this->encrypt->decode($encrypted_string);

$this->encrypt->set_cipher();

Permits you to set an Mcrypt cipher. By default it uses MCRYPT_RIJNDAEL_256. Example:

允许你设置一个Mcrypt密码。默认使用**MCRYPT_MODE_ECB**

$this->encrypt->set_cipher(MCRYPT_BLOWFISH);

Please visit php.net for a list of available ciphers.

请访问php.net看一下可用的密码。

If you'd like to manually test whether your server supports Mcrypt you can use:

如果你想手动测试一下你的服务器是否支持Mcrypt,你可以这样使用：

echo (! function_exists('mcrypt_encrypt')) ? 'Nope' : 'Yup';

$this->encrypt->set_mode();

Permits you to set an Mcrypt mode. By default it uses MCRYPT_MODE_ECB. Example:

允许你设置一个Mcrypt模型。默认使用MCRYPT_MODE_ECB

$this->encrypt->set_mode(MCRYPT_MODE_CFB);

Please visit php.net for a list of available modes.

$this->encrypt->sha1();

SHA1 encoding function. Provide a string and it will return a 160 bit one way hash. Note: SHA1, just like MD5 is non-decodable. Example:

SHA1编码函数。它提供一个字符串并返回一个160字节的杂乱信息。说明：SHA1，就像MD5一样不用解密的

$hash = $this->encrypt->sha1('Some string');

Many PHP installations have SHA1 support by default so if all you need is to encode a hash it's simpler to use the native function:

许多PHP安装程序默认都支持SHA1，所以你可以很简单的使用它的原始函数进行加密。

$hash = sha1('Some string');

If your server does not support SHA1 you can use the provided function.

如果你的服务器不支持SHA1，你可以使用别人提供的函数。

文件上传类

CodeIgniter的文件上传类允许文件被上传。您可以设置指定上传某类型的文件及指定大小的文件。

处理过程

上传文件普遍的过程：

一个上传文件用的表单，允许用户选择一个文件并上传它。

当这个表单被提交，该文件被上传到指定的目录。
同时，该文件将被验证是否符合您设定的要求。
一旦文件上传成功，还要返回一个上传成功的确认窗口。

To demonstrate this process here is brief tutorial. Afterward you'll find reference information.

创建上传表单

运用文本编辑器创建一个名为upload_form.php的文件，复制以下代码并保存在applications/views/目录里：

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

你会看到这里运用到了一个表单辅助函数来创建表单的开始标签，文件上传需要一个 multipart form，因为这个表单辅助函数为你创建了一个合适的语句。你还会看到我们运用了一个$error变量，当用户提交该表单出现错误时会显示相关的出错信息。

上传成功的页面

运用文本编辑器创建一个名为upload_success.php的文件。复制以下代码保存到applications/views/目录里：

```
<html> <head> <title>Upload Form</title> </head> <body> <h3>Your file was successfully
uploaded!</h3> <ul> <?php foreach($upload_data as $item => $value):?> <li><?=$item;?>:
<?=$value;?></li> <?php endforeach; ?> </ul> <p><?=anchor('upload', 'Upload Another File!');
?></p> </body> </html>
```

控制器

运用文本编辑器，创建一个名为upload.php的控制器.复制以下代码并保存到applications/controllers/目录里：

```
<?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); } } } ?>
```

上传文件目录

你还需要一个目标文件夹来存储上传的图片。在根目录上创建一个名为uploads的文件并设置该文件的属性为 777。（即可读写）

提交表单

要提交你的表单，输入类似如下的URL：

www.your-site.com/index.php/upload/

你将看到一个上传表单，任选一张(jpg, gif,或者png)图片进行提交. 如果你在控制器里设置的路径是正确的，它将开始工作。

偏好向导

初始化文件上传类

与CodeIgniter的其它一些类相似，文件上传类用$this->load->library函数在控制器里进行初始化:

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

一旦文件上传类被加载，对象将通过如下方法来引用：$this->upload

偏好设置

与其它库类似，你将根据你的偏好设置来控制要被上传的文件，在控制器里，你建立了如下的偏好设置：

```
$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);
// Alternately you can set preferences by calling the initialize function. Useful if you auto-load the class:
$this->upload->initialize($config);
```

以上偏好设置将被完全执行。以下是所有偏好设置参数的描述。

偏好设置参数

以下偏好设置参数是可用的。当你没有特别指定偏好设置参数时，默认值如下：

Preference	Default Value	Options	Description

upload_path	None	None	The path to the folder where the upload should be placed. The folder must be writable and the path can be absolute or relative.
allowed_types	None	None	The mime types corresponding to the types of files you allow to be uploaded. Usually the file extension can be used as the mime type. Separate multiple types with a pipe.
overwrite	FALSE	TRUE/FALSE (boolean)	If set to true, if a file with the same name as the one you are uploading exists, it will be overwritten. If set to false, a number will be appended to the filename if another with the same name exists.
max_size	0	None	The maximum size (in kilobytes) that the file can be. Set to zero for no limit. Note: Most PHP installations have their own limit, as specified in the php.ini file. Usually 2 MB (or 2048 KB) by default.
max_width	0	None	The maximum width (in pixels) that the file can be. Set to zero for no limit.
max_height	0	None	The maximum height (in pixels) that the file can be. Set to zero for no limit.
encrypt_name	FALSE	TRUE/FALSE (boolean)	If set to TRUE the file name will be converted to a random encrypted string. This can be useful if you would like the file saved with a name that can not be discerned by the person uploading it.
remove_spaces	TRUE	TRUE/FALSE (boolean)	If set to TRUE, any spaces in the file name will be converted to underscores. This is recommended.

在配置文件里设置偏好设置参数

如果你不愿意应用如上方法进行偏好设置，你可能用一个配置文件来取代它。简单创建一个名为upload.php的文件,添加 $config数组到该文件里，然后保存文件到：config/upload.php，它将被自动加载。当你把配置参数保存到该文件里，你不需要用$this->upload->initialize函数进行手动加载。

运用到的函数

以下函数被运用

$this->upload->do_upload()

根据你的偏好配置参数执行操作。注意：默认情况下上传的文件来自于提交表单里名为userfile的文件域,并且该表单必须是 "multipart"类型：

<form method="post" action="some_action" enctype="multipart/form-data" />

如果你想在执行do_upload函数之前自定义自己的文件域名称，可通过以下方法实现：

```
$field_name = "some_field_name";
$this->upload->do_upload($field_name)
```

$this->upload->display_errors()

Retrieves any error messages if the do_upload() function returned false. The function does not echo automatically, it returns the data so you can assign it however you need.

Formatting Errors

By default the above function wraps any errors within <p> tags. You can set your own delimiters like this:

```
$this->upload->display_errors('<p>', '</p>');
```

$this->upload->data()

This is a helper function that returns an array containing all of the data related to the file you uploaded. Here is the array prototype:

```
Array
(
    [file_name]     => mypic.jpg
    [file_type]     => image/jpeg
    [file_path]     => /path/to/your/upload/
    [full_path]     => /path/to/your/upload/jpg.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"
)
```

Explanation

Here is an explanation of the above array items.

Item	Description

file_name	The name of the file that was uploaded including the file extension.
file_type	The file's Mime type
file_path	The absolute server path to the file
full_path	The absolute server path including the file name
raw_name	The file name without the extension
orig_name	The original file name. This is only useful if you use the encrypted name option.
file_ext	The file extension with period
file_size	The file size in kilobytes
is_image	Whether the file is an image or not. 1 = image. 0 = not.
image_width	Image width.
image_heigth	Image height
image_type	Image type. Typically the file extension without the period.
image_size_str	A string containing the width and height. Useful to put into an image tag.

FTP 类

CodeIgniter's FTP Class permits files to be transfered to a remote server. Remote files can also be moved, renamed, and deleted. The FTP class also includes a "mirroring" function that permits an entire local directory to be recreated remotely via FTP.

Note: SFTP and SSL FTP protocols are not supported, only standard FTP.

初始化类

Like most other classes in CodeIgniter, the FTP class is initialized in your controller using the $this->load->library function:

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

Once loaded, the FTP object will be available using: $this->ftp

使用例子

In this example a connection is opened to the FTP server, and a local file is read and uploaded in ASCII mode. The file permissions are set to 755. Note: Setting permissions requires PHP 5.

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

$config['hostname'] = 'ftp.your-site.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['debug'] = TRUE;
$this->ftp->connect($config);
$this->ftp->upload('/local/path/to/myfile.html', '/public_html/myfile.html', 'ascii', 0775);
$this->ftp->close();
```

In this example a list of files is retrieved from the server.

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

$config['hostname'] = 'ftp.your-site.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['debug'] = TRUE;
$this->ftp->connect($config);
$list = $this->ftp->list_files('/public_html/');
print_r($list);
$this->ftp->close();
```

In this example a local directory is mirrored on the server.

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

$config['hostname'] = 'ftp.your-site.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['debug'] = TRUE;
$this->ftp->connect($config);
$this->ftp->mirror('/path/to/myfolder/', '/public_html/myfolder/');
$this->ftp->close();
```

函数参考

$this->ftp->connect()

Connects and logs into to the FTP server. Connection preferences are set by passing an array to the function, or you can store them in a config file.

Here is an example showing how you set preferences manually:

```
$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);
```

Setting FTP Preferences in a Config File

If you prefer you can store your FTP preferences in a config file. Simply create a new file called the ftp.php, add the $config array in that file. Then save the file at config/ftp.php and it will be used automatically.

Available connection options:

hostname - the FTP hostname. Usually something like: ftp.some-site.com

username - the FTP username.
password - the FTP password.
port - The port number. Set to 21 by default.
debug - TRUE/FALSE (boolean). Whether to enable debugging to display error messages.
passive - TRUE/FALSE (boolean). Whether to use passive mode. Passive is set automatically by default.

$this->ftp->upload()

Uploads a file to your server. You must supply the local path and the remote path, and you can optionally set the mode and permissions. Example:

$this->ftp->upload('/local/path/to/myfile.html', '/public_html/myfile.html', 'ascii', 0775);

Mode options are: ascii, binary, and auto (the default). If auto is used it will base the mode on the file extension of the source file.

Permissions are available if you are running PHP 5 and can be passed as an octal value in the fourth parameter.

$this->ftp->rename()

Permits you to rename a file. Supply the source file name/path and the new file name/path.

// Renames green.html to blue.html

$this->ftp->rename('/public_html/foo/green.html', '/public_html/foo/blue.html');

$this->ftp->move()

Lets you move a file. Supply the source and destination paths:

// Moves blog.html from "joe" to "fred"

$this->ftp->move('/public_html/joe/blog.html', '/public_html/fred/blog.html');

Note: if the destination file name is different the file will be renamed.

$this->ftp->delete_file()

Lets you delete a file. Supply the source path with the file name.

$this->ftp->delete_file('/public_html/joe/blog.html');

$this->ftp->delete_dir()

Lets you delete a directory and everything it contains. Supply the source path to the directory with a trailing slash.

Important Be VERY careful with this function. It will recursively delete **everything** within the supplied path, including sub-folders and all files. Make absolutely sure your path is correct. Try using the list_files() function first to verify that your path is correct.

$this->ftp->delete_dir('/public_html/path/to/folder/');

$this->ftp->list_files()

Permits you to retrieve a list of files on your server returned as an array. You must supply the path to the desired directory.

$list = $this->ftp->list_files('/public_html/');

print_r($list);

$this->ftp->mirror()

Recursively reads a local folder and everything it contains (including sub-folders) and creates a mirror via FTP based on it. Whatever the directory structure of the original file path will be recreated on the server. You must supply a source path and a destination path:

$this->ftp->mirror('/path/to/myfolder/', '/public_html/myfolder/');

$this->ftp->mkdir()

Lets you create a directory on your server. Supply the path ending in the folder name you wish to create, with a trailing slash. Permissions can be set by passed an octal value in the second parameter (if you are running PHP 5).

// Creates a folder named "bar"

$this->ftp->mkdir('/public_html/foo/bar/', 0777);

$this->ftp->chmod()

Permits you to set file permissions. Supply the path to the file or folder you wish to alter permissions on:

// Chmod "bar" to 777

$this->ftp->chmod('/public_html/foo/bar/', 0777);

$this->ftp->close();

Closes the connection to your server. It's recommended that you use this when you are finished uploading.

HTML Table Class

The Table Class provides functions that enable you to auto-generate HTML tables from arrays or database result sets.

Initializing the Class

Like most other classes in CodeIgniter, the Table class is initialized in your controller using the $this->load->library function:

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

Once loaded, the Table library object will be available using: $this->table

Examples

Here is an example showing how you can create a table from a multi-dimensional array. Note that the first array index will become the table heading (or you can set your own headings using the set_heading() function described in the function reference below).

```
$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);
```

Here is an example of a table created from a database query result. The table class will automatically generate the headings based on the table names (or you can set your own headings using the set_heading() function described in the function reference below).

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

$query = $this->db->query("SELECT * FROM my_table");
echo $this->table->generate($query);
```

Here is an example showing how you might create a table using discreet parameters:

```
$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();
```

Here is the same example, except instead of individual parameters, arrays are used:

```
$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();
```

Changing the Look of Your Table

The Table Class permits you to set a table template with which you can specify the design of your layout. Here is the template prototype:

```
$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);
```

Note: You'll notice there are two sets of "row" blocks in the template. These permit you to create alternating row colors or design elements that alternate with each iteration of the row data.

You are NOT required to submit a complete template. If you only need to change parts of the layout you can simply submit those elements. In this example, only the table opening tag is being changed:

```
$tmpl = array ( 'table_open'  => '<table border="1" cellpadding="2" cellspacing="1" class="mytable">' );

$this->table->set_template($tmpl);
```

Function Reference

$this->table->generate()

Returns a string containing the generated table. Accepts an optional parameter which can be an array or a database result object.

$this->table->set_caption()

Permits you to add a caption to the table.

```
$this->table->set_caption('Colors');
```

$this->table->set_heading()

Permits you to set the table heading. You can submit an array or discreet params:

```
$this->table->set_heading('Name', 'Color', 'Size'); $this->table->set_heading(array('Name', 'Color', 'Size'));
```

$this->table->add_row()

Permits you to add a row to your table. You can submit an array or discreet params:

```
$this->table->add_row('Blue', 'Red', 'Green'); $this->table->add_row(array('Blue', 'Red', 'Green'));
```

$this->table->make_columns()

This function takes a one-dimensional array as input and creates a multi-dimensional array with a depth equal to the number of columns desired. This allows a single array with many elements to be displayed in a table that has a fixed column count. Consider this example:

```
$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)
// Generates a table with this prototype
<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()

Permits you to set your template. You can submit a full or partial template.

```
$tmpl = array ( 'table_open' => '<table border="1" cellpadding="2" cellspacing="1" class="mytable">' );
$this->table->set_template($tmpl);
```

$this->table->set_empty()

Let's you set a default value for use in any table cells that are empty. You might, for example, set a non-breaking space:

```
$this->table->set_empty("&nbsp;");
```

$this->table->clear()

Lets you clear the table heading and row data. If you need to show multiple tables with different data you should to call this function after each table has been generated to empty the previous table information. Example:

```
$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();
```

图像处理类

CodeIgniter 的图像处理类可以使你完成以下的操作：

调整图像大小

创建缩略图
图像裁剪
图像旋转
添加图像水印

可以很好的支持三个主流的图像库：GD/GD2, NetPBM, and ImageMagick。

注意: 添加水印操作仅仅在使用GD/GD2时可用。另外，即使支持其他的图像处理库，但是为了计算图像的属性，GD是必需的。然而，将使用你制定的库来进行图像处理操作。

初始化类

像CodeIgniter的大多数类一样，图像处理类在你的控制器里$this->load_library 方法来初始化：

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

在图像处理库被载入后就已经做好被使用的准备了。你将用来调用所有图像处理方法的图像处理库对象是：$this->image_libO

处理一个图像

不管你想进行何种图像处理操作（调整大小，图像裁剪，图像旋转，添加水印），通常过程都是一样的。你先设置一些你想进行的图像操作的参数，然后调用四个可用方法中的一个。例如，创建一个图像缩略图：

```
$config['image_library'] = 'GD';

$config['source_image'] = '/path/to/image/mypic.jpg';
$config['create_thumb'] = TRUE;
$config['maintain_ratio'] = TRUE;
$config['width'] = 75;
$config['height'] = 50;
$this->load->library('image_lib', $config);
$this->image_lib->resize();
```

以上代码告诉image_resize函数去查找位于source_image目录且名为*mypic.jpg*的图片，然后运用GD2图像库创建75 X 50像素的缩略图。 Since the maintain_ratio option is enabled, the thumb will be as close to the target width and height as possible while preserving the original aspect ratio. The thumbnail will be called *mypic_thumb.jpg*

Note: In order for the image class to be allowed to do any processing, the folder containing the image files must have file permissions of 777.

Processing Functions

There are four available processing functions:

```
$this->image_lib->resize()

$this->image_lib->crop()
$this->image_lib->rotate()
$this->image_lib->watermark()
```

These functions return boolean TRUE upon success and FALSE for failure. If they fail you can retrieve the error message using this function:

```
echo $this->image_lib->display_errors();
```

A good practice is use the processing function conditionally, showing an error upon failure, like this:

```
if ( ! $this->image_lib->resize())
{
    echo $this->image_lib->display_errors();
}
```

Note: You can optionally specify the HTML formatting to be applied to the errors, by submitting the opening/closing tags in the function, like this:

```
$this->image_lib->display_errors('<p>', '</p>');
```

Preferences

The 14 available preferences described below allow you to tailor the image processing to suit your needs.

Note that not all preferences are available for every function. For example, the x/y axis preferences are only available for image cropping. Likewise, the width and height preferences have no effect on cropping. The "availability" column indicates which functions support a given preference.

Availability Legend:

R - Image Resizing

C - Image Cropping

X - Image Rotation

W - Image Watermarking

Preference	Default Value	Options	Description	Availability
image_library	GD2	GD, GD2, ImageMagick, NetPBM	Sets the image library to be used.	R, C, X, W
library_path	None	None	Sets the server path to your ImageMagick or NetPBM library. If you use either of those libraries you must supply the path.	R, C, X
source_image	None	None	Sets the source image name/path. The path must be a relative or absolute server path, not a URL.	R, C, S, W
dynamic_output	FALSE	TRUE/FALSE (boolean)	Determines whether the new image file should be written to disk or generated dynamically. No	R, C, X, W

			te: If you choose the dynamic setting, only one image can be shown at a time, and it can't be positioned on the page. It simply outputs the raw image dynamically to your browser, along with image headers.	
quality	90%	1 - 100%	Sets the quality of the image. The higher the quality the larger the file size.	R, C, X, W
new_image	None	None	Sets the destination image name/path. You'll use this preference when creating an image copy. The path must be a relative or absolute server path, not a URL.	R
width	None	None	Sets the width you would like the image set to.	R, C
height	None	None	Sets the height you would like the image set to.	R, C
create_thumb	FALSE	TRUE/FALSE (boolean)	Tells the image processing function to create a thumb.	R
thumb_marker	_thumb	None	Specifies the thumbnail indicator. It will be inserted just before the file extension, so mypic.jpg would become mypic_thumb.jpg	R
maintain_ratio	TRUE	TRUE/FALSE (boolean)	Specifies whether to maintain the original aspect ratio when resizing or use hard values.	R
master_dim	auto	auto, width, height	Specifies what to use as the master axis when resizing or creating thumbs. For example, let's say you want to resize an image to 100 X 75 pixels. If the source image size does not allow perfect resizing to those dimensions, this setting determines which axis should be used as the hard value. "auto" sets the axis automatically based on w	R

			hether the image is taller then wider, or vice versa.	
rotation_angle	None	90, 180, 270, vrt, hor	Specifies the angle of rotation when rotating images. Note that PHP rotates counter-clockwise, so a 90 degree rotation to the right must be specified as 270.	X
x_axis	None	None	Sets the X coordinate in pixels for image cropping. For example, a setting of 30 will crop an image 30 pixels from the left.	C
y_axis	None	None	Sets the Y coordinate in pixels for image cropping. For example, a setting of 30 will crop an image 30 pixels from the top.	C

Setting preferences in a config file

If you prefer not to set preferences using the above method, you can instead put them into a config file. Simply create a new file called the image_lib.php, add the $config array in that file. Then save the file in: config/image_lib.php and it will be used automatically. You will NOT need to use the $this->image_lib->initialize function if you save your preferences in a config file.

$this->image_lib->resize()

The image resizing function lets you resize the original image, create a copy (with or without resizing), or create a thumbnail image.

For practical purposes there is no difference between creating a copy and creating a thumbnail except a thumb will have the thumbnail marker as part of the name (ie, mypic_thumb.jpg).

All preferences listed in the table above are available for this function except these three: rotation, x_axis, and y_axis.

Creating a Thumbnail

The resizing function will create a thumbnail file (and preserve the original) if you set this preference so TRUE:

$config['create_thumb'] = TRUE;

This single preference determines whether a thumbnail is created or not.

Creating a Copy

The resizing function will create a copy of the image file (and preserve the original) if you set a path and/or a new filename using this preference:

```
$config['new_image'] = '/path/to/new_image.jpg';
```

Notes regarding this preference:

- If only the new image name is specified it will be placed in the same folder as the original
- If only the path is specified, the new image will be placed in the destination with the same name as the original.
- If both the path and image name are specified it will placed in its own destination and given the new name.

Resizing the Original Image

If neither of the two preferences listed above (create_thumb, and new_image) are used, the resizing function will instead target the original image for processing.

$this->image_lib->crop()

The cropping function works nearly identically to the resizing function except it requires that you set preferences for the X and Y axis (in pixels) specifying where to crop, like this:

```
$config['x_axis'] = '100';
$config['y_axis'] = '40';
```

All preferences listed in the table above are available for this function except these: rotation, width, height, create_thumb, new_image.

Here's an example showing how you might crop an image:

```
$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();
}
```

Note: Without a visual interface it is difficult to crop images, so this function is not very useful unless you intend to build such an interface. That's exactly what we did using for the photo gallery module in ExpressionEngine, the CMS we develop. We added a JavaScript UI that lets the cropping area be selected.

$this->image_lib->rotate()

The image rotation function requires that the angle of rotation be set via its preference:

```
$config['rotation_angle'] = '90';
```

There are 5 rotation options:

90 - rotates counter-clockwise by 90 degrees.

180 - rotates counter-clockwise by 180 degrees.
270 - rotates counter-clockwise by 270 degrees.
hor - flips the image horizontally.
vrt - flips the image vertically.

Here's an example showing how you might rotate an image:

```
$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();
}
```

Image Watermarking

The Watermarking feature requires the GD/GD2 library.

Two Types of Watermarking

There are two types of watermarking that you can use:

Text: The watermark message will be generating using text, either with a True Type font that you specify, or using the native text output that the GD library supports. If you use the True Type version your GD installation must be compiled with True Type support (most are, but not all).

Overlay: The watermark message will be generated by overlaying an image (usually a transparent PNG or GIF) containing your watermark over the source image.

Watermarking an Image

Just as with the other function (resizing, cropping, and rotating) the general process for watermarking involves setting the preferences corresponding to the action you intend to perform, then calling the watermark function. Here is an example:

```
$config['source_image'] = '/path/to/image/mypic.jpg';
$config['wm_text'] = 'Copyright 2006 - John Doe';
$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();
```

The above example will use a 16 pixel True Type font to create the text "Copyright 2006 - John Doe". The watermark will be positioned at the bottom/center of the image, 20 pixels from the bottom of the image.

Note: In order for the image class to be allowed to do any processing, the image file must have file permissions of 777.

Watermarking Preferences

This table shown the preferences that are available for both types of watermarking (text or overlay)

Preference	Default Value	Options	Description
wm_type	text	type, overlay	Sets the type of watermarking that should be used.
source_image	None	None	Sets the source image name/path. The path must be a relative or absolute server path, not a URL.
dynamic_output	FALSE	TRUE/FALSE (boolean)	Determines whether the new image file should be written to disk or generated dynamically. Note: If you choose the dynamic setting, only one image can be shown at a time, and it can't be positioned on the page. It simply outputs the raw image dynamically to your browser, along with image headers.
quality	90%	1 - 100%	Sets the quality of the image. The higher the quality the larger the file size.
padding	None	A number	The amount of padding, set in pixels, that will be applied to the watermark to set it away from the edge of your images.
wm_vrt_alignment	bottom	top, middle, bottom	Sets the vertical alignment for the watermark image.
wm_hor_alignme	center	left, center,	Sets the horizontal alignment for the watermark i

nt		right	mage.
wm_vrt_offset	None	None	You may specify a horizontal offset (in pixels) to apply to the watermark position. The offset normally moves the watermark to the right, except if you have your alignment set to "right" then your offset value will move the watermark toward the left of the image.
wm_hor_offset	None	None	You may specify a horizontal offset (in pixels) to apply to the watermark position. The offset normally moves the watermark down, except if you have your alignment set to "bottom" then your offset value will move the watermark toward the top of the image.

Text Preferences

This table shown the preferences that are available for the text type of watermarking.

Preference	Default Value	Options	Description
wm_text	None	None	The text you would like shown as the watermark. Typically this will be a copyright notice.
wm_font_path	None	None	The server path to the True Type Font you would like to use CodeIgniter includes a font in the system/fonts folder. If you do not use this option, the native GD font will be used.
wm_font_size	16	None	The size of the text. Note: If you are not using the True Type option above, the number is set using a range of 1 - 5. Otherwise, you can use any valid pixel size for the font you're using.
wm_font_color	ffffff	None	The font color, specified in hex. Note, you must use the full 6 character hex value (ie, 993300), rather than the three character abbreviated version (ie fff).
wm_shadow_color	None	None	The color of the drop shadow, specified in hex. If you leave this blank a drop shadow will not be used. Note, you must use the full 6 character hex value (ie, 993300), rather than the three character abbreviated version (ie fff).
wm_shadow_distance	3	None	The distance (in pixels) from the font that the drop shadow should appear.

Overlay Preferences

This table shown the preferences that are available for the overlay type of watermarking.

Preference	Default Value	Options	Description

wm_overlay_path	None	None	The server path to the image you wish to use as your watermark. Required only if you are using the overlay method.
wm_opacity	50	1 - 100	Image opacity. You may specify the opacity (i.e. transparency) of your watermark image. This allows the watermark to be faint and not completely obscure the details from the original image behind it. A 50% opacity is typical.
wm_x_transp	4	A number	If your watermark image is a PNG or GIF image, you may specify a color on the image to be "transparent". This setting (along with the next) will allow you to specify that color. This works by specifying the "X" and "Y" coordinate pixel (measured from the upper left) within the image that corresponds to a pixel representative of the color you want to be transparent.
wm_y_transp	4	A number	Along with the previous setting, this allows you to specify the coordinate to a pixel representative of the color you want to be transparent.

Input Class

The Input Class serves two purposes:

It pre-processes global input data for security.

It provides some helper functions for fetching input data and pre-processing it.

Note: This class is initialized automatically by the system so there is no need to do it manually.

Security Filtering

The security filtering function is called automatically when a new controller is invoked. It does the following:

Destroys the global GET array. Since CodeIgniter does not utilize GET strings, there is no reason to allow it.

Destroys all global variables in the event register_globals is turned on.
Filters the POST/COOKIE array keys, permitting only alpha-numeric (and a few other) characters.
Provides XSS (Cross-site Scripting Hacks) filtering. This can be enabled globally, or upon request.
Standardizes newline characters to \n

XSS Filtering

CodeIgniter comes with a Cross Site Scripting Hack prevention filter which can either run automatically to filter all POST and COOKIE data that is encountered, or you can run it on a per item basis. By default it does **not** run globally since it requires a bit of processing overhead, and since you may not need it in all cases.

The XSS filter looks for commonly used techniques to trigger Javascript or other types of code that attempt to hijack cookies or do other malicious things. If anything disallowed is encountered it is rendered safe by converting the data to character entities.

Note: This function should only be used to deal with data upon submission. It's not something that should be used for general runtime processing since it requires a fair amount of processing overhead.

To filter data through the XSS filter use this function:

$this->input->xss_clean()

Here is an usage example:

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

If you want the filter to run automatically every time it encounters POST or COOKIE data you can enable it by opening your application/config/config.php file and setting this:

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

Note: If you use the form validation class, it gives you the option of XSS filtering as well.

Using POST, COOKIE, or SERVER Data

CodeIgniter comes with three helper functions that let you fetch POST, COOKIE or SERVER items. The main advantage of using the provided functions rather then fetching an item directly ($_POST['something']) is that the functions will check to see if the item is set and return false (boolean) if not. This lets you conveniently use data without having to test whether an item exists first. In other words, normally you might do something like this:

```
if ( ! isset($_POST['something']))
{
    $something = FALSE;
}
else
{
    $something = $_POST['something'];
}
```

With CodeIgniter's built in functions you can simply do this:

```
$something = $this->input->post('something');
```

The three functions are:

  $this->input->post()

$this->input->cookie()
$this->input->server()

$this->input->post()

The first parameter will contain the name of the POST item you are looking for:

```
$this->input->post('some_data');
```

The function returns FALSE (boolean) if the item you are attempting to retrieve does not exist.

The second optional parameter lets you run the data through the XSS filter. It's enabled by setting the second parameter to boolean TRUE;

```
$this->input->post('some_data', TRUE);
```

$this->input->cookie()

This function is identical to the post function, only it fetches cookie data:

```
$this->input->cookie('some_data', TRUE);
```

$this->input->server()

This function is identical to the above functions, only it fetches server data:

```
$this->input->server('some_data');
```

$this->input->ip_address()

Returns the IP address for the current user. If the IP address is not valid, the function will return an IP of: 0.0.0.0

```
echo $this->input->ip_address();
```

$this->input->valid_ip($ip)

Takes an IP address as input and returns TRUE or FALSE (boolean) if it is valid or not. Note: The $this->input->ip_address() function above validates the IP automatically.

```
if ( ! valid_ip($ip))
{
     echo 'Not Valid';
}
else
{
     echo 'Valid';
}
```

$this->input->user_agent()

Returns the user agent (web browser) being used by the current user. Returns FALSE if it's not available.

echo $this->input->user_agent();

装载类

装载，顾名思义，是用来装载元素。这些元素可以是库 (类) 视图文件， 助手文件， 插件， 或者是你自己的文件。

提示： 这个类是由系统初始化的，所以，没有必要自己手动初始化。

以下为这个类里面的函数：

$this->load->library('class_name')

这个函数是用来加载核心类。 class_name 是你要加载的类的名称。 提示： “类”和“库”是可替换使用的。

比如， 你想用 CodeIgniter 来发送邮件， 第一步就是在你的控制器里加载 email 类。

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

一但被加载，就可以使用该类了， 使用 $this->email->*some_function*()。 每一个库文件都在各自的说明文档里被详细地介绍， 所以如果你想使用某个类，就去查看相应的类信息。

参数可以使用数组的形式，作为第二个参数传递给类。

$this->load->view('file_name', $data, true/false)

这个函数是用来加载你的视图文件。 If you haven't read the Views section of the user guide it is recommended that you do since it shows you how this function is typically used.

The first parameter is required. It is the name of the view file you would like to load. Note: The .php file extension does not need to be specified unless you use something other then .php.

The second **optional** parameter can take an associative array or an object as input, which it runs through the PHP extract function to convert to variables that can be used in your view files. Again, read the Views page to learn how this might be useful.

The third **optional** parameter lets you change the behavior of the function so that it returns data as a string rather than sending it to your browser. This can be useful if you want to process the data in some way. If you set the parameter to true (boolean) it will return data. The default behavior is false, which sends it to your browser. Remember to assign it to a variable if you want the data returned:

$string = $this->load->view('myfile', '', true);

$this->load->database('options', true/false)

This function lets you load the database class. The two parameters are **optional**. Please see the database section for more info.

$this->load->scaffolding('table_name')

This function lets you enable scaffolding. Please see the scaffolding section for more info.

$this->load->vars($array)

This function takes an associative array as input and generates variables using the PHP extract function. This function produces the same result as using the second parameter of the $this->load->view() function above. The reason you might want to use this function independently is if you would like to set some global variables in the constructor of your controller and have them become available in any view file loaded from any function. You can have multiple calls to this function. The data get cached and merged into one array for conversion to variables.

$this->load->helper('file_name')

This function loads helper files, where file_name is the name of the file, without the _helper.php extension.

$this->load->plugin('file_name')

This function loads plugins files, where file_name is the name of the file, without the _plugin.php extension.

$this->load->file('filepath/filename', true/false)

This is a generic file loading function. Supply the filepath and name in the first parameter and it will open and read the file. By default the data is sent to your browser, just like a View file, but if you set the second parameter to true (boolean) it will instead return the data as a string.

$this->load->lang('file_name')

This function is an alias of the language loading function: $this->lang->load()

$this->load->config('file_name')

This function is an alias of the config file loading function: $this->config->load()

Language Class

The Language Class provides functions to retrieve language files and lines of text for purposes of internationalization.

In your CodeIgniter system folder you'll find one called language containing sets of language files. You can create your own language files as needed in order to display error and other messages in other languages.

Language files are typically stored in your system/language directory. Alternately you can create a folder called language inside your application folder and store them there. CodeIgniter will look first in your system/application/language directory. If the directory does not exist or the specified language is not located there CI will instead look in your global system/language folder.

Note: Each language should be stored in its own folder. For example, the English files are located at: system/language/english

Creating Language Files

Language files must be named with _lang.php as the file extension. For example, let's say you want to create a file containing error messages. You might name it: error_lang.php

Within the file you will assign each line of text to an array called $lang with this prototype:

$lang['language_key'] = "The actual message to be shown";

Note: It's a good practice to use a common prefix for all messages in a given file to avoid collisions with similarly named items in other files. For example, if you are creating error messages you might prefix them with error_

$lang['error_email_missing'] = "You must submit an email address";

$lang['error_url_missing'] = "You must submit a URL";
$lang['error_username_missing'] = "You must submit a username";

Loading A Language File

In order to fetch a line from a particular file you must load the file first. Loading a language file is done with the following code:

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

Where filename is the name of the file you wish to load (without the file extension), and language is the language set containing it (ie, english). If the second parameter is missing, the default language set in your application/config/config.php file will be used.

Fetching a Line of Text

Once your desired language file is loaded you can access any line of text using this function:

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

Where language_key is the array key corresponding to the line you wish to show.

Note: This function simply returns the line. It does not echo it for you.

Auto-loading Languages

If you find that you need a particular language globally throughout your application, you can tell CodeIgniter to auto-load it during system initialization. This is done by opening the application/config/autoload.php file and adding the language(s) to the autoload array.

Output Class

The Output class is a small class with one main function: To send the finalized web page to the requesting browser. It is also responsible for caching your web pages, if you use that feature.

Note: This class is initialized automatically by the system so there is no need to do it manually.

Under normal circumstances you won't even notice the Output class since it works transparently without your intervention. For example, when you use the Loader class to load a view file, it's automatically passed to the Output class, which will be called automatically by CodeIgniter at the end of system execution. It is possible, however, for you to manually intervene with the output if you need to, using either of the two following functions:

$this->output->set_output();

Permits you to manually set the final output string. Usage example:

```
$this->output->set_output($data);
```

Important: If you do set your output manually, it must be the last thing done in the function you call it from. For example, if you build a page in one of your controller functions, don't set the output until the end.

$this->output->get_output();

Permits you to manually retrieve any output that has been sent for storage in the output class. Usage example:

```
$string = $this->output->get_output();
```

Note that data will only be retrievable from this function if it has been previously sent to the output class by one of the CodeIgniter functions like $this->load->view().

$this->output->set_header();

Permits you to manually set server headers, which the output class will send for you when outputting the final rendered display. Example:

```
$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();

Permits you to enable/disable the Profiler, which will display benchmark and other data at the bottom of your pages for debugging and optimization purposes.

To enable the profiler place the following function anywhere within your Controller functions:

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

When enabled a report will be generated and inserted at the bottom of your pages.

To disable the profiler you will use:

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

Pagination Class

CodeIgniter's Pagination class is very easy to use, and it is 100% customizable, either dynamically or via stored preferences.

If you are not familiar with the term "pagination", it refers to links that allows you to navigate from page to page, like this:

« First   < 1 2 **3** 4 5 >   Last »

Example

Here is a simple example showing how to create pagination in one of your controller functions:

```
$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();
```

Notes:

The $config array contains your configuration variables. It is passed to the $this->pagination->initialize function as shown above. Although there are some twenty items you can configure, at minimum you need the three shown. Here is a description of what those items represent:

base_url This is the full URL to the controller class/function containing your pagination. In the example above, it is pointing to a controller called "Test" and a function called "page". Keep in mind that you can re-route your URI if you need a different structure.

total_rows This number represents the total rows in the result set you are creating pagination for. Typically this number will be the total rows that your database query returned.

per_page The number of items you intend to show per page. In the above example, you would be showing 20 items per page.

The create_links() function returns an empty string when there is no pagination to show.

Setting preferences in a config file

If you prefer not to set preferences using the above method, you can instead put them into a config file. Simply create a new file called the pagination.php, add the $config array in that file. Then save the file in: config/pagination.php and it will be used automatically. You will NOT need to use the $this->pagination->initialize function if you save your preferences in a config file.

Customizing the Pagination

The following is a list of all the preferences you can pass to the initialization function to tailor the display.

$config['uri_segment'] = 3;

The pagination function automatically determines which segment of your URI contains the page number. If you need something different you can specify it.

$config['num_links'] = 2;

The number of "digit" links you would like before and after the selected page number. For example, the number 2 will place two digits on either side, as in the example links at the very top of this page.

Adding Enclosing Markup

If you would like to surround the entire pagination with some markup you can do it with these two prefs:

$config['full_tag_open'] = '<p>';

The opening tag placed on the left side of the entire result.

$config['full_tag_close'] = '</p>';

The closing tag placed on the right side of the entire result.

Customizing the First Link

$config['first_link'] = 'First';

The text you would like shown in the "first" link on the left.

$config['first_tag_open'] = '<div>';

The opening tag for the "first" link.

$config['first_tag_close'] = '</div>';

The closing tag for the "first" link.

Customizing the Last Link

$config['last_link'] = 'Last';

The text you would like shown in the "last" link on the right.

$config['last_tag_open'] = '<div>';

The opening tag for the "last" link.

$config['last_tag_close'] = '</div>';

The closing tag for the "last" link.

Customizing the "Next" Link

$config['next_link'] = '&gt';

The text you would like shown in the "next" page link.

$config['next_tag_open'] = '<div>';

The opening tag for the "next" link.

$config['next_tag_close'] = '</div>';

The closing tag for the "next" link.

Customizing the "Previous" Link

$config['prev_link'] = '&lt';

The text you would like shown in the "previous" page link.

$config['prev_tag_open'] = '<div>';

The opening tag for the "previous" link.

$config['prev_tag_close'] = '</div>';

The closing tag for the "previous" link.

Customizing the "Current Page" Link

$config['cur_tag_open'] = '<b>';

The opening tag for the "current" link.

$config['cur_tag_close'] = '</b>';

The closing tag for the "current" link.

Customizing the "Digit" Link

$config['num_tag_open'] = '<div>';

The opening tag for the "digit" link.

$config['num_tag_close'] = '</div>';

The closing tag for the "digit" link.

Session Class

The Session class permits you maintain a user's "state" and track their activity while they browse your site. The Session class stores session information for each user as serialized (and optionally encrypted) data in a cookie. It can also store the session data in a database table for added security, as this permits the session ID in the user's cookie to be matched against the stored session ID. By default only the cookie is saved. If you choose to use the database option you'll need to create the session table as indicated below.

Note: The Session class does **not** utilize native PHP sessions. It generates its own session data, offering more flexibility for developers.

Initializing a Session

Sessions will typically run globally with each page load, so the session class must either be initialized in your controller constructors, or it can be auto-loaded by the system. For the most part the session class will run unattended in the background, so simply initializing the class will cause it to read, create, and update sessions.

To initialize the Session class manually in your controller constructor, use the $this->load->library function:

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

Once loaded, the Sessions library object will be available using: $this->session

How do Sessions work?

When a page is loaded, the session class will check to see if valid session data exists in the user's session cookie. If sessions data does **not** exist (or if it has expired) a new session will be created and saved in the cookie. If a session does exist, its information will be updated and the cookie will be updated.

It's important for you to understand that once initialized, the Session class runs automatically. There is nothing you need to do to cause the above behavior to happen. You can, as you'll see below, work with session data or even add your own data to a user's session, but the process of reading, writing, and updating a session is automatic.

What is Session Data?

A *session*, as far as CodeIgniter is concerned, is simply an array containing the following information:

- The user's unique Session ID (this is a statistically random string with very strong entropy, hashed with MD5 for portability)
- The user's IP Address
- The user's User Agent data (the first 50 characters of the browser data string)
- The "last activity" and "last visit" time stamps.

The above data is stored in a cookie as a serialized array with this prototype:

```
[array]
(
     'session_id'     => random hash,
     'ip_address'     => 'string - user IP address',
     'user_agent'     => 'string - user agent data',
     'last_activity' => timestamp,
     'last_visit'     => timestamp
)
```

If you have the encryption option enabled, the serialized array will be encrypted before being stored in the cookie, making the data highly secure and impervious to being read or altered by someone. More info regarding encryption can be found here, although the Session class will take care of initializing and encrypting the data automatically.

Note: Session cookies are only updated every five minutes to reduce processor load. If you repeatedly reload a page you'll notice that the "last activity" time only updates if five minutes or more has passed since the last time the cookie was written.

Retrieving Session Data

Any piece of information from the session array is available using the following function:

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

Where item is the array index corresponding to the item you wish to fetch. For example, to fetch the session ID you will do this:

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

Note: The function returns FALSE (boolean) if the item you are trying to access does not exist.

Adding Custom Session Data

A useful aspect of the session array is that you can add your own data to it and it will be stored in the user's cookie. Why would you want to do this? Here's one example:

Let's say a particular user logs into your site. Once authenticated, you could add their username and email address to the session cookie, making that data globally available to you without having to run a database query when you need it.

To add your data to the session array involves passing an array containing your new data to this function:

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

Where $array is an associative array containing your new data. Here's an example:

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

If you want to add userdata one value at a time, set_userdata() also supports this syntax.

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

Note: Cookies can only hold 4KB of data, so be careful not to exceed the capacity. The encryption process in particular produces a longer data string than the original so keep careful track of how much data you are storing.

Saving Session Data to a Database

While the session data array stored in the user's cookie contains a Session ID, unless you store session data in a database there is no way to validate it. For some applications that require little or no security, session ID validation may not be needed, but if your application requires security, validation is mandatory.

When session data is available in a database, every time a valid session is found in the user's cookie, a database query is performed to match it. If the session ID does not match, the session is destroyed. Session IDs can never be updated, they can only be generated when a new session is created.

In order to store sessions, you must first create a database table for this purpose. Here is the basic prototype (for MySQL) required by the session class:

```
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, PRIMARY KEY (session_id) );
```

Note: By default the table is called ci_sessions, but you can name it anything you want as long as you update the application/config/config.php file so that it contains the name you have chosen. Once you have created your database table you can enable the database option in your config.php file as follows:

$config['sess_use_database'] = TRUE;

Once enabled, the Session class will store session data in the DB.

Make sure you've specified the table name in your config file as well:

$config['sess_table_name'] = 'ci_sessions";

Note: The Session class has built-in garbage collection which clears out expired sessions so you do not need to write your own routine to do it.

Destroying a Session

To clear the current session:

$this->session->sess_destroy();

Session Preferences

You'll find the following Session related preferences in your application/config/config.php file:

Preference	Default	Options	Description
sess_cookie_name	ci_session	None	The name you world the session cookie saved as.
sess_expiration	7200	None	The number of seconds you would like the session to last. The default value is 2 hours (7200 seconds). If you would like a non-expiring session set the value to zero: 0
sess_encrypt_cookie	FALSE	TRUE/FALSE (boolean)	Whether to encrypt the session data.
sess_use_database	FALSE	TRUE/FALSE (boolean)	Whether to save the session data to a database. You must create the table before enabling this option.
sess_table_name	ci_sessions	Any valid SQL table name	The name of the session database table.
sess_match_ip	FALSE	TRUE/FALSE (boolean)	Whether to match the user's IP address when reading the session data. Note that some ISPs dynamically changes the IP, so if you want a non-expiring session you will likely set this to FALSE.
sess_match_useragent	TRUE	TRUE/FALSE (boolean)	Whether to match the User Agent when reading the session data.

Trackback Class

The Trackback Class provides functions that enable you to send and receive Trackback data.

If you are not familiar with Trackbacks you'll find more information here.

Initializing the Class

Like most other classes in CodeIgniter, the Trackback class is initialized in your controller using the $this->load->library function:

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

Once loaded, the Trackback library object will be available using: $this->trackback

Sending Trackbacks

A Trackback can be sent from any of your controller functions using code similar to this example:

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

$tb_data = array(
                'ping_url'   => 'http://some-site.com/trackback/456',
                'url'        => 'http://www.my-site.com/blog/entry/123',
                'title'      => 'The Title of My Entry',
                'excerpt'    => 'The entry content.',
                'blog_name'  => 'My Blog Name',
                'charset'    => 'utf-8'
                );
if ( ! $this->trackback->send($tb_data))
{
     echo $this->trackback->display_errors();
}
else
{
     echo 'Trackback was sent!';
}
```

Description of array data:

- **ping_url** - The URL of the site you are sending the Trackback to. You can send Trackbacks to multiple URLs by separating each URL with a comma.
- **url** - The URL to YOUR site where the weblog entry can be seen.
- **title** - The title of your weblog entry.
- **excerpt** - The content of your weblog entry. Note: the Trackback class will automatically send only the first 500 characters of your entry. It will also strip all HTML.

blog_name - The name of your weblog.
charset - The character encoding your weblog is written in. If omitted, UTF-8 will be used.

The Trackback sending function returns TRUE/FALSE (boolean) on success or failure. If it fails, you can retrieve the error message using:

$this->trackback->display_errors();

Receiving Trackbacks

Before you can receive Trackbacks you must create a weblog. If you don't have a blog yet there's no point in continuing.

Receiving Trackbacks is a little more complex than sending them, only because you will need a database table in which to store them, and you will need to validate the incoming trackback data. You are encouraged to implement a thorough validation process to guard against spam and duplicate data. You may also want to limit the number of Trackbacks you allow from a particular IP within a given span of time to further curtail spam. The process of receiving a Trackback is quite simple; the validation is what takes most of the effort.

Your Ping URL

In order to accept Trackbacks you must display a Trackback URL next to each one of your weblog entries. This will be the URL that people will use to send you Trackbacks (we will refer to this as your "Ping URL").

Your Ping URL must point to a controller function where your Trackback receiving code is located, and the URL must contain the ID number for each particular entry, so that when the Trackback is received you'll be able to associate it with a particular entry.

For example, if your controller class is called Trackback, and the receiving function is called receive, your Ping URLs will look something like this:

http://www.your-site.com/index.php/trackback/receive/entry_id

Where entry_id represents the individual ID number for each of your entries.

Creating a Trackback Table

Before you can receive Trackbacks you must create a table in which to store them. Here is a basic prototype for such a table:

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));

The Trackback specification only requires four pieces of information to be sent in a Trackback (url, title, excerpt, blog_name), but to make the data more useful we've added a few more fields in the above table schema (date, IP address, etc.).

Processing a Trackback

Here is an example showing how you will receive and process a Trackback. The following code is intended for use within the controller function where you expect to receive Trackbacks.

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

$this->load->database();
if ($this->uri->segment(3) == FALSE)
{
        $this->trackback->send_error("Unable to determine the entry ID");
}
if ( ! $this->trackback->receive())
{
        $this->trackback->send_error("The Trackback did not contain valid data");
}
$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();
```

Notes:

The entry ID number is expected in the third segment of your URL. This is based on the URI example we gave earlier:

http://www.your-site.com/index.php/trackback/receive/entry_id

Notice the entry_id is in the third URI, which you can retrieve using:

`$this->uri->segment(3);`

In our Trackback receiving code above, if the third segment is missing, we will issue an error. Without a valid entry ID, there's no reason to continue.

The $this->trackback->receive() function is simply a validation function that looks at the incoming data and makes sure it contains the four pieces of data that are required (url, title, excerpt, blog_name). It returns TRUE on success and FALSE on failure. If it fails you will issue an error message.

The incoming Trackback data can be retrieved using this function:

```
$this->trackback->data('item')
```

Where item represents one of these four pieces of info: url, title, excerpt, or blog_name

If the Trackback data is successfully received, you will issue a success message using:

```
$this->trackback->send_success();
```

Note: The above code contains no data validation, which you are encouraged to add.

模板解析器类

模板解析器类可以解析你的视图文件中的伪变量。它可以解析简单的变量或者以变量作为标签的结构。如果你以前没有用过模板引擎，那么伪变量如下所示：

```
<html>

<head>
<title>{blog_title}</title>
</head>
<body>
<h3>{blog_heading}</h3>
{blog_entries}
<h5>{title}</h5>
<p>{body}</p>
{/blog_entries}
</body>
</html>
```

这些变量不是标准的PHP变量，但是这样的以文本形式展现可以让你很容易地区分出它与PHP变量的不同之处。

Note: CodeIgniter 不 要求你一定要用这个类，虽然比在视图文件中用纯PHP环境要快一些。然而，有一些开发者宁愿使用模板引擎，以免和界面设计者在使用PHP上产生混乱。

Also Note: The Template Parser Class is **not** not a full-blown template parsing solution. We've kept it very lean on purpose in order to maintain maximum performance.

Initializing the Class

Like most other classes in CodeIgniter, the Parser class is initialized in your controller using the $this->load->library function:

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

Once loaded, the Parser library object will be available using: $this->parser

The following functions are available in this library:

$this->parser->parse()

This variable accepts a template name and data array as input, and it generates a parsed version. Example:

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

$data = array(
               'blog_title' => 'My Blog Title',
               'blog_heading' => 'My Blog Heading'
               );
$this->parser->parse('blog_template', $data);
```

The first parameter contains the name of the view file (in this example the file would be called blog_template.php), and the second parameter contains an associative array of data to be replaced in the template. In the above example, the template would contain two variables: {blog_title} and {blog_heading}

There is no need to "echo" or do something with the data returned by $this->parser->parse(). It is automatically passed to the output class to be sent to the browser. However, if you do want the data returned instead of sent to the output class you can pass TRUE (boolean) to the third parameter:

```
$string = $this->parser->parse('blog_template', $data, TRUE);
```

Variable Pairs

The above example code allows simple variables to be replaced. What if you would like an entire block of variables to be repeated, with each iteration containing new values? Consider the template example we showed at the top of the page:

```
<html>

<head>
<title>{blog_title}</title>
</head>
<body>
<h3>{blog_heading}</h3>
{blog_entries}
<h5>{title}</h5>
<p>{body}</p>
{/blog_entries}
</body>
</html>
```

In the above code you'll notice a pair of variables: {blog_entries} data... {/blog_entries}. In a case like this, the entire chunk of data between these pairs would be repeated multiple times, corresponding to the number of rows in a result.

Parsing variable pairs is done using the identical code shown above to parse single variables, except, you will add a multi-dimensional array corresponding to your variable pair data. Consider this example:

```
$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);
```

If your "pair" data is coming from a database result, which is already a multi-dimensional array, you can simply use the database result function:

```
$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);
```

Unit Testing Class

Unit testing is an approach to software development in which tests are written for each function in your application. If you are not familiar with the concept you might do a little googling on the subject.

CodeIgniter's Unit Test class is quite simple, consisting of an evaluation function and two result functions. It's not intended to be a full-blown test suite but rather a simple mechanism to evaluate your code to determine if it is producing the correct data type and result.

Initializing the Class

Like most other classes in CodeIgniter, the Unit Test class is initialized in your controller using the $this->load->library function:

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

Once loaded, the Unit Test object will be available using: $this->unit

Running Tests

Running a test involves supplying a test and an expected result to the following function:

$this->unit->run(test, expected result, 'test name');

Where test is the result of the code you wish to test, expected result is the data type you expect, and test name is an optional name you can give your test. Example:

$test = 1 + 1;

$expected_result = 2;
$test_name = 'Adds one plus one';
$this->unit->run($test, $expected_result, $test_name);

The expected result you supply can either be a literal match, or a data type match. Here's an example of a literal:

$this->unit->run('Foo', 'Foo');

Here is an example of a data type match:

$this->unit->run('Foo', 'is_string');

Notice the use of "is_string" in the second parameter? This tells the function to evaluate whether your test is producing a string as the result. Here is a list of allowed comparison types:

- is_string
- is_bool
- is_true
- is_false
- is_int
- is_numeric
- is_float
- is_double
- is_array
- is_null

Generating Reports

You can either display results after each test, or your can run several tests and generate a report at the end. To show a report directly simply echo or return the run function:

echo $this->unit->run($test, $expected_result);

To run a full report of all tests, use this:

```
echo $this->unit->report();
```

The report will be formatted in an HTML table for viewing. If you prefer the raw data you can retrieve an array using:

```
echo $this->unit->result();
```

Strict Mode

By default the unit test class evaluates literal matches loosely. Consider this example:

```
$this->unit->run(1, TRUE);
```

The test is evaluating an integer, but the expected result is a boolean. PHP, however, due to it's loose data-typing will evaluate the above code as TRUE using a normal equality test:

```
if (1 == TRUE) echo 'This evaluates as true';
```

If you prefer, you can put the unit test class in to strict mode, which will compare the data type as well as the value:

```
if (1 === TRUE) echo 'This evaluates as FALSE';
```

To enable strict mode use this:

```
$this->unit->use_strict(TRUE);
```

Enabling/Disabling Unit Testing

If you would like to leave some testing in place in your scripts, but not have it run unless you need it, you can disable unit testing using:

```
$this->unit->active(FALSE)
```

Creating a Template

If you would like your test results formatted differently then the default you can set your own template. Here is an example of a simple template. Note the required pseudo-variables:

```
$str = '

<table border="0" cellpadding="4" cellspacing="1">
    {rows}
        <tr>
        <td>{item}</td>
        <td>{result}</td>
        </tr>
    {/rows}
</table>';
```

```
$this->unit->set_template($str);
```

Note: Your template must be declared **before** running the unit test process.

URI 类

URI类提供了帮助你分割URI字符串的函数集合。如果你使用URI路由功能，那么你就可以通过分段来重新分发地址栏信息。

注意: URI类会被系统自动初始化，不必手动启用该功能。

$this->uri->segment(n)

它允许你重新分割一个详细的URI分段。 n 为你想要得到的段数。分割数按照从左至右的顺序编排。 例如，如果你完整的URL的地址如下：

http://www.your-site.com/index.php/news/local/metro/crime_is_up

每个URI分段的数字编号为:

1. news
2. local
3. metro
4. crime_is_up

默认情况下URI没有分段那么该函数返回 FALSE(布尔值)。如果分段信息丢失，Segment函数还有第二个参数用来设置你的默认值。例如，这样设置后，当发生错误时，函数就会返回数字“0”。

```
$product_id = $this->uri->segment(3, 0);
```

如果没有该参数你的代码必须得像下面这样写:

```
if ($this->uri->segment(3) === FALSE)
{
    $product_id = 0;
}
else
{
    $product_id = $this->uri->segment(3);
}
```

$this->uri->rsegment(n)

这个函数和上面的函数基本功能一样，不同点在于它允许你在开启CodeIgniter的URI路由功能时进行详细分割并重新分发URI信息

$this->uri->slash_segment(n)

这 个 函 数 几 乎 和 $this->uri->segment() 一 模 一 样 ， 不 过 它 可 以 通 过 第 二 个 参 数 （ " ","leading","both" ）给返回的URI参数加上“/”线，如果不使用该参数，就会在后面增加斜画线。例如：

$this->uri->slash_segment(3);

$this->uri->slash_segment(3, 'leading');
$this->uri->slash_segment(3, 'both');

返回:

segment/

/segment
/segment/

$this->uri->slash_rsegment(n)

这个函数功能同上面的函数，不同点在于它允许你在开启CodeIgniter的URI路由功能时进行详细分割并重新分发URI信息并可以增加斜线“/”。

$this->uri->uri_to_assoc(n)

你可以使用这个函数把每个分段信息以"标识字"=>"具体值"的形式存放在一个联合数组Array()里。注意这个URI:

index.php/user/search/name/joe/location/UK/gender/male

使用这个函数你可以把URI以如下原型翻转到联合数组中：

```
[array]
(
    'name' => 'joe'
    'location' => 'UK'
    'gender' => 'male'
)
```

函数第一个参数可以设置偏移量，默认设置为3，因为一般情况下你的URI包含 控制器名 / 函数名 作为第一个和第二个分段。例如：

$array = $this->uri->uri_to_assoc(3);

echo $array['name'];

第二个参数可以用来设置"标识字"，这样返回的数组总会包含索引里的标识字，甚至在丢失URI的情况下也是如此。例如：

$default = array('name', 'gender', 'location', 'type', 'sort');

$array = $this->uri->uri_to_assoc(3, $default);

如果URI不包含对应你所给标识字的具体值时，该索引的值会被设置为"FALSE"。

最后，如果相应的具体值找不到给定的标识字时(若URI段数为是一个单数时)具体值也会被设置为"FALSE"(布尔值)

$this->uri->ruri_to_assoc(n)

这个函数和上面的函数基本功能一样，不同点在于它允许你在开启CodeIgniter的URI路由功能时把URI信息分割并翻转到联合数组中去。

$this->uri->assoc_to_uri()

这个函数的功能和uri_to_assoc的功能找相反，它把数组里的信息翻转成URI地址，例如：

```
$array = array('product' => 'shoes', 'size' => 'large', 'color' => 'red');

$str = $this->uri->assoc_to_uri($array);
//  输出: product/shoes/size/large/color/red
```

$this->uri->uri_string()

返回一个包含完整URI信息的字符串。例如,假设以下为你的完整URL：

http://www.your-site.com/index.php/news/local/345

函数将返回如下字符串:

news/local/345

$this->uri->ruri_string(n)

这个函数和上面的函数基本功能一样，不同点在于它允许你在开启CodeIgniter的URI路由功能时把URI转换成对应的字符串。

$this->uri->total_segments()

返回总的URI的段数

$this->uri->total_rsegments(n)

这个函数和上面的函数基本功能一样，不同点在于它允许你在开启CodeIgniter的URI路由功能时返回URI的段数。

$this->uri->segment_array()

返回一个包含URI分段的数组。例如：

```
$segs = $this->uri->segment_array();

foreach ($segs as $segment)
{
    echo $segment;
    echo '<br />';
}
```

$this->uri->rsegment_array(n)

这个函数和上面的函数基本功能一样，不同点在于它允许你在开启CodeIgniter的URI路由功能时返回URI的数组。

User Agent Class

The User Agent Class provides functions that help identify information about the browser, mobile device, or robot visiting your site. In addition you can get referrer information as well as language and supported character-set information.

Initializing the Class

Like most other classes in CodeIgniter, the User Agent class is initialized in your controller using the $this->load->library function:

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

Once loaded, the object will be available using: $this->agent

User Agent Definitions

The user agent name definitions are located in a config file located at: application/config/user_agents.php. You may add items to the various user agent arrays if needed.

Example

When the User Agent class is initialized it will attempt to determine whether the user agent browsing your site is a web browser, a mobile device, or a robot. It will also gather the platform information if it is available.

```
$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.)
```

Function Reference

$this->agent->is_browser()

Returns TRUE/FALSE (boolean) if the user agent is a known web browser.

$this->agent->is_mobile()

Returns TRUE/FALSE (boolean) if the user agent is a known mobile device.

$this->agent->is_robot()

Returns TRUE/FALSE (boolean) if the user agent is a known robot.

Note: The user agent library only contains the most common robot definitions. It is not a complete list of bots. There are hundreds of them so searching for each one would not be very efficient. If you find that some bots that commonly visit your site are missing from the list you can add them to your application/config/user_agents.php file.

$this->agent->is_referral()

Returns TRUE/FALSE (boolean) if the user agent was referred from another site.

$this->agent->browser()

Returns a string containing the name of the web browser viewing your site.

$this->agent->version()

Returns a string containing the version number of the web browser viewing your site.

$this->agent->mobile()

Returns a string containing the name of the mobile device viewing your site.

$this->agent->robot()

Returns a string containing the name of the robot viewing your site.

$this->agent->platform()

Returns a string containing the platform viewing your site (Linux, Windows, OS X, etc.).

$this->agent->referrer()

The referrer, if the user agent was referred from another site. Typically you'll test for this as follows:

```
if ($this->agent->is_referral())
{
    echo $this->agent->referrer();
}
```

$this->agent->agent_string()

Returns a string containing the full user agent string. Typically it will be something like this:

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()

Lets you determine if the user agent accepts a particular language. Example:

```
if ($this->agent->accept_lang('en'))
{
    echo 'You accept English!';
}
```

Note: This function is not typically very reliable since some browsers do not provide language info, and even among those that do, it is not always accurate.

$this->agent->accept_charset()

Lets you determine if the user agent accepts a particular language. Example:

```
if ($this->agent->accept_charset('utf-8'))
{
    echo 'You browser supports UTF-8!';
}
```

Note: This function is not typically very reliable since some browsers do not provide character-set info, and even among those that do, it is not always accurate.

表单验证

在解释CodeIgniter的数据验证前，让我们先描述下理想情况：

显示一个表单；

你填写文字并点击提交按钮；

如果你提交的东西是残缺的，又或者漏填了必须填写的项，表单重新显示并包含你提交数据的一条错误信息；

这个过程继续，知道你提交了正确的信息

在这个过程中，程序必须：

检查必须的数据。

校验数据的格式是否正确，是否符合标准(比如，如果提交用户名，那么必须只能包含有效的字符。必须有最小的长度，并且不能超过允许的长度。 用户名不可以与已有用户名重复，或者不能为保留字，等等。 Etc.)

处理数据使之更安全。

需要的情况下预格式化数据（数据需要进行剪裁，HTML格式化？等等）。

准备数据，写入数据库。

尽管上面的过程没有什么复杂的地方，但这通常需要一定数量的代码来提交错误信息，各种控制结构放置在HTML文件里。创建简单表单验证，代码通常会非常凌乱而效能地下。

CodeIgniter提供了一个全面的解决框架，真正的精简你需要写的代码数量。当然也可以从你的HTML表单中分离出控制结构，使得它成为干净而灵活的代码。

预览

为了能使用CodeIgniter的表单验证功能，你需要做以下三件事情：

一个包含表单的 视图 文件。

一个包含提交“成功”信息的视图文件。

一个控制器函数，用来接受和处理提交的数据。

让我们创建这三个事件，使用注册表单作为样例：

表单页

使用一个文本编辑器，创建一个名叫myform.php的文件。在文中写入如下代码并保存到applications/views/ 文件夹下:

<html> <head> <title>My Form</title> </head> <body> <?=$this->validation->error_string; ?> <?=form_open('form'); ?> <h5>Username</h5> <input type="text" name="username" value="" size="50" /> <h5>Password</h5> <input type="text" name="password" value="" size="50" /> <h5>Password Confirm</h5> <input type="text" name="passconf" value="" size="50" /> <h5>Email Address</h5> <input type="text" name="email" value="" size="50" /> <div><input type="submit"

value="Submit" /></div> </form> </body> </html>

成功页

使用一个文本编辑器，创建一个名叫 formsuccess.php的文件。在文中写入如下代码并保存到applications/views/ 文件夹下:

<html> <head> <title>My Form</title> </head> <body> <h3>Your form was successfully submitted!</h3> <p><?=anchor('form', 'Try it again!'); ?></p> </body> </html>

控制器页

使用一个文本编辑器，创建一个名叫 form.php的文件。在文中写入如下代码并保存到applications/controllers/ 文件夹下:

<?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'); } } } ?>

试一试!

用与下面相似的URL地址测试下你的表单：

www.your-site.com/index.php/form/

如果你提交了表单，你会发现表单重载了。那是因为你还没有设置任何验证条件，这就是我们接下来要做的。

说明：

你会注意到一些关于上面页面的事情：

那个 页面 (myform.php) 是一个除了两个例外的标准页面：

- 它使用一个表单助手 来创建表单to create the form opening. 从技术上讲，这是没有必要的。你可以用标准的HTML来创建表单。然而，使用这个助手的好处在于：它为你生成了一个段URL，就是你在CONFIG文件里定义的那个。这使得你的程序在修改时可以更简单和灵活。
- 在表单的顶部，你会注意到一下的不同点： <?=$this->validation->error_string; ?>

验证类会使用这个变量返回并显示一些错误信息。如果没有信息将没有返回值。

这个控制器 (form.php) 有一个函数： index(). 这个函数初始化验证类并加载你的视图文件需要使用的 form helper 和 URL helper 。 当然也 运行 验证过程。这取决于验证是否成功，既不显示表单也没有返回成功信息。

如果还没有告诉验证类使之生效的信息，它会一直返回“FALSE”(布尔值 **FALSE)**作为默认值。**如果数据成功地接受你的条件并且它们没有任何错误，那么run()** 函数值返回**“TRUE”**。

设置验证条件

CodeIgniter可以让你设置任意数量的条件来控制给定的范围，循环，甚至同时它可以让你准备并预处理数据。让我们先看看它的用法，我们后面再具体解释。

在你的控制器 (form.php)里，仅加入下面的初始化函数：

$rules['username'] = "required";

$rules['password'] = "required";
$rules['passconf'] = "required";
$rules['email'] = "required";
$this->validation->set_rules($rules);

你的控制器现在看起来像下面这样：

<?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'); } } } ?>

现在提交空白的表单,你将会看到错误信息。如果你提交的表单包含有完全正确的信息，你将看到你的成功页。

注意： 当数据中仍然有错误时，表单内容不会被重新提交。一旦通过解释器的验证条件，我们将不得使之即刻完成。

改变出错信息的样式

默认情况下，系统会为每个错误提示信息添加一个段落的标识(<p>)。你可以很容易地通过编码去改变这些界定符号，在控制器里写入一下代码:

$this->validation->set_error_delimiters('<div class="error">', '</div>');

在这个例子中，我们将其改成了 div 标签。

顺序编排规则

CodeIgniter可以让你把所有的验证条件有序地串联起来。我们来试一下。把你的验证条件数据改成下面的样子：

$rules['username'] = "required|min_length[5]|max_length[12]";

$rules['password'] = "required|matches[passconf]";
$rules['passconf'] = "required";
$rules['email'] = "required|valid_email";

The above code requires that:

The username field be no shorter than 5 characters and no longer than 12.

The password field must match the password confirmation field.
The email field must contain a valid email address.

Give it a try!

Note: There are numerous rules available which you can read about in the validation reference.

Prepping Data

In addition to the validation functions like the ones we used above, you can also prep your data in various ways. For example, you can set up rules like this:

$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";

In the above, we are "trimming" the fields, converting the password to MD5, and running the username through the "xss_clean" function, which removes malicious data.

Any native PHP function that accepts one parameter can be used as a rule, like htmlspecialchars, trim, MD5, etc.

Note: You will generally want to use the prepping functions **after** the validation rules so if there is an error, the original data will be shown in the form.

Callbacks: Your own Validation Functions

The validation system supports callbacks to your own validation functions. This permits you to extend the validation class to meet your needs. For example, if you need to run a database query to see if the user is choosing a unique username, you can create a callback function that does that. Let's create a simple example.

In your controller, change the "username" rule to this:

$rules['username'] = "callback_username_check";

Then add a new function called username_check to your controller. Here's how your controller should look:

<?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; } } } ?>

Reload your form and submit it with the word "test" as the username. You can see that the form field data was passed to your callback function for you to process.

To invoke a callback just put the function name in a rule, with "callback_" as the rule prefix.

The error message was set using the $this->validation->set_message function. Just remember that the message key (the first parameter) must match your function name.

Note: You can apply your own custom error messages to any rule, just by setting the message similarly. For example, to change the message for the "required" rule you will do this:

$this->validation->set_message('required', 'Your custom message here');

Re-populating the form

Thus far we have only been dealing with errors. It's time to repopulate the form field with the submitted data. This is done similarly to your rules. Add the following code to your controller, just below your rules:

$fields['username'] = 'Username';

$fields['password'] = 'Password';
$fields['passconf'] = 'Password Confirmation';
$fields['email'] = 'Email Address';
$this->validation->set_fields($fields);

The array keys are the actual names of the form fields, the value represents the full name that you want shown in the error message.

The index function of your controller should now look like this:

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'); } }

Now open your myform.php view file and update the value in each field so that it has an object corresponding to its name:

<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>

Now reload your page and submit the form so that it triggers an error. Your form fields should be populated and the error messages will contain a more relevant field name.

Showing Errors Individually

If you prefer to show an error message next to each form field, rather than as a list, you can change your form so that it looks like this:

<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" />

If there are no errors, nothing will be shown. If there is an error, the message will appear, wrapped in the delimiters you have set (<p> tags by default).

Note: To display errors this way you must remember to set your fields using the $this->validation->set_fields function described earlier. The errors will be turned into variables that have "_error" after your field name. For example, your "username" error will be available at:

$this->validation->username_error.

Rule Reference

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

Rule	Parameter	Description	Example
required	No	Returns FALSE if the form element is empty.	
matches	Yes	Returns FALSE if the form element does not match the one in the parameter.	matches[form_item]
min_length	Yes	Returns FALSE if the form element is shorter then the parameter value.	min_length[6]
max_length	Yes	Returns FALSE if the form element is longer then the parameter value.	max_length[12]
exact_length	Yes	Returns FALSE if the form element is not exactly the parameter value.	exact_length[8]
alpha	No	Returns FALSE if the form element contains anything other than alphabetical characters.	
alpha_numeric	No	Returns FALSE if the form element contains anything other than alpha-numeric characters.	
alpha_dash	No	Returns FALSE if the form element contains any	

		thing other than alpha-numeric characters, underscores or dashes.
numeric	No	Returns FALSE if the form element contains anything other than numeric characters.
valid_email	No	Returns FALSE if the form element does not contain a valid email address.
valid_ip	No	Returns FALSE if the supplied IP is not valid.

Note: These rules can also be called as discreet functions. For example:

$this->validation->required($string);

Note: You can also use any native PHP functions that permit one parameter.

Prepping Reference

The following is a list of all the prepping functions that are available to use:

Name	Parameter	Description
xss_clean	No	Runs the data through the XSS filtering function, described in the Input Class page.
prep_for_form	No	Converts special characters so that HTML data can be shown in a form field without breaking it.
prep_url	No	Adds "http://" to URLs if missing.
strip_image_tags	No	Strips the HTML from image tags leaving the raw URL.
encode_php_tags	No	Converts PHP tags to entities.

Note: You can also use any native PHP functions that permit one parameter, like trim, htmlspecialchars, urldecode, etc.

Setting Custom Error Messages

All of the native error messages are located in the following language file: language/english/validation_lang.php

To set your own custom message you can either edit that file, or use the following function:

$this->validation->set_message('rule', 'Error Message');

Where rule corresponds to the name of a particular rule, and Error Message is the text you would like displayed.

Dealing with Select Menus, Radio Buttons, and Checkboxes

If you use select menus, radio buttons or checkboxes, you will want the state of these items to be retained in the event of an error. The Validation class has three functions that help you do this:

set_select()

Permits you to display the menu item that was selected. The first parameter must contain the name of the select menu, the second parameter must contain the value of each item. Example:

```
<select name="myselect">

<option value="one" <?= $this->validation->set_select('myselect', 'one'); ?> >One</option>
<option value="two" <?= $this->validation->set_select('myselect', 'two'); ?> >Three</option>
<option value="three" <?= $this->validation->set_select('myselect', 'three'); ?> >Three</option>
</select>
```

set_checkbox()

Permits you to display a checkbox in the state it was submitted. The first parameter must contain the name of the checkbox, the second parameter must contain its value. Example:

```
<input type="checkbox" name="mycheck" value="1" <?= $this->validation->set_checkbox('mycheck', '1'); ?> />
```

set_radio()

Permits you to display radio buttons in the state they were submitted. The first parameter must contain the name of the radio button, the second parameter must contain its value. Example:

```
<input type="radio" name="myradio" value="1" <?= $this->validation->set_radio('myradio', '1'); ?> />
```

XML-RPC and XML-RPC 服务器类

CodeIgniter 的 XML-RPC 类允许你发送请求到另一个服务器, 或者建立一个你自己的XML-RPC服务器来接受请求.

什么是XML-RPC?

这是一个非常简单的两台计算机使用XML通过互联网进行通信的方法. 一台计算机 , 我们称之为 客户端 , 发送一个XML-RPC **请求** 给另外一台计算机 , 我们称之为 服务器. 当服务器收到请求并加以处理,然后将 **结果** 返回给客户端.

例如,使用 MetaWeblog 的 API(应用程序接口) , 一个XML-RPC客户端(通常是桌面发布工具)会发送一个请求到运行在你网站的XML-RPC服务端. 这个请求有可能是要发布一篇新的网志,或者是要修改已存在的网志. 当XML-RPC服务器接收到请求,将检验请求并且决定调用何种类/方法来处理请求.一旦处理完毕,服务器将返回一个响应信息.

如果要查看更详细的规范,你可以浏览 XML-RPC 网站.

初始化类

像CodeIgniter的大多数类一样, XML-RPC 和 XML-RPCS 类 在你的控制器中调用 $this->load->library 方法来进行初始化:

使用以下代码载入XML-RPC类库:

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

一旦XML-RPC类被载入, XML-RPC 库的对象就可以使用了: $this->xmlrpc

To load the XML-RPC Server class you will use:

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

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

Once loaded, the xml-rpcs library object will be available using: $this->xmlrpcs

Note: When using the XML-RPC Server class you must load BOTH the XML-RPC class and the XML-RPC Server class.

Sending XML-RPC Requests

To send a request to an XML-RPC server you must specify the following information:

The URL of the server

The method on the server you wish to call

The *request* data (explained below).

Here is a basic example that sends a simple Weblogs.com ping to the Ping-o-Matic

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

$this->xmlrpc->server('http://rpc.pingomatic.com/', 80);
$this->xmlrpc->method('weblogUpdates.ping');
$request = array('My Photoblog', 'http://www.my-site.com/photoblog/');
$this->xmlrpc->request($request);
if ( ! $this->xmlrpc->send_request())
{
    echo $xmlrpc->display_error();
}
```

Explanation

The above code initializes the XML-RPC class, sets the server URL and method to be called (weblogUpdates.ping). The request (in this case, the title and URL of your site) is placed into an array for transportation, and compiled using the request() function. Lastly, the full request is sent. If the send_request() method returns false we will display the error message sent back from the XML-RPC Server.

Anatomy of a Request

An XML-RPC request is simply the data you are sending to the XML-RPC server. Each piece of data in a request is referred to as a request parameter. The above example has two parameters: The URL and title of your site. When the XML-RPC server receives your request, it will look for parameters it requires.

Request parameters must be placed into an array for transportation, and each parameter can be one of seven data types (strings, numbers, dates, etc.). If your parameters are something other than strings you will have to include the data type in the request array.

Here is an example of a simple array with three parameters:

```
$request = array('John', 'Doe', 'www.some-site.com');

$this->xmlrpc->request($request);
```

If you use data types other than strings, or if you have several different data types, you will place each parameter into its own array, with the data type in the second position:

```
$request = array (

                     array('John', 'string'),
                     array('Doe', 'string'),
                     array(FALSE, 'boolean'),
                     array(12345, 'int')
                   );
$this->xmlrpc->request($request);
```

The Data Types section below has a full list of data types.

Creating an XML-RPC Server

An XML-RPC Server acts as a traffic cop of sorts, waiting for incoming requests and redirecting them to the appropriate functions for processing.

To create your own XML-RPC server involves initializing the XML-RPC Server class in your controller where you expect the incoming request to appear, then setting up an array with mapping instructions so that incoming requests can be sent to the appropriate class and method for processing.

Here is an example to illustrate:

```
$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();
```

The above example contains an array specifying two method requests that the Server allows. The allowed methods are on the left side of the array. When either of those are received, they will be mapped to the class and method on the right.

In other words, if an XML-RPC Client sends a request for the new_post method, your server will load the My_blog class and call the new_entry function. If the request is for the update_post method, your server will load the My_blog class and call the update_entry function.

The function names in the above example are arbitrary. You'll decide what they should be called on your server, or if you are using standardized APIs, like the Blogger or MetaWeblog API, you'll use their function names.

Processing Server Requests

When the XML-RPC Server receives a request and loads the class/method for processing, it will pass an object to that method containing the data sent by the client.

Using the above example, if the new_post method is requested, the server will expect a class to exist with this prototype:

```
class My_blog extends Controller {
    function new_post($request)
    {
    }
}
```

The $request variable is an object compiled by the Server, which contains the data sent by the XML-RPC Client. Using this object you will have access to the *request parameters* enabling you to process the request. When you are done you will send a Response back to the Client.

Below is a real-world example, using the Blogger API. One of the methods in the Blogger API is getUserInfo(). Using this method, an XML-RPC Client can send the Server a username and password, in return the Server sends back information about that particular user (nickname, user ID, email address, etc.). Here is how the processing function might look:

```
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);
    }
}
```

Notes:

The output_parameters() function retrieves an indexed array corresponding to the request parameters sent by the client. In the above example, the output parameters will be the username and password.

If the username and password sent by the client were not valid, and error message is returned using send_error_message().

If the operation was successful, the client will be sent back a response array containing the user's info.

Formatting a Response

Similar to *Requests*, *Responses* must be formatted as an array. However, unlike requests, a response is an array **that contains a single item**. This item can be an array with several additional arrays, but there can be only one primary array index. In other words, the basic prototype is this:

```
$response = array('Response data', 'array');
```

Responses, however, usually contain multiple pieces of information. In order to accomplish this we must put the response into its own array so that the primary array continues to contain a single piece of data. Here's an example showing how this might be accomplished:

```
$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'
                 );
```

Notice that the above array is formatted as a struct. This is the most common data type for responses.

As with Requests, a response can be on of the seven data types listed in the Data Types section.

Sending an Error Response

If you need to send the client an error response you will use the following:

return $this->xmlrpc->send_error_message('123', 'Requested data not available');

The first parameter is the error number while the second parameter is the error message.

Creating Your Own Client and Server

To help you understand everything we've covered thus far, let's create a couple controllers that act as XML-RPC Client and Server. You'll use the Client to send a request to the Server and receive a response.

The Client

Using a text editor, create a controller called xmlrpc_client.php. In it, place this code and save it to your applications/controllers/ folder:

<textarea rows="32" cols="50" style="width: 100%;" class="textarea"><?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 '';

    print_r($this->xmlrpc->display_response());
    echo '

'; } } } ?></textarea>

Note: In the above code we are using a "url helper". You can find more information in the Helpers Functions page.

The Server

Using a text editor, create a controller called xmlrpc_server.php. In it, place this code and save it to your applications/controllers/ folder:

<textarea rows="30" cols="50" style="width: 100%;" class="textarea"><?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); } } ?></textarea>

Try it!

Now visit the your site using a URL similar to this:

www.your-site.com/index.php/xmlrpc_client/

You should now see the message you sent to the server, and its response back to you.

The client you created sends a message ("How's is going?") to the server, along with a reqest for the "Greetings" method. The Server receives the request and maps it to the "process" function, where a response is sent back.

XML-RPC Function Reference

$this->xmlrpc->server()

Sets the URL and port number of the server to which a request is to be sent:

$this->xmlrpc->server('http://www.sometimes.com/pings.php', 80);

$this->xmlrpc->timeout()

Set a time out period (in seconds) after which the request will be canceled:

$this->xmlrpc->timeout(6);

$this->xmlrpc->method()

Sets the method that will be requested from the XML-RPC server:

$this->xmlrpc->method('method');

Where method is the name of the method.

$this->xmlrpc->request()

Takes an array of data and builds request to be sent to XML-RPC server:

$request = array(array('My Photoblog', 'string'), 'http://www.yoursite.com/photoblog/');

$this->xmlrpc->request($request);

$this->xmlrpc->send_request()

The request sending function. Returns boolean TRUE or FALSE based on success for failure, enabling it to be used conditionally.

$this->xmlrpc->set_debug(TRUE);

Enables debugging, which will display a variety of information and error data helpful during development.

$this->xmlrpc->display_error()

Returns an error message as a string if your request failed for some reason.

echo $this->xmlrpc->display_error();

$this->xmlrpc->display_response()

Returns the response from the remote server once request is received. The response will typically be an associative array.

$this->xmlrpc->display_response();

$this->xmlrpc->send_error_message()

This function lets you send an error message from your server to the client. First parameter is the error number while the second parameter is the error message.

return $this->xmlrpc->send_error_message('123', 'Requested data not available');

$this->xmlrpc->send_response()

Lets you send the response from your server to the client. An array of valid data values must be sent with this method.

```
$response = array(
                    array(
                            'flerror' => array(FALSE, 'boolean'),
                            'message' => "Thanks for the ping!")
                         )
                    'struct');
return $this->xmlrpc->send_response($response);
```

Data Types

According to the XML-RPC spec there are seven types of values that you can send via XML-RPC:

int or *i4*

boolean
string
double
dateTime.iso8601
base64
struct (contains array of values)
array (contains array of values)

Zip 编码类

CodeIgniter的zip编码类允许你创建ZIP压缩文档。文档可以保存在你的桌面或者某个文件夹里

初始化

与其他CodeIgniter里的类一样，ZIP类在控制器里完成初始化工作，函数：$this->load->library

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

一旦加载，ZIP库对象可以用$this->zip来使用

使用样例

这个例子演示了如何压缩一个文件并保存到你服务器一个文件夹，然后下载到你的桌面上。

```
$name = 'mydata1.txt';

$data = 'A Data String!';
$this->zip->add_data($name, $data);
// Write the zip file to a folder on your server. Name it "my_backup.zip"
$this->zip->archive('/path/to/directory/my_backup.zip');
// Download the file to your desktop. Name it "my_backup.zip"
$this->zip->download('my_backup.zip');
```

函数参考

$this->zip->add_data()

Permits you to add data to the Zip archive. The first parameter must contain the name you would like given to the file, the second parameter must contain the file data as a string:

```
$name = 'my_bio.txt';

$data = 'I was born in an elevator...';
$this->zip->add_data($name, $data);
```

You are allowed multiple calls to this function in order to add several files to your archive. Example:

```
$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);
```

Or you can pass multiple files using an array:

```
$data = array(

                'mydata1.txt' => 'A Data String!',
                'mydata2.txt' => 'Another Data String!'
                );
$this->zip->add_data($data);
$this->zip->download('my_backup.zip');
```

If you would like your compressed data organized into sub-folders, include the path as part of the filename:

```
$name = 'personal/my_bio.txt';
$data = 'I was born in an elevator...';
$this->zip->add_data($name, $data);
```

The above example will place my_bio.txt inside a folder called personal.

$this->zip->add_dir()

Permits you to add a directory. Usually this function is unnecessary since you can place your data into folders when using $this->zip->add_data(), but if you would like to create an empty folder you can do so. Example:

```
$this->zip->add_dir('myfolder'); // Creates a folder called "myfolder"
```

$this->zip->read_file()

Permits you to compress a file that already exists somewhere on your server. Supply a file path and the zip class will read it and add it to the archive:

```
$path = '/path/to/photo.jpg';
$this->zip->read_file($path);
// Download the file to your desktop. Name it "my_backup.zip"
$this->zip->download('my_backup.zip');
```

If you would like the Zip archive to maintain the directory structure the file is in, pass TRUE (boolean) in the second parameter. Example:

```
$path = '/path/to/photo.jpg';
$this->zip->read_file($path, TRUE);
// Download the file to your desktop. Name it "my_backup.zip"
$this->zip->download('my_backup.zip');
```

In the above example, photo.jpg will be placed inside two folders: path/to/

$this->zip->read_dir()

Permits you to compress a folder (and its contents) that already exists somewhere on your server. Supply a file path to the directory and the zip class will recursively read it and recreate it as a Zip archive. All files contained within the supplied path will be encoded, as will any sub-folders contained within it. Example:

```
$path = '/path/to/your/directory/';

$this->zip->read_dir($path);
// Download the file to your desktop. Name it "my_backup.zip"
$this->zip->download('my_backup.zip');
```

$this->zip->archive()

Writes the Zip-encoded file to a directory on your server. Submit a valid server path ending in the file name. Make sure the directory is writable (666 or 777 is usually OK). Example:

```
$this->zip->archive('/path/to/folder/myarchive.zip'); // Creates a file named myarchive.zip
```

$this->zip->download()

Causes the Zip file to be downloaded to your server. The function must be passed the name you would like the zip file called. Example:

```
$this->zip->download('latest_stuff.zip'); // File will be named "latest_stuff.zip"
```

Note: Do not display any data in the controller in which you call this function since it sends various server headers that cause the download to happen and the file to be treated as binary.

$this->zip->get_zip()

Returns the Zip-compressed file data. Generally you will not need this function unless you want to do something unique with the data. Example:

```
$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()

The Zip class caches your zip data so that it doesn't need to recompile the Zip archive for each function you use above. If, however, you need to create multiple Zips, each with different data, you can clear the cache between calls. Example:

```
$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"); // Read the file's contents
$this->zip->download('myphotos.zip');
```

数组辅助函数

数组辅助函数的文件涵盖了一些用于辅助数组操作的函数。

装载本辅助函数

本辅助函数的装载通过如下代码完成：

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

可用的函数如下：

element()

获取数组中的元素。本函数测试数组的索引是否已设定并含有数值。如果已设有数值则返回该数值，否则返回 FALSE，或任何你设定的默认数值（函数第三个参数）。范例：

```
$array = array('color' => 'red', 'shape' => 'round', 'size' => '');
```

```
// 返回 "red"
echo element('color', $array);
// 返回 NULL
echo element('size', $array, NULL);
```

random_element()

根据提供的数组，随机返回该数组内的一个元素。使用范例：

```
$quotes = array(
                "I find that the harder I work, the more luck I seem to have. - Thomas Jefferson",
                "Don't stay in bed, unless you can make money in bed. - George Burns",
                "We didn't lose the game; we just ran out of time. - Vince Lombardi",
                "If everything seems under control, you're not going fast enough. - Mario Andretti",
                "Reality is merely an illusion, albeit a very persistent one. - Albert Einstein",
                "Chance favors the prepared mind - Louis Pasteur"
                );
echo random_element($quotes);
```

Cookie Helper

The Cookie Helper file contains functions that assist in working with cookies.

Loading this Helper

This helper is loaded using the following code:

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

The following functions are available:

set_cookie()

Sets a cookie containing the values you specify. There are two ways to pass information this function so that a cookie can be set: Array Method, and Discreet Parameters:

Array Method

Using this method, an associative array is passed to the first parameter:

```
$cookie = array(
                    'name'      => 'The Cookie Name',
                    'value'    => 'The Value',
                    'expire' => '86500',
                    'domain' => '.some-domain.com',
                    'path'      => '/',
                    'prefix' => 'myprefix_',
               );

set_cookie($cookie);
```

Notes:

Only the name and value are required.

The expiration is set in **seconds**, which will be added to the current time. Do not include the time, but rather only the number of seconds from *now* that you wish the cookie to be valid. If the expiration is set to zero the cookie will only last as long as the browser is open.

To delete a cookie set it with the expiration blank.

For site-wide cookies regardless of how your site is requested, add your URL to the **domain** starting with a period, like this: .your-domain.com

The path is usually not needed since the function sets a root path.

The prefix is only needed if you need to avoid name collisions with other identically named cookies for your server.

Discreet Parameters

If you prefer, you can set the cookie by passing data using individual parameters:

```
set_cookie($name, $value, $expire, $domain, $path, $prefix);
```

get_cookie()

Lets you fetch a cookie. The first parameter will contain the name of the cookie you are looking for:

get_cookie('some_cookie');

The function returns FALSE (boolean) if the item you are attempting to retrieve does not exist.

The second optional parameter lets you run the data through the XSS filter. It's enabled by setting the second parameter to boolean TRUE;

get_cookie('some_cookie', TRUE);

delete_cookie()

Lets you delete a cookie. Unless you've set a custom path or other values, only the name of the cookie is needed:

delete_cookie("name");

This function is otherwise identical to set_cookie(), except that it does not have the value and expiration parameters. You can submit an array of values in the first parameter or you can set discreet parameters.

delete_cookie($name, $domain, $path, $prefix)

Date Helper

The Date Helper file contains functions that help you work with dates.

Loading this Helper

This helper is loaded using the following code:

$this->load->helper('date');

The following functions are available:

now()

Returns the current time as a Unix timestamp, referenced either to your server's local time or GMT, based on the "time reference" setting in your config file. If you do not intend to set your master time reference to GMT (which you'll typically do if you run a site that lets each user set their own timezone settings) there is no benefit to using this function over PHP's time() function.

mdate()

This function is identical to PHPs date() function, except that it lets you use MySQL style date codes, where each code letter is preceded with a percent sign: %Y %m %d etc.

The benefit of doing dates this way is that you don't have to worry about escaping any characters that are not date codes, as you would normally have to do with the date() function. Example:

$datestring = "Year: %Y Month: %m Day: %d - %h:%i %a";

```
$time = time();
echo mdate($datestring, $time);
```

If a timestamp is not included in the second parameter the current time will be used.

standard_date()

Lets you generate a date string in one of several standardized formats. Example:

```
$format = 'DATE_RFC822';

$time = time();
echo standard_date($format, $time);
```

The first parameter must contain the format, the second parameter must contain the date as a Unix timestamp.

Supported formats:

- DATE_ATOM
- DATE_COOKIE
- DATE_ISO8601
- DATE_RFC822
- DATE_RFC850
- DATE_RFC1036
- DATE_RFC1123
- DATE_RFC2822
- DATE_RSS
- DATE_W3C

local_to_gmt()

Takes a Unix timestamp as input and returns it as GMT. Example:

```
$now = time();

$gmt = local_to_gmt($now);
```

gmt_to_local()

Takes a Unix timestamp (referenced to GMT) as input, and converts it to a localized timestamp based on the timezone and Daylight Saving time submitted. Example:

```
$timestamp = '1140153693';

$timezone = 'UM8';
$daylight_saving = TRUE;
echo gmt_to_local($timestamp, $timezone, daylight_saving);
```

Note: For a list of timezones see the reference at the bottom of this page.

mysql_to_unix()

Takes a MySQL Timestamp as input and returns it as Unix. Example:

```
$mysql = '20061124092345';

$unix = mysql_to_unix($mysql);
```

unix_to_human()

Takes a Unix timestamp as input and returns it in a human readable format with this prototype:

YYYY-MM-DD HH:MM:SS AM/PM

This can be useful if you need to display a date in a form field for submission.

The time can be formatted with or without seconds, and it can be set to European or US format. If only the timestamp is submitted it will return the time without seconds formatted for the U.S. Examples:

```
$now = time();

echo unix_to_human($now); // U.S. time, no seconds
echo unix_to_human($now, TRUE, 'us'); // U.S. time with seconds
echo unix_to_human($now, TRUE, 'eu'); // Euro time with seconds
```

human_to_unix()

The opposite of the above function. Takes a "human" time as input and returns it as Unix. This function is useful if you accept "human" formatted dates submitted via a form. Returns FALSE (boolean) if the date string passed to it is not formatted as indicated above. Example:

```
$now = time();

$human = unix_to_human($now);
$unix = human_to_unix($human);
```

timespan()

Formats a unix timestamp so that is appears similar to this:

1 Year, 10 Months, 2 Weeks, 5 Days, 10 Hours, 16 Minutes

The first parameter must contain a Unix timestamp. The second parameter must contain a timestamp that is greater that the first timestamp. If the second parameter empty, the current time will be used. The most common purpose for this function is to show how much time has elapsed from some point in time in the past to now. Example:

```
$post_date = '1079621429';

$now = time();
echo timespan($post_date, $now);
```

Note: The text generated by this function is found in the following language file: language/<your_lang>/date_lang.php

days_in_month()

Returns the number of days in a given month/year. Takes leap years into account. Example:

echo days_in_month(06, 2005);

If the second parameter is empty, the current year will be used.

timezone_menu()

Generates a pull-down menu of timezones, like this one:

This menu is useful if you run a membership site in which your users are allowed to set their local timezone value.

The first parameter lets you set the "selected" state of the menu. For example, to set Pacific time as the default you will do this:

echo timezone_menu('UM8');

Please see the timezone reference below to see the values of this menu.

The second parameter lets you set a CSS class name for the menu.

Note: The text contained in the menu is found in the following language file: language/<your_lang>/date_lang.php

Timezone Reference

The following table indicates each timezone and its location.

Time Zone	Location
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

目录辅助函数

目录辅助函数文件包含处理目录的函数。

载入这个辅助函数

请使用如下代码载入这个辅助函数：

$this->load->helper('directory');

以下函数可以使用：

directory_map('source directory')

这个函数将读取第一个参数所给出的路径的目录，并且返回该目录所包含文件的数据。示例如下：

$map = directory_map('./mydirectory/');

注意：路径总是相对于你的index.php文件。

如果目录含有子文件夹，也将被列出。如果你只想列出根目录，那么你就要设置第二个参数为true (boolean).

$map = directory_map('./mydirectory/', TRUE);

每一个文件夹的名字都将作为数组的索引，文件夹所包含的文件将以数字作为索引。下面有个典型的数组示例:

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
)
```

Download Helper

The Download Helper lets you download data to your desktop.

Loading this Helper

This helper is loaded using the following code:

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

The following functions are available:

force_download('filename', 'data')

Generates server headers which force data to be downloaded to your desktop. Useful with file downloads. The first parameter is the **name you want the downloaded file to be named**, the second parameter is the file data. Example:

```
$data = 'Here is some text!';
$name = 'mytext.txt';
force_download($name, $data);
```

If you want to download an existing file from your server you'll need to read the file into a string:

```
$data = file_get_contents("/path/to/photo.jpg"); // Read the file's contents
$name = 'myphoto.jpg';
force_download($name, $data);
```

文件助手函数

The File Helper file contains functions that assist in working with files.

Loading this Helper

This helper is loaded using the following code:

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

The following functions are available:

read_file('path')

Returns the data contained in the file specified in the path. Example:

```
$string = read_file('./path/to/file.php');
```

The path can be a relative or full server path. Returns FALSE (boolean) on failure.

Note: The path is relative to your main site index.php file, NOT your controller or view files. CodeIgniter uses a front controller so paths are always relative to the main site index.

If you server is running an open_basedir restriction this function might not work if you are trying to access a file above the calling script.

write_file('path', $data)

Writes data to the file specified in the path. If the file does not exist the function will create it. Example:

```
$data = 'Some file data';
if ( ! write_file('./path/to/file.php', $data))
{
     echo 'Unable to write the file';
}
else
{
     echo 'File written!';
```

}

You can optionally set the write mode via the third parameter:

```
write_file('./path/to/file.php', $data, 'r+');
```

The default mode is wb. Please see the PHP user guide for mode options.

Note: In order for this function to write data to a file its file permissions must be set such that it is writable (666, 777, etc.). If the file does not already exist, the directory containing it must be writable.

Note: The path is relative to your main site index.php file, NOT your controller or view files. CodeIgniter uses a front controller so paths are always relative to the main site index.

delete_files('path')

Deletes ALL files contained in the supplied path. Example:

```
delete_files('./path/to/directory/');
```

If the second parameter is set to true, any directories contained within the supplied root path will be deleted as well. Example:

```
delete_files('./path/to/directory/', TRUE);
```

Note: The files must be writable or owned by the system in order to be deleted.

get_filenames('path/to/directory/')

Takes a server path as input and returns an array containing the names of all files contained within it. The file path can optionally be added to the file names by setting the second parameter to TRUE.

Form Helper

The Form Helper file contains functions that assist in working with forms.

Loading this Helper

This helper is loaded using the following code:

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

The following functions are available:

form_open()

Creates an opening form tag with a base URL **built from your config preferences**. It will optionally let you add form attributes and hidden input fields.

The main benefit of using this tag rather than hard coding your own HTML is that it permits your site to be more portable in the event your URLs ever change.

Here's a simple example:

echo form_open('email/send');

The above example would create a form that points to your base URL plus the "email/send" URI segments, like this:

<form method="post" action="http:/www.your-site.com/index.php/email/send" />

Adding Attributes

Attributes can be added by passing an associative array to the second parameter, like this:

$attributes = array('class' => 'email', 'id' => 'myform');

echo form_open('email/send', $attributes);

The above example would create a form similar to this:

<form method="post" action="http:/www.your-site.com/index.php/email/send" class="email" id="myform" />

Adding Hidden Input Fields

Hidden fields can be added by passing an associative array to the third parameter, like this:

$hidden = array('username' => 'Joe', 'member_id' => '234');

echo form_open('email/send', '', $hidden);

The above example would create a form similar to this:

<form method="post" action="http:/www.your-site.com/index.php/email/send" class="email" id="myform" />

<input type="hidden" name="username" value="Joe" />
<input type="hidden" name="member_id" value="234" />

form_open_multipart()

This function is absolutely identical to the form_open() tag above except that it adds a multipart attribute, which is necessary if you would like to use the form to upload files with.

form_hidden()

Lets you generate hidden input fields. You can either submit a name/value string to create one field:

form_hidden('username', 'johndoe');

// Would produce:
<input type="hidden" name="username" value="johnodoe" />

Or you can submit an associative array to create multiple fields:

$data = array(

```
                'name'    => 'John Doe',
                'email' => 'john@some-site.com',
                'url'      => 'http://www.some-site.com'
             );
echo form_hidden($data);
// Would produce:
<input type="hidden" name="name" value="John Doe" />
<input type="hidden" name="email" value="john@some-site.com" />
<input type="hidden" name="url" value="http://www.some-site.com" />
```

form_input()

Lets you generate a standard text input field. You can minimally pass the field name and value in the first and second parameter:

```
echo form_input('username', 'johndoe');
```

Or you can pass an associative array containing any data you wish your form to contain:

```
$data = array(
                'name'               => 'username',
                'id'                 => 'username',
                'value'              => 'johndoe',
                'maxlength'          => '100',
                'size'               => '50',
                'style'              => 'width:50%',
             );

echo form_input($data);
// Would produce:
<input type="text" name="username" id="username" value="johndoe" maxlength="100" size="50"
style="width:50%" />
```

If you would like your form to contain some additional data, like JavaScript, you can pass it as a string in the third parameter:

```
$js = 'onClick="some_function()"';

echo form_input('username', 'johndoe', $js);
```

form_password()

This function is identical in all respects to the form_input() function above except that is sets it as a "password" type.

form_upload()

This function is identical in all respects to the form_input() function above except that is sets it as a "file" type, allowing it to be used to upload files.

form_textarea()

This function is identical in all respects to the form_input() function above except that it generates a "textarea" type. Note: Instead of the "maxlength" and "size" attributes in the above example, you will instead specify "rows" and "cols".

form_dropdown()

Lets you create a standard drop-down field. The first parameter will contain the name of the field, the second parameter will contain an associative array of options, and the third parameter will contain the value you wish to be selected. Example:

```
$options = array(
                  'small'    => 'Small Shirt',
                  'med'       => 'Medium Shirt',
                  'large'    => 'Large Shirt',
                  'xlarge' => 'Extra Large Shirt',
                );
echo form_dropdown('shirts', $options, 'large');
// Would produce:
<select name="shirts">
<option value="small">Small Shirt</option>
<option value="med">Medium Shirt</option>
<option value="large" selected>Large Shirt</option>
<option value="xlarge">Extra Large Shirt</option>
</select>
```

If you would like the opening <select> to contain additional data, like JavaScript, you can pass it as a string in the fourth parameter:

```
$js = 'onChange="some_function()"';
echo form_dropdown('shirts', $options, 'large', $js);
```

form_checkbox()

Lets you generate a checkbox field. Simple example:

```
echo form_checkbox('newsletter', 'accept', TRUE);
// Would produce:
<input type="checkbox" name="newsletter" value="accept" checked="checked" />
```

The third parameter contains a boolean TRUE/FALSE to determine whether the box should be checked or not.

Similar to the other form functions in this helper, you can also pass an array of attributes to the function:

```
$data = array(
                'name'          => 'newsletter',
                'id'            => 'newsletter',
                'value'         => 'accept',
                'checked'       => TRUE,
                'style'         => 'margin:10px',
            );
echo form_checkbox($data);
// Would produce:
<input type="checkbox" name="newsletter" id="newsletter" value="accept" checked="checked"
style="margin:10px" />
```

As with other functions, if you would like the tag to contain additional data, like JavaScript, you can pass it as a string in the fourth parameter:

```
$js = 'onClick="some_function()"';

echo form_checkbox('newsletter', 'accept', TRUE, $js)
```

form_radio()

This function is identical in all respects to the form_checkbox() function above except that is sets it as a "radio" type.

form_submit()

Lets you generate a standard submit button. Simple example:

```
echo form_submit('mysubmit', 'Submit Post!');

// Would produce:
<input type="submit" name="mysubmit" value="Submit Post!" />
```

Similar to other functions, you can submit an associative array in the first parameter if you prefer to set your own attributes. The third parameter lets you add extra data to your form, like JavaScript.

form_close()

Produces a closing </form> tag. The only advantage to using this function is it permits you to pass data to it which will be added below the tag. For example:

```
$string = "</div></div>";

echo form_close($string);
// Would produce:
```

```
</form>
</div></div>
```

form_prep()

Allows you to safely use HTML and characters such as quotes within form elements without breaking out of the form. Consider this example:

$string = 'Here is a string containing **"quoted"** text.';

```
<input type="text" name="myform" value="$string" />
```

Since the above string contains a set of quotes it will cause the form to break. The form_prep function converts HTML so that it can be used safely:

```
<input type="text" name="myform" value="<?php echo form_prep($string); ?>" />
```

Note: If you use any of the form helper functions listed in this page the form values will be prepped automatically, so there is no need to call this function. Use it only if you are creating your own form elements.

HTML辅助函数

HTML辅助函数的文件涵盖了一些用于HTML操作的函数。

装载本函数

本辅助函数的装载通过如下代码完成:

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

可用的函数如下：

heading()

帮助你创建 HTML <h1> 标签. 第一个字段用于标记显示内容，第二个字段用于标该标签所用字号。例如：

```
echo heading('Welcome!', 3);
```

如上代码将显示： <h3>Welcome!</h3>

ol() 和 ul()

允许你通过简单或多维的数组生成有序或无序的HTML列表。例如:

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

$list = array(
                'red',
                'blue',
                'green',
                'yellow'
                );
$attributes = array(
                        'class' => 'boldlist',
                        'id'     => 'mylist'
                        );
echo ul($list, $attributes);
```

如上代码将显示：

```
<ul class="boldlist" id="mylist">
  <li>red</li>
  <li>blue</li>
  <li>green</li>
  <li>yellow</li>
</ul>
```

这几是更复杂的例子，应用了多维数组：

```
$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);
```

如上代码将显示：

```
<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>
```

nbs()

生成不换行的指定个数的空格标签Generates non-breaking spaces (&nbsp;)。例如：

echo nbs(3);

如上代码将显示： &nbsp;&nbsp;&nbsp;

br()

生成指定个数的换行标签 (<br />) 。例如：

echo br(3);

如上代码将显示： <br /><br /><br />

Inflector 辅助函数

Inflector 辅助函数文件包含允许你把单词更改为复数、单数或骆驼拼写法等形式的函数。

装载辅助函数

使用以下代码装载辅助函数：

$this->load->helper('inflector');

以下函数可用：

singular()

把一个单词的复数形式更改为单数形式。例如：

$word = "dogs";

echo singular($word); // Returns "dog"

plural()

把一个单词的单数形式更改为复数形式。例如：

$word = "dog";

echo plural($word); // Returns "dogs"

强制单词以“es”结尾，则把第二个参数置为“true”。

$word = "pass";

echo plural($word, true); // Returns "passes"

camelize()

把一个以空格或下划线分隔的单词字符串更改为骆驼拼写法。例如：

$word = "my_dog_spot";

echo camelize($word); // Returns "myDogSpot"

underscore()

把以空格分隔的多个单词更改为以下划线分隔。例如：

$word = "my dog spot";

echo underscore($word); // Returns "my_dog_spot"

humanize()

把以下划线分隔的多个单词更改为以空格分隔，并且每个单词以大写开头。例如：

$word = "my_dog_spot";

echo humanize($word); // Returns "My Dog Spot"

Security Helper

The Security Helper file contains security related functions.

Loading this Helper

This helper is loaded using the following code:

$this->load->helper('security');

The following functions are available:

xss_clean()

Provides Cross Site Script Hack filtering. This function is an alias to the one in the Input class. More info can be found there.

dohash()

Permits you to create SHA1 or MD5 one way hashes suitable for encrypting passwords. Will create SHA1 by default. Examples:

$str = dohash($str); // SHA1

```
$str = dohash($str, 'md5'); // MD5
```

strip_image_tags()

This is a security function that will strip image tags from a string. It leaves the image URL as plain text.

```
$string = strip_image_tags($string);
```

encode_php_tags()

This is a security function that converts PHP tags to entities. Note: If you use the XSS filtering function it does this automatically.

```
$string = encode_php_tags($string);
```

Smiley Helper

The Smiley Helper file contains functions that let you manage smileys (emoticons).

Loading this Helper

This helper is loaded using the following code:

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

Overview

The Smiley helper has a renderer that takes plain text simileys, like :-) and turns them into a image representation, like 🙂

It also lets you display a set of smiley images that when clicked will be inserted into a form field. For example, if you have a blog that allows user commenting you can show the smileys next to the comment form. Your users can click a desired smiley and with the help of some JavaScript it will be placed into the form field.

Clickable Smileys Tutorial

Here is an example demonstrating how you might create a set of clickable smileys next to a form field. This example requires that you first download and install the smiley images, then create a controller and the View as described.

Important: Before you begin, please download the smiley images and put them in a publicly accessible place on your server. This helper also assumes you have the smiley replacement array located at application/config/smileys.php

The Controller

In your application/controllers/ folder, create a file called smileys.php and place the code below in it.

Important: Change the URL in the get_clickable_smileys() function below so that it points to your smiley folder.

You'll notice that in addition to the smiley helper we are using the Table Class.

```
<?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); } }
```

In your application/views/ folder, create a file called smiley_view.php and place this code in it:

```
<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>
```

When you have created the above controller and view, load it by visiting http://www.your=site.com/index.php/smileys/

Function Reference

get_clickable_smileys()

Returns an array containing your smiley images wrapped in a cliackable link. You must supply the URL to your smiley folder via the first parameter:

```
$image_array = get_clickable_smileys("http://www.your-site.com/images/smileys/");
```

js_insert_smiley()

Generates the JavaScript that allows the images to be clicked and inserted into a form field. The first parameter must contain the name of your form, the second parameter must contain the name of the form field. This function is designed to be placed into the <head> area of your web page.

```
<?php echo js_insert_smiley('blog', 'comments'); ?>
```

parse_smileys()

Takes a string of text as input and replaces any contained plain text smileys into the image equivalent. The first parameter must contain your string, the second must contain the the URL to your smiley folder:

```
$str = 'Here are some simileys: :-) ;-)'; $str = parse_smileys($str, "http://www.your-site.com/images/smileys/"); echo $str;
```

String Helper

The String Helper file contains functions that assist in working with strings.

Loading this Helper

This helper is loaded using the following code:

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

The following functions are available:

random_string()

Generates a random string based on the type and length you specify. Useful for creating passwords or generating random hashes.

The first parameter specifies the type of string, the second parameter specifies the length. The following choices are available:

- **alnum**: Alpha-numeric string with lower and uppercase characters.
- **numeric**: Numeric string.
- **nozero**: Numeric string with no zeros.
- **unique**: Encrypted with MD5 and uniquid(). Note: The length parameter is not available for this type. Returns a fixed length 33 character string.

Usage example:

```
echo random_string('alnum', 16);
```

alternator()

Allows two or more items to be alternated between, when cycling through a loop. Example:

```
for ($i = 0; $i < 10; $i++)
{
	echo alternator('string one', 'string two');
}
```

You can add as many parameters as you want, and with each iteration of your loop the next item will be returned.

```
for ($i = 0; $i < 10; $i++)
{
	echo alternator('one', 'two', 'three', 'four', 'five');
}
```

Note: To use multiple separate calls to this function simply call the function with no arguments to re-initialize.

repeater()

Generates repeating copies of the data you submit. Example:

$string = "\n";

echo repeater($string, 30);

The above would generate 30 newlines.

reduce_double_slashes()

Converts double slashes in a string to a single slash, except those found in http://. Example:

$string = "http://www.some-site.com//index.php";

echo reduce_double_slashes($string); // results in "http://www.some-site.com/index.php"

trim_slashes()

Removes any leading/trailing slashes from a string. Example:

$string = "/this/that/theother/";
echo trim_slashes($string); // results in this/that/theother

Text Helper

The Text Helper file contains functions that assist in working with text.

Loading this Helper

This helper is loaded using the following code:

$this->load->helper('text');

The following functions are available:

word_limiter()

Truncates a string to the number of **words** specified. Example:

$string = "Here is a nice text string consisting of eleven words.";

$string = word_limiter($string, 4);
// Returns: Here is a nice…

The third parameter is an optional suffix added to the string. By default it add an ellipsis.

character_limiter()

Truncates a string to the number of **characters** specified. It maintains the integrity of words so the character count may be slightly more or less then what you specify. Example:

$string = "Here is a nice text string consisting of eleven words.";

$string = character_limiter($string, 20);
// Returns: Here is a nice text string…

The third parameter is an optional suffix added to the string. By default it add an ellipsis.

ascii_to_entities()

Converts ASCII values to character entities, including high ASCII and MS Word characters that can cause problems when used in a web page, so that they can be shown consistently regardless of browser settings or stored reliably in a database. There is some dependence on your server's supported character sets, so it may not be 100% reliable in all cases, but for the most part it should correctly identify characters outside the normal range (like accented characters). Example:

$string = ascii_to_entities($string);

entities_to_ascii()

This function does the opposite of the previous one; it turns character entities back into ASCII.

word_censor()

Enables you to censor words within a text string. The first parameter will contain the original string. The second will contain an array of words which you disallow. The third (optional) parameter can contain a replacement value for the words. If not specified they are replaced with pound signs: ####. Example:

$disallowed = array('darn', 'shucks', 'golly', 'phooey');

$string = word_censor($string, $disallowed, 'Beep!');

highlight_code()

Colorizes a string of code (PHP, HTML, etc.). Example:

$string = highlight_code($string);

The function uses PHP's highlight_string() function, so the colors used are the ones specified in your php.ini file.

highlight_phrase()

Will highlight a phrase within a text string. The first parameter will contain the original string, the second will contain the phrase you wish to highlight. The third and fourth parameters will contain the opening/closing HTML tags you would like the phrase wrapped in. Example:

$string = "Here is a nice text string about nothing in particular.";

$string = highlight_phrase($string, "nice text", '<span style="color:#990000">', '</span>');

The above text returns:

Here is a nice textstring about nothing in particular.

word_wrap()

Wraps text at the specified **character** count while maintaining complete words. Example:

```
$string = "Here is a simple string of text that will help us demonstrate this function.";
```

```
echo word_wrap($string, 25);
// Would produce:
Here is a simple string
of text that will help
us demonstrate this
function
```

Typography Helper

The Typography Helper file contains functions that help your format text in semantically relevant ways.

Loading this Helper

This helper is loaded using the following code:

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

The following functions are available:

auto_typography()

Formats text so that it is semantically and typographically correct HTML. Takes a string as input and returns it with the following formatting:

- Surrounds paragraphs within <p></p> (looks for double line breaks to identify paragraphs).
- Single line breaks are converted to <br />, except those that appear within <pre> tags.
- Block level elements, like <div> tags, are not wrapped within paragraphs, but their contained text is if it contains paragraphs.
- Quotes are converted to correctly facing curly quote entities, except those that appear within tags.
- Apostrophes are converted to curly apostrophy entities.
- Double dashes (either like -- this or like--this) are converted to em—dashes.
- Three consecutive periods either preceding or following a word are converted to ellipsis…
- Double spaces following sentences are converted to non-breaking spaces to mimic double spacing.

Usage example:

```
$string = auto_typography($string);
```

Note: Typographic formatting can be processor intensive, particularly if you have a lot of content being formatted. If you choose to use this function you may want to consider caching your pages.

nl2br_except_pre()

Converts newlines to <br /> tags unless they appear within <pre> tags. This function is identical to the native PHP nl2br() function, except that it ignores <pre> tags.

Usage example:

```
$string = nl2br_except_pre($string);
```

URL 辅助函数

URL 辅助函数文件包含一些在处理 URL 中很有用的函数

加载辅助函数

本辅助函数通过如下代码加载:

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

可用函数如下:

site_url()

以一个字符串的形式返回在 config.php 中指定的 base_url 和 index.php (或者在 config.php 中设定的 index_page) 还有传递给函数的的 URI 段参数.

在生成 URL 的任何时候鼓都励您使用该函数, 即便以后 base_url 和 index_page 改动也会不会出问题.

做为参数传递给该函数的 URI 段可以是一个字符串，也可以是一个数组. 下面是一个字符串的例子:

```
echo site_url("news/local/123");
```

上面的例子将返回: http://www.your-site.com/index.php/news/local/123

这是一个以数组形式传递 URI 段的例子:

```
$segments = array('news', 'local', '123');

echo site_url($segments);
```

base_url()

返回在 config.php 中设定的 base_url. 例:

```
echo base_url();
```

index_page()

返回在 config.php 中设定的 index_page. 例:

echo index_page();

anchor()

Creates a standard HTML anchor link based on your local site URL:

Click Here

The tag has three optional parameters:

anchor(uri segments, text, attributes)

The first parameter can contain any segments you wish appended to the URL. As with the site_url() function above, segments can be a string or an array.

Note: If you are building links that are internal to your application do not include the base URL (http://...). This will be added automatically from the information specified in your config file. Include only the URI segments you wish appended to the URL.

The second segment is the text you would like the link to say. If you leave it blank, the URL will be used.

The third parameter can contain a list of attributes you would like added to the link. The attributes can be a simple string or an associative array.

Here are some examples:

echo anchor('news/local/123', 'My News');

Would produce: My News

echo anchor('news/local/123', 'My News', array('title' => 'The best news!'));

Would produce: My News

anchor_popup()

Nearly identical to the anchor() function except that it opens the URL in a new window. You can specify JavaScript window attributes in the third parameter to control how the window is opened. If the third parameter is not set it will simply open a new window with your own browser settings. Here is an example with attributes:

```
$atts = array(
                'width'      => '800',
                'height'     => '600',
                'scrollbars' => 'yes',
                'status'     => 'yes',
                'resizable'  => 'yes',
                'screenx'      => '0',
                'screeny'      => '0'
            );
echo anchor_popup(news/local/123, 'Click Me!', $atts);
```

Note: The above attributes are the function defaults so you only need to set the ones that are different from what you need. If you want the function to use all of its defaults simply pass an empty array in the third parameter:

```
echo anchor_popup('news/local/123', 'Click Me!', array());
```

mailto()

Creates a standard HTML email link. Usage example:

```
echo mailto('me@my-site.com', 'Click Here to Contact Me');
```

As with the anchor() tab above, you can set attributes using the third parameter.

safe_mailto()

Identical to the above function except it writes an obfuscated version of the mailto tag using ordinal numbers written with JavaScript to help prevent the email address from being harvested by spam bots.

auto_link()

Automatically turns URLs and email addresses contained in a string into links. Example:

```
$string = auto_link($string);
```

The second parameter determines whether URLs and emails are converted or just one or the other. Default behavior is both if the parameter is not specified

Converts only URLs:

```
$string = auto_link($string, 'url');
```

Converts only Email addresses:

```
$string = auto_link($string, 'email');
```

The third parameter determines whether links are shown in a new window. The value can be TRUE or FALSE (boolean):

```
$string = auto_link($string, 'both', TRUE);
```

url_title()

Takes a string as input and creates a human-friendly URL string. This is useful if, for example, you have a blog in which you'd like to use the title of your entries in the URL. Example:

```
$title = "What's wrong with CSS?";

$url_title = url_title($title);
// Produces: whats-wrong-with-css
```

The second parameter determines the word delimiter. By default dashes are used. Options are: dash, or underscore:

```
$title = "What's wrong with CSS?";

$url_title = url_title($title, 'underscore');
// Produces: whats_wrong_with_css
```

prep_url()

This function will add http:// in the event it is missing from a URL. Pass the URL string to the function like this:

```
$url = "www.some-site.com";

$url = prep_url($url);
```

redirect()

Does a "header redirect" to the local URI specified. Just like other functions in this helper, this one is designed to redirect to a local URL within your site. You will **not** specify the full site URL, but rather simply the URI segments to the controller you want to direct to. The function will build the URL based on your config file values.

The second parameter allows you to choose between the "location" method or the "refresh" method. Location is faster, but on Windows servers it can sometimes be a problem. Example:

```
if ($logged_in == FALSE)

{
        redirect('/login/form/', 'refresh');
}
```

Note: In order for this function to work it must be used before anything is outputted to the browser since it utilizes server headers.

XML Helper

The XML Helper file contains functions that assist in working with XML data.

xml辅助函数可以帮助你操作xml数据

Loading this Helper

This helper is loaded using the following code:

你需要用下面的方法加载xml辅助函数

$this->load->helper('xml');

The following functions are available:

然后你可以使用下面的函数

xml_convert('string')

Takes a string as input and converts the following reserved XML characters to entities:

给这个函数一个字符串参数，转成实体数据的时候它会保留一些xml的特殊字符如下：

Ampersands: &　　　符号名称：=and

Less then and greater than characters: < >　　　小于号和大于号
Single and double quotes: '　"　　单引号和双引号
Dashes: -　　短横线

This function ignores ampersands if they are part of existing character entities. Example:

如果部分存在这些特殊符号，此函数则会忽略掉这些符号。比如：

$string = xml_convert($string);

快速参考图

此图的 PDF 版本，单击这里。

CLASS REFERENCE

benchmark->

mark	
elapsed_time	
memory_usage	

calendar->

generate	
initialize	

config->

load	site_url
item	system_url
set_item	

db-> (Query)

query	insert_string
escape	update_string
escape_str	

db-> (Results)

result	num_fields
result_array	insert_id
row	affected_rows
num_rows	versions

db-> (Active Records)

get	groupby
select	having
from	orderby
join	limit
where	insert
orwhere	set
like	update
orlike	delete

db-> (Field Data)

field_names	
field_data	

db-> (Table Data)

tables	
table_exists	

db-> (Custom Function Calls)

call_function	

email->

from	message
reply_to	alt_message
to	clear
cc	send
bcc	attach
subject	print_debugger

encrypt->

encode	set_mode
decode	sha1
set_cypher	

upload->

do_upload	
display_errors	
data	

image_lib->

resize	watermark
crop	display_errors
rotate	

input->

xss_clean	ip_address
post	valid_ip
cookie	user_agent

load->

library	helper
database	plugin
scaffolding	file
view	lang
vars	config

lang->

load	
line	

output->

set_output	
get_output	

pagination->

initialize	
create_links	

session->

userdata	
set_userdata	

trackback->

send	send_error
display_errors	send_success
receive	data

parser->

parse	

uri->

segment	uri_string
slash_segment	total_segments
uri_to_assoc	segment_array
assoc_to_uri	

validation->

run	set_message
set_rules	set_fields
required	set_error_delimiters

xmlrpc->

server	set_debug
timeout	display_error
method	display_response
request	send_error_message
send_request	send_response

xmlrpcs->

initialize	
serve	

RESERVED FUNCTION NAMES

Controller
CI_Base
_ci_autoload
_ci_autoloader
_ci_assign_core
_ci_initialize
_ci_init_database
_ci_init_scaffolding
_ci_is_loaded
_ci_load
_ci_scaffolding
_ci_set_view_path

Add'l Reserved PHP4 Only

CI_Loader
config
database
file
helper
helpers
language
library
plugin
plugins
scaffolding
script
view
vars

Directory Structure

- ▼ system
  - ▼ application
    - ▶ config
    - ▶ controllers
    - ▶ errors
    - ▶ models
    - ▶ scripts
    - ▶ views
  - ▶ cache
  - ▶ codeigniter
  - ▶ init
  - ▶ drivers
  - ▶ fonts
  - ▶ logs
  - ▶ helpers
  - ▶ language
  - ▶ libraries
  - ▶ plugins
  - ▶ scaffolding
- ▶ user_guide

HELPER REFERENCE

array
random_element

cookie
set_cookie

date
now
mdate
local_to_gmt
gmt_to_local
mysql_to_unix
unix_to_human
human_to_unix
timespan
days_in_month
timezone_menu

directory
directory_map

file
read_file
write_file
delete_files

form
form_open
form_open_multipart
form_hidden
form_input
form_password
form_upload
form_textarea
form_dropdown
form_checkbox
form_radio
form_submit
form_close
form_prep

html
heading
nbs
br

security
xss_clean
hash
strip_image_tags
encode_php_tags

string
random_string
alternator
repeater

text
word_limiter
character_limiter
ascii_to_entities
entities_to_ascii
word_censor
highlight_code
highlight_phrase
word_wrap

typography
auto_typography
nl2br_except_pre

url
site_url
base_url
index_page
anchor
anchor_popup
mailto
safe_mailto
auto_link
url_title
prep_url
redirect

xml
xml_convert

Load Helpers with:
```
$this->load->helper('name');
```

Use Helpers just like any PHP function
```
<?=helper_name();?>
```

CodeIgniter Version 1.3 Quick Reference Chart

CodeIgniter™