ESCUELA DE INFORMÁTICA

PROGRAMA: TÉCNICO EN DESARROLLO DE SOFTWARE
SUBMÓDULO
ADMINISTRACIÓN DE SITIOS WEB
MANUAL CODEIGNITER

DP03 Ver. 02

Jaime Eduardo Montoya Montoya

---

**Introducción**

Este documento está basado en la documentación de https://ellislab.com/codeigniter/user-guide/ y de http://escodeigniter.com/guia_usuario/ para el framework CodeIgniter. En las referencias al final del texto, aparecen otros documentos que se utilizaron para la creación de este documento. Se tomaron los temas más relevantes para el curso. El estudiante puede revisar y consultar directamente las fuentes oficiales para extender aun más las posibilidades con la herramienta.

**Framework CodeIgniter**

Un framework es un programa para desarrollar otros programas, CodeIgniter, por tanto, es un programa o aplicación web desarrollada en PHP para la creación de cualquier tipo de aplicación web bajo PHP. Es un producto de código libre, libre de uso para cualquier aplicación.

Como cualquier otro framework, CodeIgniter contiene una serie de librerías que sirven para el desarrollo de aplicaciones web y además propone una manera de desarrollarlas que debemos seguir para obtener provecho de la aplicación. Esto es, marca una manera específica de codificar las páginas web y clasificar sus diferentes scripts, que sirve para que el código esté organizado y sea más fácil de crear y mantener. CodeIgniter implementa el proceso de desarrollo llamado Model View Controller (MVC), que es un estándar de programación de aplicaciones, utilizado tanto para hacer sitios web como programas tradicionales. Este sistema tiene sus características, que veremos más adelante.

CodeIgniter no lo hace todo por uno, pero contiene muchas ayudas para la creación de aplicaciones PHP avanzadas, que hacen que el proceso de desarrollo sea más rápido. A la vez, define una arquitectura de desarrollo que hará que programemos de una manera más ordenada ya que contiene diversas herramientas que ayudan a hacer aplicaciones más versátiles y seguras.

CodeIgniter y otros frameworks PHP pueden ayudar a dar el salto definitivo como desarrollador PHP, creando aplicaciones web más profesionales y con código más reutilizable, con la diferencia que CodeIgniter está creado para que sea fácil de instalar en cualquier servidor y de empezar a usar, más que cualquier otro framework. Además, muchas de sus utilidades y modos de funcionamiento son opcionales, lo que hace que se goce de mayor libertad a la hora de desarrollar sitios web.

**Características generales de CodeIgniter**

Algunos de los puntos más interesantes sobre este framework, sobre todo en comparación con otros productos similares, son los siguientes:

**Versatilidad:** Quizás la característica principal de CodeIgniter, en comparación con otros frameworks PHP. CodeIgniter es capaz de trabajar en la mayoría de los entornos o servidores, incluso en sistemas de alojamiento compartido, donde sólo se tiene acceso por FTP para enviar los archivos al servidor y donde no se tiene acceso a su configuración.

**Compatibilidad:** CodeIgniter es compatible con la versión PHP 4 y PHP 5. Sin embargo, desde la versión 2 de CodeIgniter ya solo es compatible con la versión 5 de PHP. La versión antigua del framework (como la 1.7 u otra) se encuentra disponible para los que aun utilicen PHP 4.

**Facilidad de instalación:** No es necesario más que la cuenta de FTP para subir CodeIgniter al servidor y su configuración se realiza con la edición de un archivo, donde debemos escribir aspetos tales como el acceso a la base de datos. Si la instalación es local, es similar, solo descomprimimos el paquete en la carpeta donde vamos a trabajar con el framework.

**Flexibilidad:** CodeIgniter es menos rígido que otros frameworks. Define una manera de trabajar específica, aunque podría no seguirse ya que sus reglas de codificación muchas veces las podemos saltar para trabajar como más a gusto nos sintamos. Algunos módulos como el uso de plantillas son totalmente opcionales. Esto ayuda también a que el aprendizaje sea más sencillo al principio.

**Ligereza:** El núcleo de CodeIgniter es bastante ligero, lo que permite que el servidor no se sobrecargue interpretando o ejecutando grandes porciones de código. La mayoría de los módulos o clases que ofrece se pueden cargar de manera opcional, sólo cuando se van a utilizar realmente.

**Documentación:** La documentación de CodeIgniter es fácil de seguir y de asimilar, porque está escrita en modo de tutorial. Esto no facilita mucho la referencia rápida, cuando ya sabemos acerca del framework y queremos consultar sobre una función o un método en concreto, pero para iniciar sin duda es muy importante.

**Instalación y configuración**
Lo primero es descargar el framework del sitio de CodeIgniter, del cual se obtiene un archivo empaquetado .zip. La versión disponible (marzo de 2013) es la 2.1.3. El sitio oficial de CodeIgniter, donde puede descargarse el framework, es:

*http://ellislab.com/codeigniter*

**Requisitos para trabajar con CodeIgniter**
Necesitamos disponer de PHP 5 para trabajar con CodeIgniter 2.x.x.

En lo que respecta al trabajo con bases de datos, CodeIgniter es compatible con varios motores, entre ellos MySQL (4.1 o posterior), MySQLi, MS SQL, PostgreSQL, Oracle, SQLite, y acceso a cualquier base de datos en entornos Windows por ODBC.

**Instalación de CodeIgniter**
- Descomprimir el paquete en la carpeta del sitio o del servidor (con lo que estará disponible para las aplicaciones que allí creemos)

- Configurar la URL base de la aplicación web
Es necesario indicarle a CodeIgniter la URL base de la aplicación, es decir, la URL para acceder a la raíz de CodeIgniter, según en el servidor y directorio donde se ha puesto el código del framework. Para ello abrimos el archivo de configuración, que se encuentra en ***application/config/config.php***, con cualquier editor de texto y cambiar la variable de configuración llamada que se guarda en ***$config['base_url']***.

Si la instalación es local, puede ponerse algo así: http://localhost/ y si está en un directorio específico, podría ser algo como http://localhost/directorio_codeigniter

Si hemos instalado el framework en un dominio de Internet podremos indicar algo como http://midominio.co/ y si creamos una carpeta para subir CodeIgniter en ella pondremos el nombre del dominio y luego el nombre de la carpeta o carpetas, separadas por barras y acabando siempre en una barra: http://midominio.co/carpeta/otracarpeta/

Según nos indican en el manual de instalación, aparte de este dato podemos opcionalmente escribir una llave de encriptación en la variable $config['encryption_key'], que servirá si deseamos usar la clase de encriptado que proporciona CodeIgniter o queremos que nuestras variables de sesión estén encriptadas, algo que hace el framework de manera transparente para nosotros.

- Configurar la base de datos
En este último paso hay que indicar los datos de acceso a la base de datos que se utilizará con CodeIgniter, ya que prácticamente todas las aplicaciones web que pueden crearse con el framework van a tener que utilizar la base de datos para algo. Para ello tenemos que editar el archivo ***application/config/database.php*** e indicar los parámetros de conexión al servidor de base de datos, como el nombre del servidor, el nombre de la base de datos, el motor (driver), el usuario y la contraseña entre otros.

Con esto ya se tiene listo el entorno de trabajo para comenzar a crear aplicaciones web PHP con el framework. Para verificar que CodeIgniter está funcionando, basta con acceder a la URL donde se instaló. Se muestra el mensaje de bienvenida de CodeIgniter, lo cual indica que está funcionando.

CodeIgniter define una serie de reglas de trabajo que son necesarias conocer antes de escribir líneas de código. Entendiendo la metodología de trabajo del framework, el desarrollo de aplicaciones se facilitará bastante y se comprenderá mejor las ventajas que ofrece.

**Flujo de aplicación de CodeIgniter**
En CodeIgniter existe un procedimiento para atender una solicitud de página del cliente. Este proceso se realiza internamente por el propio CodeIgniter y de manera transparente para el desarrollador. Durante el proceso participan varios módulos como el enrutamiento de la solicitud, la caché interna, etc., que ha medida que se avanza en su estudio se comprende mejor cual es su función.

La siguiente imagen es tomada de la documentación de CodeIgniter, donde se han traducido algunos nombres de los módulos que interactúan en el framework.


En resumen, para que se pueda entender el flujo de aplicación que implementa CodeIgniter, se pueden seguir los siguientes puntos:

1. Toda solicitud de una página a partir de CodeIgniter comienza en un index.php que hay en la raíz del framework.
2. Luego se realiza un filtrado de la URL para saber cuál es elemento que tiene que procesar esta página.

3. Si la página se había generado antes y está en la caché de CodeIgniter, se devuelve el archivo de la caché ya generado, con lo que se ahorra procesamientos repetidos. La caché se puede configurar y si lo deseamos, incluso deshabilitar.
4. Antes de continuar con el proceso se realiza un tratamiento de seguridad sobre la entrada que tengamos, tanto de la información que haya en la URL como de la información que haya en un posible POST, si lo hemos configurado así.
5. El controlador adecuado realiza el procesamiento de la solicitud. CodeIgniter decide el controlador que debe procesar la solicitud en función de la URL solicitada.
6. El controlador comunica con una serie de módulos, los que necesite, para producir la página.
7. A través de las vistas adecuadas, el controlador genera la página, tal cual se tiene que enviar al navegador.
8. Si la página no estaba en la caché, se introduce, para que las futuras solicitudes de esta página sean más rápidas.

Algunos de estos módulos, como la caché o el enrutamiento, funcionan de manera transparente para nosotros. Algunos otros, como los controladores, modelos y vistas, los tenemos que programar por nuestra cuenta y localizan cada una de las partes de nuestro programa que, al estar sepadadas nos ayudan a organizar también nuestro código. También tenemos a nuestra disposición diversas librerías, ayudantes (helpers) y plugins ya escritos en CodeIgniter con numerosas clases y funciones muy útiles para el desarrollo de aplicaciones web.

El múdulo de enrutamiento (routing) permite que cualquier URL que se solicite al servidor se ejecute en el controlador adecuado. La URL se analiza y los datos se procesan y aseguran antes de enviarlos al controlador adecuado, en el que simplemente tendremos que codificar sus diversos métodos.

**MVM Modelo - Vista - Controlador (Model - View - Controller)**

El Modelo, Vista, Controlador es típicamente utilizado para la creación de aplicaciones web y no sólo CodeIgniter lo implementa, sino también otra serie de frameworks de desarrollo web, en PHP u otros lenguajes. Es interesante porque separa en varios grupos las complejidades de las distintas partes que componen una página web, como la vista y la lógica, así como el acceso a la base de datos.

El Modelo - Vista - Controlador es un patrón de desarrollo o un estilo de arquitectura de software que separa el código fuente de las aplicaciones en tres grupos:

**Modelo**

Todo el código que tiene que ver con el acceso a la base de datos. En el modelo mantendremos encapsulada la complejidad de nuestra base de datos y simplemente crearemos funciones para recibir, insertar, actualizar o borrar información de nuestras tablas. Al mantenerse todas las llamadas a la base de datos en un mismo código, desde otras partes del programa podremos invocar las funciones que necesitemos del modelo y éste se encargará de procesarlas. En el modelo nos podrán preocupar cosas como el tipo de base de datos con la que trabajamos, o las tablas y sus relaciones, pero desde las otras partes del programa simplemente llamaremos a las funciones del modelo sin importarnos qué tiene que hacer éste para conseguir realizar las acciones invocadas.

**Vista**

La vista codifica y mantiene la presentación final de nuestra aplicación de cara al usuario. Es decir, en la vista colocaremos todo el código HTML, CSS, Javascript, etc. que se tiene que generar para producir la página tal cual queremos que la vea el usuario. En la práctica la vista no sólo sirve para producir páginas web, sino también cualquier otra salida que queramos enviar al usuario, en formatos o lenguajes distintos, como pueden ser XML, feeds RSS, archivos JSON, etc.

**Controlador**

El controlador podríamos decir que es la parte más importante, porque hace de enlace entre el modelo, la vista y cualquier otro recurso que se tenga que procesar en el servidor para generar la página web. En resumen, en el controlador guardamos la lógica de nuestras páginas y realizamos todas las acciones que sean necesarias para generarlas, ayudados del modelo o la vista.

**Nota**

Esto quiere decir, en la práctica para el caso de CodeIgniter, que según sea el ámbito donde estemos codificando, tendremos que escribir las líneas de código de cualquier página web en tres grupos de archivos distintos. En una aplicación estándar podremos tener varios modelos (por ejemplo, para cada una de las entidades distintas de la base de datos), varias vistas (una o varias para cada página o sección) y varios controladores (para cada página o sección de la web). Luego veremos dónde tenemos que guardar los archivos de cada uno de estos tres grupos.

Durante el desarrollo con CodeIgniter será muy recomendable seguir las normas del diseño Modelo - Vista - Controlador (MVC), pero realmente el framework es bastante flexible y permite que, si lo deseamos, no sigamos el desarrollo atendiendo a dicha arquitectura. En este caso, podríamos tener simplemente controladores y dentro de ellos realizar todas las acciones de acceso a la base de datos directamente, sin hacer llamadas al modelo, o escribir texto en la salida sin utilizar las vistas. Obviamente, esta opción no aprovechará las ventajas de separación de código entre presentación, lógica y modelo de base de datos.

En el caso que no utilizemos los modelos, no ocurrirá ningún efecto negativo en el desempeño del framework, pero en el caso de las vistas, si no las utilizamos y escribimos salida directamente desde el controlador, como por ejemplo con sentencias echo de PHP en el código de los controladores, perderemos algunas de las ventajas que CodeIgniter realiza por nosotros para procesar la salida antes de enviarla al usuario.

**Nota**

Separar la vista o presentación de la lógica de los programas es una ventaja importante, ya que ayuda a mantener un código más sencillo, reducido y con mayor facilidad de mantenimiento. En principio puede parecer un tanto complejo el hecho de manejar varios archivos con códigos distintos, pero a medio plazo nos vendrá muy bien separar el código de cada parte de nuestra aplicación. Nos evitará muchos de los problemas que a veces tenemos en desarrollos PHP, donde en un mismo archivo tenemos mezclado código en varios lenguajes como HTML, CSS, SQL, con el propio código PHP para

generar la lógica de nuestro negocio.

**URLs amigables a buscadores**

Uno de los puntos que debemos conocer antes de empezar a trabajar con CodeIgniter es sobre cómo son las direcciones URL que utiliza este popular framework PHP para cada una de las páginas de las aplicaciones PHP que creemos utilizándolo. La verdad es que es un punto que realmente resulta transparente para nosotros, puesto que las URL se generan automáticamente a medida que vamos programando el sitio web, por lo que vamos a comentar algunas aspectos sobre esto.

Uno de los puntos fuertes de este framework PHP es que las URL se presentan siempre en un formato amigable a los buscadores. Esto significa que cualquier motor de búsqueda puntuará positivamente, a priori, las direcciones de las páginas. Del mismo modo, las direcciones tendrán una forma fácil de entender y recordar por los seres humanos.

Por ejemplo, las siguientes URL's no suelen ser puntuadas bien por los buscadores:

http://www.midominio.com/ejemplo.php?ced=37

http://www.midominio.com/ejemplo.php?nombre=Evaristo

Si lo vemos, tenemos una página, ejemplo.php, que se le pasan distintos parámetros, pero los buscadores muchas veces interpretan que es la misma página. Con CodeIgniter las URL tienen mejor arquitectura, con formas como estas:

http://www.midominio.com/ejemplos/ejemplo/37

www.midominio.com/controlador/funcion/parametro

Las diferencias saltan a la vista, tanto para nosotros humanos como para motores de búsqueda como Google u otro. Y lo bueno es que nosotros no tenemos que hacer nada para conseguir este tipo de direcciones.

**Nota**

Las URL en CodeIgniter tienen el nombre de una página llamada index.php, pero esto es algo que podemos hacer desaparecer si sabemos configurar el framework.

**Query String desactivado**

En CodeIgniter en principio está desactivada la posibilidad de envío de variables a través de la URL, lo que se conoce en inglés como Query String. Es decir, que direcciones en las que se envían variables a

través de las URL, que decíamos que eran poco amigables a buscadores, no funcionarán.

Si se desea, se puede hacer que CodeIgniter reconozca las variables enviadas por la URL, pero como en principio el sistema de URLs amigables a buscadores que implementa el framework está pensado para poder evitar el problemático Query String, su uso está desactivado.

**Segmentos de la URL y el Modelo - Vista - Controlador**

Cada una de las partes de la URL de las aplicaciones creadas con el framework sirve para identificar qué controlador, del ya explicado Modelo - Vista - Controlador de CodeIgniter, se va a hacer cargo del procesamiento de la página, así como de la función que se invocará y los parámetros que se le enviarán a la misma. Por ejemplo:

aplicacioncodeiginter.com/facturacion/editarempresa/5610

aplicacioncodeiginter.com es el nombre del supuesto dominio donde tenemos CodeIgniter instalado.

facturacion es el nombre del controlador que se encargará de procesar la solicitud.

editarempresa es el nombre de la función que habrá dentro del controlador y donde estará el código que ejecute y genere la página. (Estrictamente hablando, en la terminología de la programación orientada a objetos POO, en vez de función, deberíamos llamarle método o función miembro).

Por último, 5610, es el parámetro que se le pasa a la función editarempresa, que servirá en este caso para que editarempresa sepa cuál es la empresa que se desea editar. Si queremos o necesitamos enviar varios parámetros a esta función, y no sólo el identificador de la empresa a editar, podremos colocarlos a continuación, separados por barras.

**Nota**

CodeIgniter pone a nuestra disposición una clase para trabajar con URLs llamada URI Class y una librería llamada URL Helper que contienen funciones para trabajar fácilmente con URLs y datos enviados en las mismas. En estas librerías hay funciones tan interesantes como site_url() que sirve para que el propio CodeIgniter cree una URL dentro del sitio a partir de un parámetro que le pasemos. Otro ejemplo es base_url(), que simplemente devuelve la URL raíz donde está nuestra aplicación CodeIgniter.

**Todo pasa por index.php**

En CodeIgniter existe un index.php que está en la raíz del framework que se encarga de las funciones de enrutamiento hacia el controlador que se debe encargar de procesar la solicitud. Por ello, de manera predeterminada en CodeIgniter veremos que las URLs incluyen el nombre del archivo index.php. Este comportamiento se puede configurar.

Más adelante veremos cómo eliminar este index.php en las URLs de CodeIgniter, algo que simplificará las direcciones.

**Añadir un sufijo a las URL**

Otro de los detalles que podemos hacer con CodeIgniter, que pueden personalizar aun más nuestras direcciones URL, es añadir un sufijo, que nosotros deseemos, al final de todas las URL que formen parte del framework.  Por ejemplo, podríamos desear que todas las URL acaben en .html o en .php, o como queramos.  Esto se puede hacer a través de los archivos de configuración del framework.

La idea es que una URL como esta:

http://dom.com/index.php/blog/post/cualquier-articulo

Pase a ser una dirección como esta otra:

http://dom.com/index.php/blog/post/cualquier-articulo.html

Para esto editamos el archivo de configuraciones generales:

application/config/config.php y editamos la variable url_suffix y colocar el valor que deseemos, por ejemplo:

```
$config['url_suffix'] = ".html";
```

**Controladores**

Los controladores son el corazón de tu aplicación, es el encargado de determinar, como las solicitudes HTTP deben ser manejadas.

Un controlador es simplemente un archivo de clase que es llamado en una forma que puede ser asociado con un URI.

considere esta URI:

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

En el ejemplo anterior, Codeigniter trataría de encontrar un controlador llamado blog.php y lo cargaría.

**Cuando el nombre de un controlador coincide con el primer segmento de una URI, esta será cargada.**

**Vamos a intentarlo: Hola mundo!**

Vamos a crear un controlador sencillo, así lo podrás ver en acción. Usando tu editor de texto, cree un archivo llamado blog.php, y coloca dentro el siguiente código:

```
<?php
class Blog extends CI_Controller {
        function index()
        {
                echo 'Hola mundo!';
        }
}
?>
```

Luego guarda el archivo en tu carpeta application/controllers/.

Ahora vista tu sitio usando una URL similar a esto:

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

Si lo has hecho correctamente, deberías ver `Hola mundo!`.

Nota: Los nombres de Clases deben empezar con una letra mayúscula. En otras palabras, esto es válido:

```
<?php
class Blog extends CI_Controller {

}
?>
```

Esto **no** es válido:

```
<?php
class blog extends CI_Controller {

}
?>
```

También, siempre debes asegurarte de que tu controlador herede de la clase padre "Controller" así podrá heredar todas sus funciones.

**Funciones**

En el ejemplo anterior el nombre de la función es index(). La función "index" es siempre cargada por defecto si el **segundo segmento** de la URI está vacía. Otra forma de mostrar tu mensaje "Hola mundo!" podría ser esta:

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

**El segundo segmento de la URI determina que función en el controlador será llamada.**

Vamos a probarlo. Agrega una nueva función a tu controlador:

```
<?php
class Blog extends CI_Controller {
        function index()
        {
                echo 'Hola mundo!';
        }
        function comments()
        {
                echo 'Mira esto!';
        }
}
?>
```

Ahora carga la siguiente URL para ver la función comment:

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

Deberias ver el siguiente mensaje. `Mira esto!`

**Pasando Segmentos URI a tus funciones**

Si tu URI contiene mas de dos segmentos, ellos seran pasados a tu función como parámetros.

Por ejemplo, digamos que tienes una URI como esta:

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

A tu funcion se le pasaran los segmentos URI 3 y 4 ("sandals" y "123"):

```
<?php
class Products extends CI_Controller {

        function shoes($sandals, $id)
        {
                echo $sandals;
                echo $id;
        }
}
?>
```

**Importante:** Si estas usando la característica URI Routing, los segmentos pasados a tus funciones seran re-ruteadas.

### Definiendo un Controlador por Defecto

Se le puede decir a CodeIgniter que cargue un controlador por defecto, cuando una URI no esta presente, ese sería el caso cuando solamente la raíz del sitio es llamada. Para especificar un controlador por defecto abre tu archivo application/config/routes.php y coloca este valor a la variable:

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

Donde *Blog* es el nombre de la clase del controlador que tú quieres usar. Si ahora cargas tu archivo principal index.php sin especificar ningún segmento URI podrás ver el mensaje Hola Mundo! por defecto.

### Remapeando llamadas a Funciones

Como señalamos anteriormente, el segundo segmento de la URI típicamente determina cual función en el controlador sera llamado. CodeIgniter te permite anular este comportamiento a través del uso de la función `_remap()`:

```
function _remap()
{
    // Algún codigo por aquí...
}
```

**Importante:** Si tu controlador contiene una función llamada `_remap()`, esta será **siempre** llamada independientemente de que contenga tu URI. Esto anula el normal comportamiento en el cual la URI determina que función es llamada, permitiéndote definir tus propias reglas de ruteo.

La llamada a la función anulada (típicamente el segundo segmento de la URI) será pasada como parámetro a la función `_remap()`:

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

### Procesando la Salida

CodeIgniter tiene una clase de salida que se encarga de enviar tus datos finales al navegador web automáticamente. Más información sobre esto puede ser encontrado en las páginas Views (vistas) y Output class (Clase Salida). En algunos casos, sin embargo, tu podrías querer post-procesar los datos finalizados en alguna forma para enviarlo al navegador por tu cuenta. CodeIgniter te lo permite agregando una función llamada _output() al controlador que recibe la salida de datos finalizada.

**Importante:** Si tu controlador contiene una función llamada `_output()`, esta sera **siempre** llamada por la clase "Output" en lugar de imprimir el dato finalizado directamente. El primer parametro de la función deberá contener la salida finalizada.

Aqui tenemos un ejemplo:

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

Por favor note que tu funcion _output() deberá recibir el dato en su estado finalizado. El punto de referencia y el uso de memoria, los datos serán prestados, los archivos cache serán escritos (si tu tienes habilitado el cacheo), y las cabeceras serán enviadas (si usa esa caraterística) antes de que esta sea entregada a la función _output(). Si estas usando esta característica el tiempo de ejecucíon de la página y el uso de memoria podría no ser preciso ya que no toma en cuenta ninguno de los procesos que tu haces. Para una alternativa a la forma de controlar la salida *antes* de que ninguno de los procesos finales estén hechos, por favor vea los métodos disponibles en la clase Output.

### Funciones Privadas

En algunos casos usted podría querer esconder ciertas funciones del acceso publico. Para hacer una funcion privada, simplemente agregue un guion bajo como prefijo y esta no podrá ser accedida via pedido URL. Por ejemplo, si tuvieras una función como esta:

```
function _utility()
{
  // algún codigo aquí
}
```

Tratar de accederlo via la URL, como esta, no funcionará:

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

### Organizando Tus Controladores en sub-carpetas

Si tu estas construyendo una gran aplicación usted podría encontrar conveniente organizar tus controladores en sub-carpetas. CodeIgniter te permite hacer esto.

Simplemente debes crear las carpetas dentro de tu directorio application/controllers y ubicar tus clases de controladores dentro.

**Nota:** Cuando usamos esta característica usando esta característica el primer segmento de tu URI debe especificar la carpeta. Por ejemplo, digamos que tienes un controlador ubicado aqui:

`application/controllers/products/shoes.php`

Para llamar al controlador anterior tu URI deberia ser algo similar a esto:

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

Cada una de tus sub-carpetas deberan contener un controlador por defecto el cual sera llamado si la URL contiene solamente la sub-carpeta. Simplemente nombra a tu controlador por defecto como lo especificaste en tu archivo application/config/routes.php

CodeIgniter tambien te permite remapear tus URIs usando su caracteristica URI Routing.

**Constructores de Clase**

Si tiene la intención de usar un constructor en alguno de sus controladores, **TIENE** que colocar la siguiente línea de código en él:

```
parent::__construct();
```

La razón de porqué esta línea es necesaria se debe a que su constructor local anulará al de su clase padre, por lo que necesitamos llamarlo manualmente.

```
<?php
class Blog extends CI_Controller {

public function __construct()
        {
                parent::__construct();
                // Su propio código de constructor
        }
}
?>
```

Los constructores son útiles si necesita establecer algunos valores por defecto, o ejecutar procesos por defecto cuando se instancia su clase. Los constructores no pueden devolver un valor, pero pueden hacer algún trabajo por defecto.

**Nombres de Funciones Reservadas**

Como sus clases controlador extenderán al controlador principal de la aplicación, tiene que ser cuidadoso de no nombrar a sus funciones del mismo modo que aquellas usadas por esa clase, sino sus funciones locales las anularán. Para conocer la lista completa, vea Nombres Reservados.

**Vistas**

Una vista es simplmenete una página web, o un fragmento de ella, como un encabezado, un píe de página, una barra lateral, etc. De hecho, las vistas pueden ser flexiblemente embebidas dentro de otras vistas (dentro de otras vistas, etc., etc.) si necesita este tipo de jerarquía.

Las vistas nunca son llamadas directamente, ellas deben ser cargadas por un controlador. Recuerda que en un entorno de trabajo MVC, el Controlador actua como el "policia de tránsito", así que es responsable de traer una vista en particular. Sino ha leído la página del Controlador debería hacerlo antes de continuar.

Usando el controlador que creó en la página de controlador, le permite agregar una vista a él.

**Crear una vista**

Usando su editor de texto, cree un archivo llamado vistablog.php, y ponga esto en él:

```
<html>
<head>
<title>Mi Blog</title>
</head>
<body>
        <h1>Bienvenido a mi Blog!</h1>
</body>
</html>
```

Entonces guarde el archivo en su carpeta application/views/.

**Cargando una Vista**

Para cargar un archivo de vista en particular, usará la siguiente función:

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

Donde *nombre* es el nombre de su archivo de vista. Nota: no es necesario especificar la extensión .php del archivo a menos que use una distinta de `.php`.

Ahora, abra el archivo controlador que hizo previamente llamado blog.php, y reemplace la sentencia "echo" con la función de carga de vista:

```
load->view('vistablog');
        }
}
?>
```

Si visita su sitio usando la URL que usó antes, debería ver su nueva vista. La URL era similar a esta:

`www.su-sitio.com/index.php/`*`blog`*`/`

**Cargando múltiples vistas**

CodeIgniter manejará inteligentemente múltiples llamadas a $this->load->view desde dentro de un controlador. Si más de una llamada ocurre serán agregados juntos. Por ejemplo, puede querer tener una

vista de encabezado, una vista de menu, una vista de contenido, y una vista de píe de página. Eso puede verse más o menos así:

```
<?php

class Pagina extends CI_Controller {

   function index()
   {
      $datos['titulo_pagina'] = 'Su titulo';
      $this->load->view('encabezado');
      $this->load->view('menu');
      $this->load->view('contenido', $datos);
      $this->load->view('pie_de_pagina');
   }

}
?>
```

En el ejemplo anterior, estamos usando "datos agregados dinámicamente", el cual verá debajo.

**Guardando Vistas dentro de Sub-carpetas**

Sus archivos de vista pueden ser guardados dentro de sub-carpetas si prefiere ese tipo de organización. Cuando lo haga, necesitará incluir el nombre de la carpeta al cargad la vista. Ejemplo:

```
$this->load->view('nombre_carpeta/nombre_archivo');
```

**Agregar Datos Dinámicos a la Vista**

Los datos son pasados desde el controlador a la vista de la forma de un **arreglo** o un **objeto** en el segundo parámetro de la función de carga de vistas. Aquí hay un ejemplo usando un arreglo:

```
$datos = array(
               'titulo' => 'Mi Titulo',
               'encabezado' => 'Mi Encabezado',
               'mensaje' => 'Mi Mensaje'
          );

$this->load->view('vistablog', $datos);
```

Y aquí hay un ejemplo usando un objeto:

```
$datos = new Someclass();
$this->load->view('vistablog', $datos);
```

Nota: si usa un objeto las variables de clase serán convertidas en un arreglo de elementos.

Probémoslo con su archivo controlador. Ábralo y agregue este código:

```
<?php
class Blog extends CI_Controller {
        function index()
        {
```

```
                $datos['titulo'] = "Mi Titulo Real";
                $datos['encabezado'] = "Mi Encabezado Real";

                $this->load->view('vistablog', $datos);
        }
}
?>
```

Ahora puede abrir su archivo de vista y cambiar el texto a variables que correspondan a las claves del arreglo de sus datos:

```
<html>
<head>
<title><?php echo $titulo;?></title>
</head>
<body>
        <h1><?php echo $encabezado;?></h1>
</body>
</html>
```

Entonces cargue la página en la URL que viene usando y debería ver las variables reemplazadas.

**Creando Iteraciones**

El arreglo de datos que le pasa a su archivo de vista no está limitado a variables simples. Puede pasar arreglos multidimensionales, los que serán iterados para generar múltiples filas. Por ejemplo, si trajo datos de su base de datos tipicalmente será en la forma de un arreglo múltidimensional.

Aquí hay un ejemplo simple. Agregue esto a su controlador:

```
<?php
class Blog extends CI_Controller {
        function index()
        {
                $datos['lista_de_tareas'] = array('Limpiar la Casa',
'Llamar a Mamá', 'Hacer un Recado');
                $datos['titulo'] = "Mi Título Real";
                $datos['encabezado'] = "My Encabezado Real";

                $this->load->view('vistablog', $datos);
        }
}
?>
```

Ahora abra su archivo de vista y cree la iteración:

```
<html>
<head>
<title><?php echo $titulo;?></title>
</head>
<body>
```

```
<h1><?php echo $encabezado;?></h1>

<h3>Mi Lista de Tareas</h3>
<ul>
<?php foreach($lista_de_tareas as $item):?>
<li><?php echo $item;?></li>
<?php endforeach;?>
</ul>

</body>
</html>
```

**Nota:** Notará que en el ejemplo de arriba estamos usando la sintaxis alternativa de PHP. Si no está familiarizado con ella, puede leer acerca de ella aquí.

**Devolver Vistas como Datos**

Hay un tercer parámetro **opcional** que le permite cambiar el comportamiento de la función para que devuelva datos como una cadena en lugar de enviarla al navegador. Esto puede ser útil si desea procesar los datos en alguna forma. Si establece el parámetro a `TRUE` (booleano), devolverá datos. El comportamiento por defecto `FALSE`, la enviará al navegador. Recuerde asignarla a una variable si quiere que devuelva datos:

```
$string = $this->load->view('mi_archivo', '', TRUE);
```

**Modelos**

**Qué es un modelo?**

Los modelos son clases PHP que se han diseñado para trabajar con la información en su base de datos. Por ejemplo, digamos que usted usa CodeIgniter para administrar un blog. Puede que tenga una clase de modelo que contiene funciones para insertar, actualizar y recuperar datos de su blog. Aquí está un ejemplo de lo que podría ser la clase del modelo:

```
class Blogmodel extends CI_Model{

        var $title   = '';
        var $content = '';
        var $date    = '';

        function __construct()
        {
                // Llamando al contructor del Modelo
                parent::__construct();
        }

        function get_last_ten_entries()
        {
                $query = $this->db->get('entries', 10);
                return $query->result();
        }

        function insert_entry()
        {
                $this->title   = $_POST['title']; // por favor leer
la nota de abajo
                $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']));
        }

}
```

Nota: Las funciones en el ejemplo anterior usan funciones de base de datos Active Record.

**Nota:** En aras de la simplicidad, en este ejemplo usamos $_POST directamente. Esto es generalmente una mala práctica, y el enfoque más común sería usar la Clase Input $this->input->post('title')

### Anatomia de un Modelo

La clases de Modelo estan almacenadas en su carpeta application/models/. Se puede anidar dentro de sub-carpetas si desea esta organización.

El prototipo básico para una clase de Modelo es esta:

```
class Model_name extends CI_Model {

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

Donde *Model_name* es el nombre de su clase. Los nombre de clase **TIENEN QUE TENER** la primera letra en mayúscula con el resto del nombre en minúscula. Asegúrese de que su clase extiende el modelo de base de la clase.

El nombre del archivo será la versión en minúscula de su nombre de clase. Por ejemplo, si su clase es esta:

```
class User_model extends CI_Model {

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

su archivo será este:

```
application/models/user_model.php
```

### Cargando un Modelo

Sus modelos suelen ser cargados y llamados desde sus funciones de controlador. Para carga un modelo debe usar la siguiente función:

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

Si su modelo esta localizado en una sub-carpeta, incluya la ruta relativa de su carpeta de modelos. Por ejemplo, si tiene un modelo localizado en application/models/blog/queries.php cargará usando esto:

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

Una vez cargado, tendrá que acceder a su funciones de modelo utilizando un objeto con el mismo nombre que su clase:

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

```
$this->Model_name->function();
```

Si desea que su modelo este asignado a un nombre de objeto diferente, ud. lo puede especificar por medio del segundo parámetro de la función de carga:

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

$this->fubar->function();
```

Aquí hay un ejemplo de un controlador que carga un modelo y luego sirve a una vista:

```
class Blog_controller extends CI_Controller {

    function blog()
    {
        $this->load->model('Blog');

        $data['query'] = $this->Blog->get_last_ten_entries();

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

**Cargar un Modelo Automáticamente**

Si encuentra que necesita tener un modelo disponible globalmente a lo largo de su aplicación, puede decirle a CodeIgniter que lo cargue automáticamente durante la inicialización del sistema. Esto se hace al abrir el archivo application/config/autoload.php y agregando el modelo al array $autoload.

**Conexión a su base de datos**

Cuando se carga un modelo, éste **NO** se conecta automáticamente a su base de datos. Las siguientes opciones de conexión están habilitadas para ud:

- Puede conectarse usando los métodos de base de datos standard descriptos aquí, ya sea dentro de su clase Controlador o su clase Modelo.
- Puede decirle a la función de autocarga del modelo para auto-conectarse pasándole `TRUE` (boolean) a través del tercer parámetro, y la configuración de conectividad, tal como se define en su archivo de configuración de la base de datos a ser utilizado: `$this->load->model('Model_name', '', TRUE);`
- Puede pasar manualmente la configuración de la conectividad de base de datos a través del tercer parámetro:

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

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

**Funciones Helper**

Los helpers, como el nombre sugiere, le ayudan con tareas. Cada archivo helper es simplemente una colección de funciones de una categoría particular. Existe un Helper de URL, que ayuda en la creación de hipervínculos, está el Helper de Formularios que ayuda a crear elementos de formularios, , Helper de Texto que realiza varias rutinas de formato,Helper de Cookie establece y lee cookies, Helper de Archivos ayuda a trabajar con archivos, etc.

A diferencia de muchos otros sistemas en CodeIgniter, los Helpers no están escritos en un formato Orientado a Objetos. Son simples funciones de procedimiento. Cada función helper realiza una tarea específica, sin dependencia en otras funciones.

CodeIgniter no carga Archivos Helper por defecto, así que el primer paso en el uso de Helpers es cargarlos. Una vez cargados, se vuelven disponibles globalmente en su controlador y vistas.

Los helpers son típicamente guardados en su directorio system/helpers. Alternativamente puede crear una carpeta llamada `helpers` dentro de su carpeta `application` y guardarlos allí. CodeIgniter primero buscará en su direcotrio system/application/helpers. Si el directorio no existe o el helper especificado no está ubicado allí, CI buscará entonces en su carpeta global system/helpers.

**Cargando un Helper**

Cargar un archivo helper es bastante simple usando la función siguiente:

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

Dónde *nombre* es el nombre del archivo del helper, sin la extensión .php o la parte "helper".

Por ejemplo, para cargar el archivo Helper de URL, que es nombrado *url_helper.php*, haría así:

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

Un helper puede ser cargado en cualquier lugar dentro de su función controlador (o incluso dentro de sus archivos Vista, aunque no es una buena práctica), siempre que lo cargue antes de usarlo. Puede cargar sus helpers en el constructor de su controlador, para que esté disponible automáticamente en cualquier función, o puede cargarlo en una función específica que lo necesite.

Nota: La función de carga de Helper no devuelve un valor, así que no intente asignarselo a una variable. Simplemente úselo como se muestra.

**Cargando Múltiples Helpers**

Si necesita cargar más de un helper puede especificarlos en un arreglo, así:

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

**Auto-cargando Helpers**

Si descubre que necesita un helper particular globalmente a lo largo de su aplicación, puede decirle a CodeIgniter que lo auto-cargue durante la inicialización del sistema. Esto se hace abriendo el archivo *application/config/autoload.php* y agregando el helper al arreglo autoload.

**Usando un Helper**

Una vez que haya cargado el Archivo Helper conteniendo la función que tiene intención de usar, puede llamarla de la forma estándar de PHP.

Por ejemplo, para crear un hipervínculo usando la función anchor() en uno de sus archivos de vista, haría esto:

```
<?=anchor('blog/comentarios', 'Cliquee aquí');?>
```

Donde "Cliquee aquí" es el nombre del hipervínculo, y "blog/comentarios" es la URI al controlador/función al que desee dirigir.

**"Extendiendo" Helpers**

Para "extender" Helpers, creer un archivo en su carpeta application/helpers/ con nombre idéntico al Helper existente, pero con préfijo con `MY_` (este item es configurable. Vea abajo.).

Si todo lo que necesita hacer es agregar alguna funcionalidad a un helper existente - tal vez agregar una función o dos, o cambiar como una función helper en particular opera - entonces es exagerado reemplazar el helper entero con su versión. En este caso, es mejor simlpemente "extender" el Helper. El término "extender" es usado como aproximación ya que las funciones Helpers son procedimientos discretos y no pueden ser extendidos en el sentido tradicional de programación. Bajo este velo, se da la habilidad para agregar funciones a los Helpers proveidos, o modificar como una función de Helper operante.

Por ejemplo, para extender el nativo `Helper de Arreglo` creará un archivo llamado application/helpers/`MY_array_helper.php`, y agrega o sobreescribe funciones:

```
// cualquiera_en_arreglo() no es en el Helper de Arreglo, así que
define una nueva función
function cualquiera_en_arreglo($aguja, $pajar)
{
    $aguja = (is_array($aguja)) ? $aguja : array($aguja);

    foreach ($aguja as $item)
    {
        if (in_array($item, $pajar))
        {
            return TRUE;
        }
        }

    return FALSE;
}

// random_element() está incluido en el Helper de Arreglo, así que
sobreescribe la función nativa
function random_element($arreglo)
{
    shuffle($arreglo);
    return array_pop($arreglo);
}
```

**Estableciendo Su Propio Prefijo**

El prefijo del nombre de archivo "extendiendo" Helpers es el mismo usado para extender librerías y clases de núcleo. Para establecer su propio prefijo, abra su archivo application/config/config.php y busque este item:

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

Por favor note que todas las librerías nativas de CodeIgniter son prefijadas con `CI_` así que NO use esto como prefijo.

**Usar Librerías de CodeIgniter**

Todas las librerías disponibles están ubicadas en su carpeta system/libraries. En la mayoría de los casos, para usar uno de esas clases involucra inicializarla dentro de un controlador usando la siguiente función de inicialización:

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

Donde *nombre de clase* es el nombre de la clase que quiere invocar. Por ejemplo, para cargar la clase de validación de formularios haría esto:

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

Una vez inicializada puede usarlo como se indica en la página de la guía del usuario correspondiente a esa clase.

Además, se pueden cargar varias librerías al mismo tiempo al pasar un array de librerías a la función de carga.

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

**Crear Librerías**

Cuando usamos el termino "Librerías" normalmente nos referimos a las clases que se localizan en el directorio `libraries` y descriptas en la Referencia de Clases de su Guia de Usuario. En este caso, sin embargo, en lugar de ello describiremos como puede crear sus propias librerias dentro del directorio application/libraries con el fin de mantener la separación entre sus recursos locales y los recursos del marco de trabajo global.

Como un agregado extra, CodeIgniter permite `extender` sus clases nativas a sus librerias si simplemente necesita agregar alguna funcionalidad a una libreria existente. O puede incluso sustituir a las librerias nativas colocando nombres idénticos de versiones en su carpeta application/libraries.

En Resumen:

- Puede crear librerias totalmente nuevas.
- Puede extender librerias nativas.
- Puede reemplazar librerias nativas.

La siguiente página explica estos tres conceptos en detalle.

**Nota:** Las clases de Base de Datos no pueden ser extendidas o reemplazadas con sus propias clases. Todas las otras clases estan habilitadas para ser reemplazadas/extendidas.

**Almacenamiento**

Su libreria de clases debe almacenarce dentro de su carpeta application/libraries, pues es allí donde CodeIgniter las buscará cuando sean inicializadas.

**Convenciones de Nombre**

- Los nombres de archivos deben ser capitalizados. Por ejemplo: Myclass.php
- Las declaraciones de clases deben ser capitalizadas. Por ejemplo: `class Myclass`
- Los nombre de las clases y los nombres del archivo deben coincidir.

**El Archivo de Clase**

Las clases deben tener este prototipo básico (Nota: Estamos utilizando el nombre Someclass puramente como un ejemplo):

```
<?php if (!defined('BASEPATH')) exit('No permitir el acceso directo
al script');

class Someclass {

    function some_function()
    {
    }
}

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

**Usando Su Clase**

Desde cualquiera de sus funciones de Controller puede inicializar su clases utilizando el estándar:

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

Donde *someclass* es el nombre del archivo, sin la extensión ".php" del archivo. Puede enviar el nombre del archivo en mayúscula o minúscula. A CodeIgniter no le importa.

Una vez cargado puede acceder a su clase utilizando la versión en `minúsculas`:

```
$this->someclass->some_function();  // Las instancias de objetos
serán siempre en minúsculas
```

**Pasando Parámetros Cuando Inicializa Su Clase**

En la función que carga la libreria puede pasar dinámicamente datos a travéz del segundo parámetro y ellos serán pasados a su constructor de clase:

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

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

Si utiliza esta función debe configurar su constructor de la clase para esperar los datos:

```
<?php if (!defined('BASEPATH')) exit('No permitir el acceso directo
al script');

class Someclass {

    function Someclass($params)
    {
        // Hacer algo con $params
    }
}

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

También puede pasar parámetros almacenados en un archivo de configuración. Simplemente cree un archivo de configuración que se llame igual al `nombre del archivo` de la clase y almacénelo en su carpeta application/config/. Tenga en cuenta que si pasa el dinámicamente los parámetros descritos anteriormente, la opción en el archivo de configuración no estará disponible.

**Utilizando los Recursos de CodeIgniter dentro de su Libreria**

Para acceder a los recursos nativos de CodeIgniter dentro de su libreria use la función `get_instance()`. Esta función retorna el objeto super de CodeIgniter.

Normalmente desde sus funciones del controlador llamará a cualquiera de las funciones habilitadas en CodeIgniter usando el constructor `$this`:

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

`$this`, sin embargo, sólo trabaja directamente dentro de sus controladores, sus modelos, o sus vistas. Si desea utilizar las clases de CodeIgniter dentro de sus propias clases puede hacerlo de la siguiente manera:

Primero, asigne el objeto de CodeIgniter a una variable:

```
$CI =& get_instance();
```

Una vez que se ha asignado el objeto a una variable, va a utilizar esa variable *en lugar* de `$this`:

```
$CI =& get_instance();

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

**Nota:** Se dará cuenta de que la anterior función get_instance() esta siendo pasada por referencia:

*$CI =& get_instance();*

`Esto es muy importante.` La asignación por referencia le permite utilizar el objeto original de CodeIgniter en lugar de crear una copia del mismo.

**Reemplazando Librerías Nativas con Sus Versiones**

Simplemente por asignar un nombre a sus archivos de clase idéntico a una libreria nativa causará que CodeIgniter los utilice en lugar de los nativos. Para usar esta característica debe nombrar el archivo y la declaración de la clase exactamente como la libreria nativa. Por ejemplo, para reemplazar la libreria nativa `Email` deberá crear un archivo llamado application/libraries/Email.php, y declarar su clase con:

```
class CI_Email {

}
```

Tenga en cuenta que la mayoría de las clases son con prefijo `CI_`.

Para cargar su libreria puede ver la función de carga estándar:

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

**Nota:** En este momento las clases de base de datos no puede ser reemplazadas con sus propias versiones.

### Extendiendo las Librerias Nativas

Si todo lo que necesita hacer es añadir alguna funcionalidad a una libreria existente - quizás añadir una función o dos - entonces es excesivo sustituir toda la libreria con su versión. En este caso es mejor simplemente extender la clase. La extensión de una clase es casi igual que la sustitución de una clase con un par de excepciones:

- La declaración de la clase debe extender la clase padre.
- Su nuevo nombre de la clase y el nombre del archivo se debe prefijar con `MY_` (Este tema es configurable. Véase a continuación.).

Por ejemplo, para extender la clase nativa `Email` deberá crear un archivo llamado application/libraries/`MY_Email.php`, y declarar su clase con:

```
class MY_Email extends CI_Email {

}
```

Nota: Si necesita utilizar un constructor en su clase, asegúrese de extender el padre constructor:

```
class MY_Email extends CI_Email {

    function My_Email()
    {
        parent::CI_Email();
    }
}
```

### Cargando Su Sub-Clase

Para cargar su sub-clase debe usar la sintaxis estándar que normalmente se utiliza. NO incluya su prefijo. Por ejemplo, para cargar el ejemplo anterior, que extiende la clase Email, se utiliza:

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

Una vez cargado se utiliza la variable de clase como siempre lo ha hecho para la clase de la que se extiende. En el caso de la clase Email utilizaremos todas las llamadas:

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

### Estableciendo Su Propio Prefijo

Para establecer su propio prefijo de sub-clase, abra su application/config/config.php y busque este tema:

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

Tenga en cuenta que todas las librerias nativas de CodeIgniter están prefijadas con `CI_` NO USE esto como su prefijo.

**Usar Drivers de CodeIgniter**

Los Drivers son un tipo especial de Biblioteca que tienen una clase padre y cualquier cantidad clases hijas potenciales. Las clases hijas tienen acceso a la clase padre, pero no a sus hermanas. Los drivers proveen una sintaxis elegante a sus controladores para bibliotecas que se benefician o necesitan dividirse en clases discretas.

Los Drivers están ubicados en la carpeta system/libraries, en su propia carpeta que se llama igual que la clase de la biblioteca padre. También dentro de esa carpeta hay una subcarpeta llamada `drivers`, que contiene todos los posibles archivos de clases hijas.

Para usar un driver, lo inicializará dentro de un controlador usando la siguiente función de inicialización:

```
$this->load->driver('nombre de clase');
```

Donde *nombre de clase* es el nombre de la clase driver que desea invocar. Por ejemplo, para cargar un driver llamado "algun_padre" debería hacer esto:

```
$this->load->driver('algun_padre');
```

Los métodos de las clases se pueden invocar con:

```
$this->some_parent->algun_metodo();
```

Las clases hijas y los drivers en sí mismos, se pueden llamar directamente a través de la clase padre, sin inicializarlos:

```
$this->algun_padre->hija_uno->algun_metodo();
$this->algun_padre->hija_dos->otro_metodo();
```

## Crear Drivers

**Directorio del Driver y Estructura de Archivos**

Muestra del directorio del driver y diseño de la estructura de archivos:

- /application/libraries/Nombre_de_driver
    - Nombre_de_driver.php
    - drivers
        - Driver_name_subclase_1.php
        - Driver_name_subclase_2.php
        - Driver_name_subclase_3.php

**Nota:** Para mantener la compatibilidad en sistemas de archivo sensibles a la mayúscula, aplicarle *ucfirst()* al directorio `Nombre_de_driver`.

### Creación de Clases de Sistema de Núcleo

Cada vez que CodeIgniter corre hay varias clases base que son inicializados automáticamente como parte del núcleo del entorno de trabajo. Es posible, sin embargo, intercambiar cualquiera de las clases de sistema de núcleo con sus propias versiones o incluso extender las versiones de núcleo.

**La mayoría de los usuarios nunca necesitará hacer nada de esto, pero la opción de reemplazar o extenderla existe para quienes qioerem significantemente alterar el núcleo de CodeIgniter.**

**Nota:** Meterse con una clase de sistema de núcleo tiene varias implicaciones, así que esté seguro que sabe lo que está haciendo antes de intentarlo.

### Lista de Clases de Sistema

La siguiente es una lista de archivos de sistema de núcleo que son invocados cada vez que CodeIgniter corre:

- Benchmark
- Config
- Controller
- Exceptions
- Hooks
- Input
- Language
- Loader
- Log
- Output
- Router
- URI
- Utf8

### Reemplazar las Clases de Núcelo

Para usar una de sus propias clases de sistema en vez de una de las predeterminadas simplemente ubique su versión dentro de su directorio local application/libraries:

`application/libraries/`alguna-clase.php

Si el directorio no existe, puedes crearlo.

Cualquier archivo llamado idénticamente a uno de la lista de arriba será usado en vez del normalmente usado

Por favor note que su clase debe usar `CI` como prefijo. Por ejemplo, si su archivo es llamado`Input.php` la clase será llamada:

```
class CI_Input {

}
```

**Extendiendo las Clases de Núcleo**

Si todo lo que necesita hacer es agregar alguna funcionalidad a una librería existente - quizás agregar una función o dos - entonces es exagerado reemplazar la librería entera con su versión. En este caso es mejor simplemente extender la clase. Extender la clase es casi idéntico a reemplazar la clase con un par de exceptiones:

- La declaración de clase debe extender la clase padre.
- Su nuevo nombre de clase y de archivo debe ser prefijado con `MY_` (este item es configurable. Vea abajo.).

Por ejemplo, para extender la clase nativa `Input` creará un archivo llamado dfn>application/libraries/`MY_Input.php`, y declara su clase con:

```
class MY_Input extends CI_Input {

}
```

Nota: Si necesita usar un constructor en su clase, asegurese que extiende al constructor del padre:

```
class MY_Input extends CI_Input {

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

**Consejo:** Cualquier función en su clase que es nombrada idénticamente a las funciones en la clase padre serán usadas en vez de las nativas (esto es conocido como "sobreescritura de método"). Esto le permite alterar sustancialmente el núcleo de CodeIgniter.

Si está extendiendo la clase Controller del núcleo, entonces asegúrese de extender su nueva clase en los contructores de sus controladores de la aplicación.

```
class Welcome extends MY_Controller {

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

    function index()
    {
        $this->load->view('welcome_message');
    }
}
```

**Estableciendo Su Propio Prefijo**

Para establecer su propio prefijo de sub-clase, abra su archivo application/config/config.php y busque este item:

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

Por favor note que todas las librerías nativas de CodeIgniter están prefijadas con `CI_` para NO usar ese prefijo.

**Auto-Carga de Recursos**

CodeIgniter viene con una característica de "Auto-carga" que permite a librerías, asistentes(helpers), y complementos ser inicializados automáticamente cada vez que el sistema arranque. Si necesita ciertos recursos a nivel global a lo largo de su aplicación, debe considerar la posibilidad de Auto-carga para su conveniencia.

Los siguientes elementos pueden ser cargados automáticamente:

- Clases Core encontradas en la carpeta "libraries"
- Helpers encontrados en la carpeta "helpers"
- Archivos de configuración encontrados en la carpeta "config"
- Archivos de idioma encontrados en la carpeta "system/language"
- Modelos encontrados en la carpeta "models"

Para autocargar recursos, abra el archivo *application/config/autoload.php* y agregue el elemento que desea cargar al arreglo `autoload`. Encontrará las instrucciones en el archivo correspondiente a cada uno de los tipos de elementos.

**Nota:** No incluya la extensión del archivo (.php) cuando agregue elementos al arreglo de autocargado.

**Funciones Comunes**

CodeIgniter usa unas pocas funciones para su operación que se definen globalmente y están disponibles en cualquier lugar. Estas funciones no requieren cargar ninguna biblioteca o helper.

**is_php('*numero_de_version*')**

is_php() determina si la versión de PHP usada es mayor o igual que el *numero_de_version* suministrado.

```
if (is_php('5.3.0'))
{
    $str = quoted_printable_encode($str);
}
```

Devuelve el booleano `TRUE` si la versión instalada de PHP es mayor o igual que el número de versión suministrado. Devuelve `FALSE` si la versión instalada de PHP es menor que el número de versión suministrado.

**is_really_writable('*ruta/al/archivo*')**

is_writable() devuelve `TRUE` en servidores Windows cuando realmente no se puede escribir en el archivo, ya que el sistema operativo le informa `FALSE` a PHP solamente si el atributo de solo lectura está marcado. Esta función determina si un archivo es escribible realmente al intentar escribir primero en él. Por lo general sólo se recomienda en las plataformas donde esta información puede no ser fiable.

```
if (is_really_writable('file.txt')) {
echo "Podría escribir lo que quiera";
}
else
{
echo "No se puede escribir en el archivo";
}
```

**config_item('*clave_de_item*')**

La forma preferida para acceder a la información de configuración es la Librería Config. Sin embargo, config_item() se puede usar para recuperar claves simples. Para más información, lea la documentación de la Librería Config.

**show_error('*mensaje*'), show_404('*pagina*'), log_message('*nivel*', '`mensaje`')**

Cada uno de estos se describe en la página Manejo de Errores.

**set_status_header(*codigo*, '*texto*')**

Le permite establecer manualmente el encabezado de estado del servidor. Ejemplo:

```
set_status_header(401);
// Establece el encabezado como: No autorizado
```

Vea aquí la lista completa de encabezados.

**remove_invisible_characters(*$str*)**

Esta función evita la inserción de caracteres nulos entre caracteres ASCII como Java\0script.

**html_escape(*$mixed*)**

Esta función proporciona un acces directo a la función htmlspecialchars(). Acepta cadenas y arrays. Es muy útil para evitar Cross Site Scripting (XSS).

**Ruteo URI**

Tipicamente hay una relacion uno-a-uno entre una cadena URL y su correspondiente controlador clase/metodo. Los segmentos en una URI normalmente siguen este patron:

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

En algunas instancias, sin embargo, querras remapear esta relación para que una clase/funcion diferente pueda ser llamada en lugar de la correspondiente a la URL.

Por ejemplo, digamos que quieres que tus URLs tengan este prototipo:

www.tu-sitio.com/product/1/
www.tu-sitio.com/product/2/
www.tu-sitio.com/product/3/
www.tu-sitio.com/product/4/

Normalmente el segundo segmento de la URL es reservada para el nombre de funcion, pero en este ejemplo en lugar de eso tiene el ID del producto. Para superar esto, CodeIgniter te permite remapear el manejador URI.

**Creando tus propias reglas de ruteo**

Las reglas de ruteo son definidas dentro de tu archivo *application/config/routes.php*. Dentro de el deberias poder ver un arreglo llamado $route que te permite especificar tus criterios de ruteo. Las rutas podrian ser especificadas utilizando comodines o Expresiones Regulares

**Comodines**

Un tipico comodidin de ruteo deberia parecerse a esto:

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

En una ruta, la clave del arreglo contiene la URI que debe coincidir, mientras que el valor del arreglo contiene el destino a donde debe ser re-ruteado. En el ejemplo anterior, si el literalmente la palabra "product" es encontrada en el primer segmento de la URL, y el numero es encontrado en el segundo segmento, la clase "catalog" y el metodo "product_lookup" seran usados.

Puedes hacer que coincidan los valores literales, o puedes usar dos tipos de comodines:

**(:num)** coincidira con un segmento que unicamente contiene numeros.
**(:any)** coincidira con un segmento que contenga cualquier caracter.

**Nota:**Las Rutas correran en el orden que sean definidas. Rutas superiores siempre tomaran precedencia sobre las que tienen menos precedencia.

**Ejemplos**

He aqui algunos ejemplos de ruteo:

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

Una URL conteniendo la palabra "journals" en el primer segmento sera remapeada a la clase "blogs".

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

una URL conteniendo el segmento blog/joe sera remapead a la clase "blogs" y el metodo "users". El ID

sera seteado a "34".

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

Una URL con "product" como el primer segmento, y cualquier cosa en el segundo sera remapeado a la clase "catalog" y el metodo "product_lookup".

```
$route['producto/(:num)'] = "catalogo/producto_buscar_por_id/$1";
```

Una URL con "producto" como primer segmento y un número en el segundo, se remapeará a la clase "catalogo" y al método "producto_buscar_por_id" pasándole una variable a la función.

**Importante:** No use barras al comienzo/final.

**Expresiones Regulares**

Si lo prefieres, puedes utilizar expresiones regulares para definir tus reglas de ruteo. Cualquier expresion regular es permitida, como lo son las back-references.

**Note:** Si usa back-references debes usar la sintaxis de dolares antes que la sintaxis de doble barra diagonal.

Una tipica Expresion Regular deberia parecerse a esto:

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

En el ejemplo anterior, una URI similar a products/shirts/123 llamara en su lugar a la clase controlador shirts y la funcion id_123.

Tambien puedes mesclar comodines con expresiones regulares.

**Rutas Reservadas**

Hay dos rutas reservadas:

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

Esta ruta indica cual clase controlador debera ser cargado si la URI no contiene datos, caso que ocurrira cuando la gente cargue la URL raiz. En el ejemplo anterior, la clase "welcome" seria la que se cargue. Te animamos a que siempre tengas una ruta por defecto, caso contrario una pagina 404 apareceria por defecto.

```
$route['404_override'] = '';
```

Esta ruta indica cual clase controlador debera ser cargada si el controlador solicitado no se encuentra. Se anulará la página de error 404 por defecto. No afectará a la función `show_404()`, que seguirá cargando el archivo por defecto error_404.php en *application/errors/error_404.php*.

**Importante:** Las rutas reservadas deben aparecer antes que cualquier comodin o expresion regular de ruteo.

**Manejo de Errores**

CodeIgniter le permite crear un reporte de errores en sus aplicaciones usando las funciones descritas abajo. Además, tiene una clase de historial de errores que le permite que los mensajes de error y depuración sean guardados en archivos de texto.

**Nota:** Por defecto, CodeIgniter muestra todos los errores PHP. Puede querer cambiar este comportamiento una vez que su desarrollo esté completo. Encontrará la función error_reporting() ubicada al principio de su archivo index.php principal. Deshabilitar el reporte de errores NO prevendrá que el archivo de historial sea escrito si hay errores.

A diferencia de la mayoría de los sistemas en CodeIgniter, las funciones de error son interfaces de simple procedimiento que están disponibles globalmente a lo largo de la aplicación. Este aproximamiento le permite que los mensajes de error sean activados sin tener que preocuparse de el ámbito de la clase/función.

La siguientes funciones le permiten generar errores:

**show_error('*mensaje*')**

Esta función mostrará el mensaje de error suministrado usando la siguiente plantilla de error:

application/errors/`error_general.php`

**show_404('*pagina*')**

Esta función mostrará el mensaje de error 404 suministrado usando la siguiente plantilla de error:

application/errors/`error_404.php`

La función espera que la cadena pasada sea la ruta del archivo a la página que no se encontró. Note que CodeIgniter automáticamente mostrará mensajes 404 si los controladores no serán encontrados.

CodeIgniter registra automáticamente cualquier llamada a show_404(). El registro se detiene cuando se establece el segundo parámetro opcional a `FALSE`.

**log_message('*nivel*', '`mensaje`')**

Esta función le permitirá escribir mensajes de error en sus archivos de historial. Debe suministrar uno de tres "niveles" en el primer parámetro, indicando que tipo de mensaje es (depuración, error, info), con el mensaje mismo en el segundo parámetro. Ejemplo:

```
if ($alguna_variable == "")
{
    log_message('error', 'Alguna variable no contenía un valor.');
}
else
{
    log_message('debug', 'Alguna variable era correctamente
establecida');
}

log_message('info', 'El proposito de alguna variabl es proveer algún
```

```
valor.');
```

Hay tres tipos de mensajes:

1. Mensajes de Error. Estos son mensajes de error, tal como errores de PHP o del usuario.
2. Mensajes de Depuración. Estos son mensajes que le asisten al depurar. Por ejemplo, si una clase ha sido inicializadam podría guardar en el historial esta información de depuración.
3. Mensajes Informacionales. Estos son los mensajes de menor prioridad, simplemente dan información acerca de algún proceso. CodeIgniter no genera nativamente ningún mensaje de informaciójn pero puede quererlo en su aplicación.

**Nota:** Para que el archivo de historial sea realmente escriturable, la carpeta "logs" debe ser escriturable. Además debe establecer el "threshold" del historial. Puede, por ejemplo, sólo querer que los mensajes de errore sean guardados, y no los otros dos tipos. Si lo establece a cero el historial será deshabilitado.

**Almacenamiento en Cache de Páginas Webs**

CodeIgniter le permite hacer caché de sus páginas con el fin de lograr el máximo rendimiento.

Aunque CodeIgniter es bastante rápido, la cantidad de información dinámica que se muestren en sus páginas se correlaciona directamente a los recursos del servidor, la memoria y los ciclos de procesamiento utilizados, que afectan a su velocidad de carga de páginas. Por cachear sus páginas, ya que se guardan en su estado plenamente renderizadas, puede alcanzar el rendimiento que se acerca a la de las páginas web estáticas.

**Cómo Funciona el Trabajo de Almacenar en Caché ?**

Se puede habilitar el almacenamiento en caché para cada página, y puede establecer el tiempo que debe permanecer una página en caché antes de ser refrescada. Cuando una página se carga por primera vez, el archivo de caché se escribirá en su carpeta system/cache. En posteriores cargas de la página el archivo de caché se recibirá y se enviará a la solicitud del navegador del usuario. Si ha caducado, será eliminado y actualizado antes de ser enviado al navegador.

Nota: La etiqueta Benchmark no fue cacheada para que pueda ver la velocidad de carga de páginas cuando está permitido el almacenamiento en caché.

**Habilitar el Almacenamiento en Caché**

Para habilitar el almacenamiento en caché, poner la siguiente etiqueta en cualquiera de sus funciones de controlador:

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

Donde *n* es el número de **minutos** que desea que la página permanezca en caché entre refrescos.

La etiqueta puede ir a cualquier parte dentro de una función. No se ve afectada por la orden en la que aparece, de modo que la puede poner en el lugar donde le parezca mas lógico para usted. Una vez que la etiqueta esté en su lugar, sus páginas comenzarán a ser cacheadas.

**Arvertencia:** Debido a la forma en CodeIgniter almacena el contenido de la producción, el almacenamiento en caché sólo funcionará si está generando vistas para su controlador con una vista.

**Nota:** Antes de que los archivos de caché puedan ser escritos, debe configurar los permisos de archivo en su carpeta system/cache de tal modo que se pueda grabar.

**Borrar Caches**

Si ya no desea un archivo de caché, puede quitar la etiqueta de cacheo y ya no será refrescado cuando expire. Nota: La eliminación de la etiqueta no implica la supresión de la memoria caché inmediatamente. Tendrá que expirar normalmente. Si quiere eliminar lo anterior tendrá que eliminarla manualmente de la carpeta de caché.

### Sintaxis Alternativa PHP para Archivos de Vista

En caso de no utilizar el motor de plantilla de CodeIgniter, estará usando PHP puro en su archivos de Vista. Para minimizar el código PHP en estos archivos, y para que sea más fácil identificar los bloques de código se recomienda que utilice sintáxis alternativa PHP para las estructuras de control y etiquetas cortas para las declaraciones "echo". Si no está familiarizado con esta sintaxis, ésta le permite eliminar las llaves de su código, y eliminar las declaraciones "echo".

### Soporte Automático para Etiquetas Cortas

**Nota:** Si encuentra que la sintáxis que se describe en esta página no funciona en su servidor podría ser que las "etiquetas cortas" esten desactivadas en su archivo ini de PHP. CodeIgniter, opcionalmente, reescribirá etiquetas cortas sobre la marcha, lo cual le permite utilizar la sintáxis, incluso si su servidor no lo soporta. Esta característica puede ser habilitada en su archivo config/config.php.

Tenga en cuenta que si hace uso de esta función, si encuentra errores de PHP en sus **archivos de vista**, el mensaje de error y el número de línea no se muestran correctamente. En lugar de ello, todos los errores que se mostrará como errores `eval()`.

### Alternativas Echos

Normalmente para "echo", o imprimir una variable se haría esto:

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

Con la sintáxis alternativa lo puede hacer de esta manera:

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

### Estructuras de Control Alternativas

Las estructuras de control, como *if*, *for*, *foreach*, y *while* pueden ser escritas en un formato más simple también. Aquí tiene un ejemplo de uso de foreach:

```
<ul>

<?php foreach($todo as $item): ?>

<li><?=$item?></li>

<?php endforeach; ?>

</ul>
```

Note que no hay llaves. En lugar de ello, la llave que cierra es reemplazada con un *endforeach*. Cada una de las estructuras de control mencionadas anteriormente tiene una sintaxis similar de cierre: *endif*, *endfor*, *endforeach*, y *endwhile*

Observe también que en lugar de utilizar un punto y coma después de cada estructura (excepto la última), hay dos puntos. Esto es importante!

Aquí hay otro ejemplo, usando if/elseif/else. Nótese los dos puntos:

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

```
    <h3>Hola Sally</h3>
```

```
<?php elseif ($username == 'joe'): ?>
```

```
    <h3>Hola Joe</h3>
```

```
<?php else: ?>
```

```
    <h3>Hola usuario desconocido</h3>
```

```
<?php endif; ?>
```

## Seguridad

Aquí se describen algunas "buenas prácticas" acerca de segurdad web, y detalles de características de seguridad interna de CodeIgniter.

## Seguridad en URI

CodeIgniter es bastante restrictivo sobre que caracteres permitir en las cadenas URI para ayudar a minimizar la posibilidad de que datos maliciosos puedan ser pasados a su aplicación. Las URIs sólo pueden contener lo siguiente:

- Texto alfanumérico
- "Tilde": ~
- Punto: .
- Dos puntos: :
- Guion bajo: _
- Guion: -

## Datos GET, POST, y COOKIE

Los datos GET son simplemente anulados por CodeIgniter ya que el sistema utiliza segmentos URI en vez de las tradicionales query strings de URL (a menos que la opción query string esté habilitada en su archivo config). El arreglo global GET es **destruido** por la clase Input durante la inicialización del sistema.

## Register_globals

Durante la inicialización del sistema, todas las variables globales son destruidas, excepto aquellas encontradas en los arreglos *$_POST* y *$_COOKIE*. La rutina de eliminación es efectivamente lo misma que `register_globals = off`.

## error_reporting

En entornos de producción, es normalmente deseable deshabilitar el reporte de errores de PHP estableciendo la bandera interna error_reporting al valor 0. Esto deshabilita los errores nativos de PHP evitando que se presenten por pantalla, los que pueden contener información sensible.

Establecer la constante `ENVIRONMENT` de CodeIgniter en el archivo index.php al valor '`production`', desconectará esos errores. En el modo de desarrollo, se recomienda que se use el valor '`development`'. Se puede encontrar más información acerca de la diferencia entre entornos en la página Manejo de Múltiples Entornos.

## magic_quotes_runtime

La directiva magic_quotes_runtime es apagada durante la inicialización del sistema para que no tenga que remover las barras cuando se recuperen datos de la base de datos.

## Buenas prácticas

Antes de aceptar cualquier dato en su aplicación, ya sean datos POST desde el envío de un formulario,

datos COOKIE, datos URI, datos XML-RPC, o incluso datos desde el arreglo SERVER, está alentado a practicar estos tres pasos de acercamiento:

1. Filtrar los datos como si fueran contaminados.
2. Validar los datos para asegurarse que conforman el tipo correcto, largo, tamaño, etc. (a veces, este paso puede reemplazar al paso uno)
3. Escapar los datos entes de enviarlo a su base de datos.

CodeIgniter provee las siguientes funciones para asistirlo en este proceso

- **Filtro de XSS**

  CodeIgniter viene con un filtro de XSS (Cross Site Scripting). Este filtro busca técnicas comunmente usadas para embeber Javascript malicioso a sus datos, u otro tipo de código que intente "secuestrar" (hijack) cookies o hacer otra cosa maliciosa. El Filtro XSS es descripto aquí.

- **Validar los datos**

  CodeIgniter tiene una Clase de Validación que le asiste en la validación, filtro y preparación de datos.

- **Escapar todos los datos antes de insertarlo a la base de datos**

  Nunca inserte información a su base de datos sin escaparla.

**Estilo y Sintaxis Generales**

Se describe el uso de reglas de codificación usadas al desarrollar CodeIgniter.

### Formato de Archivo

Los archivos deberían guardarse con codificación Unicode (UTF-8). *No* debería usarse el BOM. A diferencia de UTF-16 y UTF-32, no hay un bit de orden para indicar en un archivo codificado con UTF-8, y el BOM puede tener efectos colaterales negativos en PHP en el envío de la salida, evitando que la aplicación sea capaz de establecer sus encabezados. Se deberían usar las terminaciones de línea de Unix (LF).

Esta es la forma de aplicar esas configuraciones en los editores de texto más comunes. Las instrucciones para su editor de texto pueden variar; lea la documentación de su editor.

**TextMate**

1. Abra las Preferencias de la Aplicación
2. Hacer clic en Avanzado, y luego en la solapa "Guardar"
3. En "Codificación de Archivo", seleccione "UTF-8 (recomendado)"
4. En "Terminación de Línea", seleccione "LF (recomendado)"
5. *Opcional:* Marque "También usar para los archivos existentes" si desea modificar las terminaciones de línea de archivos que abre para las nuevas preferencias.

**BBEdit**

1. Abra las Preferencias de la Aplicación
2. Seleccione "Codificaciones del Texto" en la izquierda.
3. En "Codificación del texto para nuevos documentos", seleccione "Unicode (UTF-8, sin BOM)"
4. *Opcional:* En "Si no se puede determinar la codificación del archivo, usar", seleccione "Unicode (UTF-8, sin BOM)"
5. Selecciones "Archivos de Texto" en la izquierda.
6. En "Saltos de Línea por Defecto", seleccione "Mac OS X y Unix (LF)"

**Etiqueta de Cierre de PHP**

La etiqueta de cierre de PHP en un documento PHP **?>** es opcional para el analizador de PHP. Sin embargo, si se usa, cualquier espacios en blanco que siga a la etiqueta de cierre, sea introducida por el desarrollador, el usuario o una aplicación FTP, puede causar una salida no deseada, errores de PHP, o si la última se suprime, páginas en blanco. Por esta razón, todos los archivos de PHP deberían **OMITIR** la etiqueta de cierre de PHP, y en su lugar usar un bloque de comentario para marcar el fin del archivo y su ubicación relativa a la raíz de la aplicación. Esto le permite aún identificar al archivo como completo y no trunco.

**INCORRECTO:**
```
<?php echo "Este es mi código!"; ?>
```
**CORRECTO:**
```
<?php echo "Este es mi código!"; /* Fin del archivo mi_archivo.php */ /* Ubicación: ./system/modules/mi_modulo/mi_archivo.php */
```

**Nomenclatura de Clases y Métodos**

Los nombres de clases siempre deberían comenzar con una letra mayúscula. Varias palabras se deberían separar con un guión de subrayado y no usar CamelCase. Todos los otros métodos de clase se deberían escribir completamente en minúsculas y su nombre debería indicar claramente su función, incluyendo preferiblemente un verbo. Trate de evitar los nombres demasiado largos y detallados.

**INCORRECTO:** `class superclass class SuperClass` **CORRECTO:** `class Super_class class Super_class { function __construct() { } }`

Ejemplos de una nomenclatura de métodos adecuada e inadecuada:

**INCORRECTO:** `function fileproperties() // no es descriptivo y necesita un guión de subrayado de separación function fileProperties() // no es descriptivo y usa CamelCase function getfileproperties() // Mejor! Pero todavía le falta el guión de subrayado de separación function getFileProperties() // usa CamelCase function get_the_file_properties_from_the_file() // demasiada palabrería` **CORRECTO:** `function get_file_properties() // descriptivo, guión de subrayado de separación y todas la letras son minúsculas`

**Nombres de Variable**

La directriz para el nombramiento de variables es muy similar al usado para métodos de clase. Concretamente, las variables deberían contener solamente letras minúsculas, usar guiones de subrayado como separadores y tener un nombre que razonablemente indique su propósito y contendo. Variables de nombre muy corto o sin palabras se deberían usar solamente como iteradores en ciclos for().

**INCORRECTO:** `$j = 'foo'; //las variables de una sola letra se deberían usar solamente en ciclos for() $Str // contiene letras mayúsculas $bufferedText // usas CamelCase y podría acortarse sin perder sentido semántico $groupid // varias palabras, necesita un separador de guión de subrayado $name_of_last_city_used // demasiado largo` **CORRECTO:** `for ($j = 0; $j < 10; $j++) $str $buffer $group_id $last_city`

**Comentarios**

En general, el código debe ser comentado de forma prolífica. No sólo ayuda a describir el flujo y la intención del código para los programadores con menos experiencia, sino que puede resultar muy valiosa al regresar a su propio código meses después en línea. No hay un formato establecido para comentarios, pero se recomienda lo siguiente.

Estilo de comentarios DocBlock que precede declaraciones de clases y métodos, que puede ser levantado por los IDEs:

```
/** * Super Class * * @package Nombre del paquete * @subpackage Subpaquete * @category Categoría * @author Nombre del autor * @link http://ejemplo.com */ class Super_class { /** * Codifica una cadena para usarla en XML * * @access public * @param string * @return string */ function xml_encode($str)
```

UUsar comentarios en línea simple dentro del código, dejando una línea en blanco entre un bloque

largo de comentarios y el código.

```
// rompe las cadenas mediante caracteres de nueva línea $parts =
explode("\n", $str); // Un comentario más grande de lo que necesita
para dar un gran detalle de lo que // está ocurriendo y porque puede
usar varios comentarios de línea simple. Trate // de mantener un
ancho razonable, alrededor de 70 caracteres es más fácil para //
leer. No dude en vincular recursos externos que pueden proveer
grandes // detalles: // //
http://ejemplo.com/informacion_acerca_de_algo/en_particular/ $parts =
$this->foo($parts);
```

**Constantes**

Las constantes siguen las mismas directrices que las variables, excepto que las constantes siempre deberían escribirse completamente en mayúsculas. *Siempre usar constantes de CodeIgniter cuando sea adecuado, por ejemplo, SLASH, LD, RD, PATH_CACHE, etc.*

**INCORRECTO:** `myConstant // missing underscore separator and not fully uppercase N // no single-letter constants S_C_VER // not descriptive $str = str_replace('{foo}', 'bar', $str); // should use LD and RD constants` **CORRECTO:** `MY_CONSTANT NEWLINE SUPER_CLASS_VERSION $str = str_replace(LD.'foo'.RD, 'bar', $str);`

**TRUE, FALSE y NULL**

Las palabras clave **TRUE**, **FALSE** y **NULL** siempre deberían escribirse completamente en mayúsculas.

**INCORRECTO:** `if ($foo == true) $bar = false; function foo($bar = null)` **CORRECTO:** `if ($foo == TRUE) $bar = FALSE; function foo($bar = NULL)`

**Operadores Lógicos**

Se desaconseja el uso de **||** dado que la claridad de algunos dispositivos de salida es baja (por ejemplo, podría verse como el número 11). Es preferible **&&** en lugar de **AND** pero ambos son aceptables y un espacio siempre debería preceder y seguir a **!**.

**INCORRECTO:** `if ($foo || $bar) if ($foo AND $bar) // okay but not recommended for common syntax highlighting applications if (!$foo) if (! is_array($foo))` **CORRECTO:** `if ($foo OR $bar) if ($foo && $bar) // recommended if ( ! $foo) if ( ! is_array($foo))`

**Comparar Valores de Retorno y Typecasting**

Algunas funciones de PHP devuelven FALSE en caso de falla, pero también puede haber un valor de retorno válido de "" o 0, lo que se evaluaría como FALSE en comparaciones poco precisas. Sea explícito al comparar el tipo de variable al usar esos valores de retorno en condicionales para asegurar que el valor de retorno es en realidad lo que espera, y no un valor que tiene un equivalente de evaluación de tipo relajado.

Use el mismo rigor cuando en el retorno y verificación de sus propias variables. Use === y !== según sea necesario.

**INCORRECTO:** `// Si 'foo' está al inicio de la cadena, strpos devolverá un 0, // resultando esta evaluación condicional como TRUE if (strpos($str, 'foo') == FALSE)` **CORRECTO:** `if (strpos($str, 'foo') === FALSE)` **INCORRECTO:** `function build_string($str = "") { if ($str == "") // oh-oh! ¿Qué ocurre si se pasa como argumento FALSE o el entero 0? { } }` **CORRECTO:** `function build_string($str = "") { if ($str === "") { } }`

Vea también acerca del typecasting, lo que puede ser muy útil. El typecasting tiene un efecto ligeramente distinto que puede ser deseable. Al convertir una variable a una cadena, por ejemplo, las variables `NULL` y `FALSE` se convierten en cadenas vacías, 0 (y otros números) se convierten en cadenas de dígitos, y el booleano `TRUE` se convierte en "1":

```
$str = (string) $str; // trata a $str como una cadena
```

### Código de Depuración

No se puede dejar código de depuración en el lugar a menos que se lo comente, por ejemplo, ninguna llamada a var_dump(), print_r()/>, die(), o exit() usada durante la creación de un complemento.

```
// print_r($foo);
```

### Espacios en Blanco en Archivos

Los espacios en blanco no pueden preceder a la etiqueta de apertura de PHP o seguir a la etiqueta de cierre de PHP. La salida se almacena en búfer, por lo tanto los espacios en blanco en sus archivos pueden provocar que la salida comience antes que CodeIgniter imprima su contenido, conduciendo a errores y a la incapacidad de CodeIgniter de enviar los encabezados adecuados. En el siguiente ejemplo, seleccione el texto con el ratón para revelar los espacios en blanco INCORRECTOS.

**INCORRECTO**:

```
<?php // ...hay un espacio en blanco y un salto de línea antes de la etiqueta de // apertura de PHP // así como un espacio en blanco después de la etiqueta de cierre de PHP ?>
```

**CORRECTO**:

```
<?php // este ejemplo no tiene espacios en blanco antes o después de las etiquetas // de apertura y cierre de PHP ?>
```

### Compatibilidad

A menos que sea mencionado específicamente en la documentación de su complemento, todo código tiene que ser compatible con PHP versión 4.3 o superior. Además, no usar funciones de PHP que necesiten instalar de bibliotecas no estándares, a menos que su código contenga un método alternativo cuando la función no esté disponible, o implícitamente documente que su complemento necesita dichas bibliotecas de PHP.

### Nombres de Clases y Archivos usando Palabras Comunes

Cuando su clase o nombre de archivo son una palabra común, o puede ser bastante probable que nombre igual en otro script de PHP, proveer un prefijo único para ayudar a impedir esa colisión. Siempre darse cuenta que sus usuarios finales pueden ejecutar otros complementos o scripts de PHP o de terceras partes. Elija un prefijo que sea único para identificarlo como desarrollador o compañía.

**INCORRECTO:** `class Email pi.email.php class Xml ext.xml.php class Import mod.import.php` **CORRECTO:** `class Pre_email pi.pre_email.php class Pre_xml ext.pre_xml.php class Pre_import mod.pre_import.php`

### Nombres de Tablas de Base de Datos

Cualquier tabla que su complemento pueda usar tiene que tener el prefijo '`exp_`', seguido por un prefijo de unicidad que lo identifique a Ud como desarrollador o compañía, y luego un breve nombre descriptivo de tabla. No necesita preocuparse acerca del prefijo de base de datos que se usa en la instalación del usuario, ya que la clase database de CodeIgniter convertirá automáticamente '`exp_`' a lo que realmente se usa.

**INCORRECTO:** `email_addresses // faltan ambos prefijos pre_email_addresses // falta el prefijo exp_ exp_email_addresses // falta el prefijo único` **CORRECTO:** `exp_pre_email_addresses`

**Nota:** Tenga en cuenta que MySQL tiene un límite de 64 caracteres para los nombres de tablas. Esto no debería ser un problema, ya que los nombres de tablas que superan esa cantidad probablemente no tengan nombres razonables. Por ejemplo, el siguiente nombre de tabla excede esta limitación por un caracter. Tonto ¿no? **exp_pre_email_addresses_of_registered_users_in_seattle_washington**

### Un Archivo por Clase

Usar archivos separados para cada clase que su complemento use, a menos que las clases estén *estrechamente relacionadas*. Un ejemplo de archivos de CodeIgniter que contienen varias clases es el archivo de la clase `Database`, que contiene tanto a la clase `DB` como a la clase `DB_Cache`, y el complemento `Magpie`, que contiene tanto a la clase `Magpie` como a `Snoopy`.

### Espacios en Blanco

Usar tabuladores en lugar de espacios en blanco en su código. Esto puede parecer una pequeñez, pero usando tabuladores en lugar de espacios en blanco le permite al desarrollador que mira su código para tener indentación en los niveles que prefiera y personalizar en cualquier aplicación que use. Y como beneficio colateral, resulta en archivos (un poco) más compactos, almacenando un caracter de tabulador contra, digamos, cuatro caracteres de espacio.

### Saltos de Línea

Los archivos se tienen que guardar con saltos de línea de Unix. Esto es más un problema para los desarrolladores que trabajan en Windows, pero en cualquier caso, asegúrese de que su editor de texto está configurado para guardar los archivos con saltos de línea de Unix.

### Indentación de Código

Usar el estilo de indentación de Allman. Con excepción de las declaraciones de clase, las llaves siempre se ubican en línea con ellas mismas, e indentadas al mismo nivel que las sentencias de control que las "poseen".

**INCORRECTO:** `function foo($bar) { // ... } foreach ($arr as $key => $val) { // ... } if ($foo == $bar) { // ... } else { // ... } for ($i = 0; $i < 10; $i++) { for ($j = 0; $j < 10; $j++) { // ... } }`
**CORRECTO:** `function foo($bar) { // ... } foreach ($arr as $key => $val) { // ... } if ($foo == $bar) { // ... } else { // ... } for ($i = 0; $i < 10; $i++) { for ($j = 0; $j < 10; $j++) { // ... } }`

### Espaciado de Paréntesis y Llaves

En general, los paréntesis y las llaves no deberían usar espacios adicionales. La excepción es que un espacio siempre debería seguir a las estructuras de control de PHP que acepten argumentos entre paréntesis (declare, do-while, elseif, for, foreach, if, switch, while), para ayudar a distinguirlas de las funciones e incrementar la legibilidad.

In general, parenthesis and brackets should not use any additional spaces. The exception is that a space should always follow PHP control structures that accept arguments with parenthesis (declare, do-while, elseif, for, foreach, if, switch, while), to help distinguish them from functions and increase readability.
`INCORRECTO: $arr[ $foo ] = 'foo'; CORRECTO: $arr[$foo] = 'foo'; // no spaces around array keys INCORRECTO: function foo ( $bar ) { } CORRECTO: function foo($bar) // no spaces around parenthesis in function declarations { } INCORRECTO: foreach( $query->result() as $row ) CORRECTO: foreach ($query->result() as $row) // single space following PHP control structures, but not in interior parenthesis`

### Texto Localizado

Cualquier texto que se muestre en el panel de control, debería usar variables de idioma en su archivo de idioma para permitir la localización

`INCORRECTO: return "Invalid Selection"; CORRECTO: return $this->lang->line('invalid_selection');`

### Métodos Privados y Variables

Se deberían prefijar con un guión de subrayado los métodos y variables que solamente son accedidos internamente por su clase, tales como utilidades y helpers de funciones usan para abstracción del código.

`convert_text() // método público _convert_text() // método privado`

### Errores de PHP

El código tiene que ejecutar libre de errores y no depender que las advertencias y avisos estén ocultos para cumplir con este requisito. Por ejemplo, nunca acceder una variable que no estableció por si mismo (tal como claves del array $_POST), sin primero verificar si están establecidas con isset().

Asegurarse que durante el desarrollo de su complemento, el reporte de errores esté habilitado para

TODOS los usuarios y que display_errors está habilitado en el entorno de PHP. Puede verificar esto con:

```
if (ini_get('display_errors') == 1) { exit "Enabled"; }
```

En algunos servidores donde display_errors está deshabilitado, y no tiene la posibilidad de cambiar esto en el php.ini, a menudo se puede habilitar con:

```
ini_set('display_errors', 1);
```

**NOTA:** Establecer el parámetro display_errors con ini_set() en tiempo de ejecución, no es lo mismo que tenerlo habilitado en el entorno de PHP. Es decir, no tendrá ningún efecto si el script contiene errores fatales.

### Etiquetas de Apertura Cortas

Usar siempre etiquetas de apertura de PHP completas, en caso que el servidor no tenga habilitada la directiva short_open_tag.

**INCORRECTO:** `<? echo $foo; ?> <?=$foo?>` **CORRECTO:** `<?php echo $foo; ?>`

### Una Sentencia por Línea

Nunca combinar sentencias en una sola línea.

**INCORRECTO:** `$foo = 'this'; $bar = 'that'; $bat = str_replace($foo, $bar, $bag);` **CORRECTO:** `$foo = 'this'; $bar = 'that'; $bat = str_replace($foo, $bar, $bag);`

### Cadenas

Siempre use cadenas de comillas simples a menos que necesite variables analizadas, y en casos donde necesite variables analizadas, use llaves para impedir ávidos análisis sintácticos de elementos. También puede usar cadenas de comillas dobles si la cadena contiene comillas simples, por lo tanto no hay necesidad de escapar caracteres.

**INCORRECTO:** `"My String" // no hay análisis de variables, por lo tanto no use comillas dobles "My string $foo" // se necesitan llaves 'SELECT foo FROM bar WHERE baz = \'bag\'' // repugnante` **CORRECTO:** `'My String' "My string {$foo}" "SELECT foo FROM bar WHERE baz = 'bag'"`

### Consultas SQL

Las palabras clave de MySQL se ponen siempre en mayúsculas: SELECT, INSERT, UPDATE, WHERE, AS, JOIN, ON, IN, etc.

Dividir las consultar largas en varias líneas para darles legibilidad, preferiblemente cortando en cada cláusula.

**INCORRECTO:** `// las palabras clave están en minúsculas y las consultas son demasiado largas // para una línea simple (... indica continuación de línea) $query = $this->db->query("select foo, bar, baz, foofoo, foobar as raboof, foobaz ...from exp_pre_email_addresses`

```
...where foo != 'oof' and baz != 'zab' order by foobaz limit 5,
100"); CORRECTO: $query = $this->db->query("SELECT foo, bar, baz,
foofoo, foobar AS raboof, foobaz FROM exp_pre_email_addresses WHERE
foo != 'oof' AND baz != 'zab' ORDER BY foobaz LIMIT 5, 100");
```

**Argumentos Por Defecto de Funciones**

Cuando sea adecuado, proveer argumentos por defecto a las funciones, que ayudan a evitar errores de PHP con llamadas erróneas y proveen valores comunes alternativos que pueden salvar unas pocas líneas de código. Ejemplo:

```
function foo($bar = '', $baz = FALSE)
```

## Clase Javascript

CodeIgniter provee una biblioteca para ayudarlo con ciertas funciones comunes que puede querer usar con Javascript. Por favor advierta que CodeIgniter no necesita la biblioteca jQuery para ejecutarse, y que cualquier otra biblioteca de scripting también funcionará igualmente bien. La biblioteca jQuery se presenta simplemente por conveniencia si elije usarla.

## Inicializar la Clase

Para inicializar manualmente la Clase Javascript en el constructor de su controlador, use la función $this->load->library. Actualmente, solamente está disponible la biblioteca jQuery, la que se cargará automáticamente haciendo esto:

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

La Clase Javascript también acepta parámetros, js_library_driver (string) por defecto 'jquery' y autoload (bool) por defecto TRUE. Puede anular los valores por defecto si desea enviar a un array asociativo:

```
$this->load->library('javascript', array('js_library_driver' =>
'scripto', 'autoload' => FALSE));
```

Otra vez, actualmente solo 'jquery' está disponible. Sin embargo, puede querer establecer autoload a FALSE, si no quiere que la biblioteca jQuery incluya automáticamente una etiqueta script para el archivo jQuery principal. Esto es útil si se está cargando desde un lugar fuera de CodeIgniter, o que ya tienen la etiqueta de script en el marcado.

Una vez cargada, el objeto de la biblioteca jQuery estará disponible usando: $this->javascript.

## Instalación y Configuración

### Establecer estas Variables en su Vista

Como con una biblioteca de Javascript, sus archivos tienen que estar disponibles para su aplicación.

Como Javascript es un lenguaje del lado del cliente, la biblioteca tiene que ser capaz de escribir contenido en su salida final. Generalmente, esto significa una vista. Necesitará incluir las siguientes variables en la sección <head> de su salida.

```
<?php echo $library_src;?>
<?php echo $script_head;?>
```

*$library_src*, es donde se cargará el archivo real de la biblioteca, así como cualquier llamada de otro script de plugin; *$script_head* es donde se presentarán eventos específicos, funciones y otros comandos.

### Establecer la ruta a las bibliotecas con ítems de configuración

Hay algunos ítems de configuración en la biblioteca Javascript. Estos se pueden establecer en `application/config/config.php`, dentro de su propio archivo `config/javascript.php` o dentro de cualquier controlador usando la función set_item().

Una imagen se puede usar como "cargador de ajax" o indicador de progreso. Sin ella, aparecerá un simple mensaje de texto de "carga" cuando se necesite hacer la llamada Ajax.

```
$config['javascript_location'] =
'http://localhost/codeigniter/themes/js/jquery/';
$config['javascript_ajax_img'] = 'images/ajax-loader.gif';
```

Si mantiene sus archivos en los mismos directorios desde donde se descargaron, entonces no necesita establecer estos ítems de configuración.

### La Clase jQuery

Para inicializar manualmente la clase jQuery en el constructor de su controlador, use la función $this->load->library:

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

Puede enviar un parámetro opcional para determinar si se incluirá automáticamente o no una etiqueta script al archivo jQuery principal cuando se cargue la biblioteca. Por defecto se creará. Para evitar esto, cargue la biblioteca como se indica:

```
$this->load->library('jquery', FALSE);
```

Una vez cargada, el objeto de la biblioteca jQuery estará disponible usando: $this->jquery.

### Eventos jQuery

Los eventos se establecen usando la siguiente sintaxis.

```
$this->jquery->event('element_path', code_to_run());
```

En el ejemplo anterior:

- "event" es cualquier de estos: blur, change, click, dblclick, error, focus, hover, keydown, keyup, load, mousedown, mouseup, mouseover, mouseup, resize, scroll, o unload.
- "element_path" es cualquier selector jQuery válido. Debido a la sintaxis única de selector de jQuery, este es normalmente un elemento id, o selector CSS. Por ejemplo "#notice_area" afectaría a  y a "#content a.notice" afectaría a todas las anclas con una clase de "notice" en el div con id "content".
- "code_to_run()" es el script que Ud escribe, o una acción tal como un efecto de la biblioteca jQuery.

### Efectos

La biblioteca jQuery soporta un poderoso repertorio de Efectos. Antes que se puede usar un efecto, hay que cargarlo:

```
$this->jquery->effect([ruta opcional] nombre de plugin);

// por ejemplo
$this->jquery->effect('bounce');
```

### hide() / show()

Cada una de estas funciones afectará la visibilidad de un ítem en la página. hide() hará que el ítem sea invisible, mientras que show() lo mostrará.

```
$this->jquery->hide(target, optional speed, optional extra
```

```
information);
$this->jquery->show(target, optional speed, optional extra
information);
```

- "target" será cualquier selector jQuery válido o selectores.
- "speed" es opcional y se establece como slow, normal, fast, o alternativamente a una cantidad de milisegundos.
- "extra information" es opcional y podría incluir un callback u otra información adicional.

**toggle()**

Cambiará la visibilidad de un ítem al opuesto del estado actual, ocultando elementos visibles y volviendo visibles los ocultos.

```
$this->jquery->toggle(target);
```

- "target" será cualquier selector jQuery válido o selectores.

**animate()**

```
$this->jquery->animate(target, parameters, optional speed, optional
extra information);
```

- "target" será cualquier selector jQuery válido o selectores.
- "parameters" en jQuery generalmente incluyen una serie de propiedades CSS que desea cambiar.
- "speed" es opcional y se establece como slow, normal, fast, o alternativamente a una cantidad de milisegundos.
- "extra information" es opcional y podría incluir un callback u otra información adicional.

Podrá ver un resumen completo en http://docs.jquery.com/Effects/animate.

Aquí hay un ejemplo de animate() llamado en un div con id = "note" y disparado por un clic usando el evento click() de la biblioteca jQuery.

```
$params = array(
'height' => 80,
'width' => '50%',
'marginLeft' => 125
);
$this->jquery->click('#trigger', $this->jquery->animate('#note',
$params, normal));
```

**toggleClass()**

Esta función agregará o eliminará un clase CSS al target.

```
$this->jquery->toggleClass(target, class)
```

- "target" será cualquier selector jQuery válido o selectores.
- "class" es cualquier nombre de clase CSS. Advierta que esta clase tiene que estar definida y disponible en un CSS que ya esté cargado.

**fadeIn() / fadeOut()**

Estos efectos causan que los elementos desaparezcan y vuelvan a aparecer con el tiempo.

```
$this->jquery->fadeIn(target, optional speed, optional extra
information);
$this->jquery->fadeOut(target, optional speed, optional extra
information);
```

- "target" será cualquier selector jQuery válido o selectores.
- "speed es opcional y se establece como slow, normal, fast, o alternativamente a una cantidad de milisegundos.
- "extra information es opcional y podría incluir un callback u otra información adicional.

**slideUp() / slideDown() / slideToggle()**

Estos efectos causan que los elementos se deslicen.

```
$this->jquery->slideUp(target, optional speed, optional extra
information);
$this->jquery->slideDown(target, optional speed, optional extra
information);
$this->jquery->slideToggle(target, optional speed, optional extra
information);
```

- "target" será cualquier selector jQuery válido o selectores.
- "speed" es opcional y se establece como slow, normal, fast, o alternativamente a una cantidad de milisegundos.
- "extra information" es opcional y podría incluir un callback u otra información adicional.

**Plugins**

Algunos plugins de jQuery están disponibles usando esta biblioteca.

**corner()**

Se lo usa para agregar distintas esquinas a los elementos de página. Para más detalles ver http://www.malsup.com/jquery/corner/.

```
$this->jquery->corner(target, corner_style);
```

- "target" será cualquier selector jQuery válido o selectores.
- "corner_style" es opcional y se puede establecer a cualquier estilo válido tal como round, sharp, bevel, bite, dog, etc. Las curvas individuales se pueden establecer siguiendo el estilo con un espacio y usando "tl" (top left), "tr" (top right), "bl" (bottom left), or "br" (bottom right).

```
$this->jquery->corner("#note", "cool tl br");
```

**Clase Email**

La robusta clase Email de CodeIgniter soporta las siguientes características:

- Múltiple Protocolos: Mail, Sendmail, y SMTP
- Varios Destinatarios
- CC y BCCs
- HTML o email de texto plano
- Adjuntos
- Ajuste de Linea(Word wrapping)
- Prioridades
- BCC Modo Batch, permitiendo grandes listas de correo electrónico que se divida en pequeños lotes BCC.
- Herramientas de depuración de Email

**Enviar Correo Electrónico**

Enviar un correo electrónico no es simple, pero puedes configurarlo 'al vuelo' o establecer tus preferencias en un archivo de configuración.

Aquí está un ejemplo básico demostrando como debes enviar un email. Nota: Este ejemplo asume que estas enviando un email desde uno de tus controladores.

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

$this->email->from('tu_direccion@tu_sitio.com', 'Tu nombre');
$this->email->to('alguien@ejemplo.com');
$this->email->cc('otro@otro-ejemplo.com');
$this->email->bcc('ellos@su-ejemplo.com');

$this->email->subject('Correo de Prueba');
$this->email->message('Probando la clase email');

$this->email->send();

echo $this->email->print_debugger();
```

**Establecer Preferencias de Correo Electrónico**

Hay 17 diferentes preferencias disponibles para adaptar la forma en la que se envían sus correos electrónicos. Puedes establecer cualquiera de ellos manualmente como se describe aquí, o automáticamente por las preferencias establecidas en tu archivo de configuración, descrita mas abajo:

Las preferencias son establecidas por el paso de valores de un arreglo de preferencias a la funcion initialize(inicialización) de email. Aqui tenemos un ejemplo de como debes establecer algunas preferencias:

```
$config['protocol'] = 'sendmail';
$config['mailpath'] = '/usr/sbin/sendmail';
$config['charset'] = 'iso-8859-1';
$config['wordwrap'] = TRUE;
```

```
$this->email->initialize($config);
```

**Nota:** La mayoría de las preferencias tienen un valor por defecto que será usado si no los estableces

**Establecer preferencias de Email en un archivo de configuración**

Si prefieres no establecer preferencias usando el método anterior, puedes ubicarlos en un archivo de configuración. Simplemente debes crear un nuevo archivo llamado *email.php*, agrega el arreglo *$config* en ese archivo. Luego guárdalo en *config/email.php* y el sera usado automáticamente. Tu NO necesitas usar la función $this->email->initialize() si guardas las preferencias en un archivo de configuración.

**Preferencias de Email**

Lo que sigue es una lista de todas las preferencias que pueden ser establecidas cuando se envia un e-mail.

| Preferencia | Valor por defecto | Opciones | Descripción |
|---|---|---|---|
| **useragent** | CodeIgniter | None | El "user agent". |
| **protocol** | mail | mail, sendmail, or smtp | El protocolo en envio de email. |
| **mailpath** | /usr/sbin/sendmail | None | La ruta al Sendmail. |
| **smtp_host** | No Default | None | Dirección del servidor SMTP. |
| **smtp_user** | No Default | None | Usuario SMTP. |
| **smtp_pass** | No Default | None | Clave SMTP. |
| **smtp_port** | 25 | None | Puerto SMTP. |
| **smtp_timeout** | 5 | None | SMTP Timeout (en segundos). |
| **wordwrap** | TRUE | TRUE or FALSE (boolean) | Activar ajuste de linea. |
| **wrapchars** | 76 | | Cantidad de caracteres para ajustar. |
| **mailtype** | text | text or html | Tipo de correo. Si envias un correo HTML debes enviarla como una pagina web completa. Asegurese de que no tiene links relativos o rutas de imagenes relativas, de otra forma no funcionaran. |
| **charset** | utf-8 | | Juego de Caracteres(utf-8, iso-8859-1, etc.). |
| **validate** | FALSE | TRUE or FALSE (boolean) | Si se validara la direccion del correo. |
| **priority** | 3 | 1, 2, 3, 4, 5 | Prioridad del email. 1 = Alta. 5 = Baja. 3 = Normal. |
| **crlf** | \n | "\r\n" or "\n" or "\r" | Caracter de Nueva Línea. (Use "\r\n" para cumplir con RFC 822) |
| **newline** | \n | "\r\n" or "\n" | Caracter de nueva linea. (Use "\r\n" para cumplir con la RFC 822). |
| **bcc_batch_mode** | FALSE | TRUE or FALSE | Activa el Modo Batch BCC. |

| | | | |
|---|---|---|---|
| | | (boolean) | |
| **bcc_batch_size** | 200 | None | Numero de emails en cada BCC batch. |

**Referencia de la función email**

**$this->email->from()**

Establece la dirección de email y el nombre de la persona que envía el email:

```
$this->email->from('tu@tu-sitio.com', 'Tu Nombre');
```

**$this->email->reply_to()**

Establece la direccion de respuesta. Si la información no es proveida, se usara de la funcion "from". Ejemplo:

```
$this->email->reply_to('tu@tu-sitio.com', 'Tu Nombre');
```

**$this->email->to()**

Establece la(s) direccion(es) de email del destinatario(s). Puede ser una direccion simple, una lista separada por comas o un arreglo:

```
$this->email->to('alguien@ejemplo.com'); $this->email->to('uno@ejemplo.com, dos@ejemplo.com, tres@ejemplo.com'); $list = array('uno@ejemplo.com', 'dos@ejemplo.com', 'tres@ejemplo.com');

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

**$this->email->cc()**

Estable las direcciones CC (con copia). Igual al "to", puede ser una dirección simple, o una lista separada por comas, o un arreglo.

**$this->email->bcc()**

Establece las direcciones BCC (con copia oculta). Es igual a la forma "to", puede ser una direccion simple, una lista delimitada por comas, o un array.

**$this->email->subject()**

Establece el asunto del email:

```
$this->email->subject('Este es el asunto');
```

**$this->email->message()**

Establece el cuerpo del mensaje:

```
$this->email->message('Este es mi mensaje');
```

**$this->email->set_alt_message()**

Establece un cuerpo de mensaje alternativo:

```
$this->email->set_alt_message('Este es el mensaje alternativo');
```

Este es una cadena de mensaje opcional que puede ser usado si envias un email con formato HTML. El te permite especificar un mensaje alternativo sin formato HTML el cual es adherido a la cadena de la cabecera. Para la gente que no acepta email con formato HTML. Si no estableces un mensaje alternativo, CodeIgniter extraera el mensaje de tu email HTML y quitara las etiquetas.

**$this->email->clear()**

Inicializa todas las variables de email a un estado vacio. Esta función esta hecha para ser usada en caso de que se use la funcion enviar email en un ciclo repetitivo (loop), permitiendo que los datos sean restablecidos entre los ciclos.

```
foreach ($lista as $nombre => $direccion)
{
    $this->email->clear();

    $this->email->to($direccion);
    $this->email->from('tu@tu-sitio.com');
    $this->email->subject('Aqui tu informacion '.$nombre);
    $this->email->message('Hola '.$nombre.' Aqui esta la informacion
que solicitaste.');
    $this->email->send();
}
```

Si estableces el parametro a TRUE cualquier adjunto sera limpiado también:

```
$this->email->clear(TRUE);
```

**$this->email->send()**

La función de envío de emails. Retorna el booleano TRUE o FALSE basado en el exito o el fracaso, permitiendo su uso condicionalmente:

```
if ( ! $this->email->send())
{
    // Generar error
}
```

**$this->email->attach()**

Te permite enviar un adjunto. Coloque el path/nombre en el primer parametro. Nota: Use un path al archivo, no una URL. Para multiples adjuntos use la función varias veces. Por ejemplo:

```
$this->email->attach('/path/a/photo1.jpg');
$this->email->attach('/path/a/photo2.jpg');
$this->email->attach('/path/a/photo3.jpg');

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

**$this->email->print_debugger()**

Retorna una cadena conteniendo cualquier mensaje del servidor, la cabecera del email, y el mensaje. Muy util para debugging.

**Sobre-escribir el ajuste de palabras**

Si tienes el ajuste de palabras (word wrapping) activado (recomendado para cumplir con la RFC 822) y tienes un enlace muy largo en tu email, este puede ser ajustado tambien, causando que se vuelva in-clickeable por la persona que lo reciba. CodeIgniter te permite manualmente sobre-escribir el (word wrapping) ajuste de palabra en una parte de tu mensaje como este:

```
El texto de tu email que
sera ajustado normalmente.

{unwrap}http://www.ejemplo.com/un_largo_enlace_que_deberia_no_ser_ajustado.html{/unwrap}

Mas texto que luego deberia
ser ajustado normalmente.
```

Ubique el item que quieres que no sea ajustado entre :*{unwrap} {/unwrap}*

### Clase Encrypt

La clase de encriptacion provee dos formas de encriptacion. Usa un esquema que pre-compila el mensaje usando un algoritmo hash aleatorio XOR esquema de codificación, el cual es entonces encriptado usando la libreria Mcrypt. Si Mcrypt no esta disponible en el servidor el mensaje codificado todavia proveera aun un razonable grado de seguridad para sesiones encriptadas u otros de proposito "ligero". Si Mcrypt está disponible, efectivamente tendrás una cadena de mensaje doblemente encriptada, la cual provee un alto grado de seguridad.

### Estableciendo tú Clave

Una *Clave* es una pieza de información que controla el proceso criptográfico y permite que una cadena encriptada sea decodificada. De hecho, la clave que elijas proveera el **unico** medio para decodificar el dato que has encriptado con esa clave, así no solamente debes elejir cuidadosamente la clave, nunca debes cambiarla si tiene la intención de usarala para para mantener los datos persistentes.

Y ni que decir de que debes ser cuidadoso de donde guardas la clave, pues si alguien obtiene acceso a la clave, las informaciones seran facilmente decodificadas. Si tu servidor no esta completamente bajo tu control es imposible asegurar la clave de seguridad, asi que deberás pensar cuidadosamente antes de usaala en algo que requiere alta seguridad, como almacenar numeros de tarjetas de credito.

Para tomar la maxima ventaja del algoritmo de encriptación, tu clave deberá ser de 32 caracteres de longitud (128 bits). La clave debe ser aleatoriamente una cadena que tu puedes concoct, con numeros y letras tanto mayusculas como minusculas. Tu clave**no** deberá ser una cadena de texto simple. De manera a que sea lo mas seguara criptograficamente hablando se necesita que la clave sea lo mas aleatoria posible.

Tu clave tanto puede ser almacenada en tu application/config/config.php, como puedes diseñar tu propio mecanismo de almacenamiento y pasarla dinamicamente cuando codificas/decodificas.

Para guardar tu clave en tu application/config/config.php, abre el archivo y establece:

```
$config['encryption_key'] = "TU CLAVE";
```

### Longitud del Mensaje

Es importante para usted saber que los mensajes codificados generan aproximadamente 2.6 veces la longitud del mensaje original Por ejemplo, si encriptas la cadena "mi super dato secret", cuya longitud es de 21 caracteres en longitud, terminaras con una cadena que es de aproximadamente 55 caracteres (decimos aproximadamente debido a que la cadena codificada incrementa en agrupaciones de 64 bit, por tanto no es necesariamente linear). Manteniendo esta información en mente cuando seleccione el mecanismo de almacenamietno. Los cookies, por ejemplo pueden mantener solamente 4K de información.

### Inicializando la clase

Igual que en las otras clases en CodeIgniter, la clase Encryptation es inicializada en tu controlador usando la funcion: $this->load->library

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

Una vez cargada, el objeto Encrypt estara disponible usando: $this->encrypt

**$this->encrypt->encode()**

Realizando el cifrado de datos y lo devuelve como una cadena. Ejemplo:

```
$msg = 'Mi mensaje secreto';

$encrypted_string = $this->encrypt->encode($msg);
```

Opcionalmente puedes pasar tu clave de encriptación en el segundo parámetro si no quiere usar la clave que esta en tu archivo de configuración:

```
$msg = 'Mi mensaje secreto';
$key = 'clave-super-secreta';

$encrypted_string = $this->encrypt->encode($msg, $key);
```

**$this->encrypt->decode()**

Desencripta una cadena codificada. Ejemplo:

```
$encrypted_string = 'APANtByIGI1BpVXZTJgcsAG8GZl8pdwwa84';

$plaintext_string = $this->encrypt->decode($encrypted_string);
```

Opcionalmente puedes pasar su clave de encriptación mediante el segundo parámetro si no quiere usar la de tu archivo de configuración:

```
$msg = 'Mi mensaje secreto';
$key = 'clave-super-secreta';

$encrypted_string = $this->encrypt->decode($msg, $key);
```

**$this->encrypt->set_cipher();**

Te permite establecer un cifrado Mcrypt. Por defeco usa `MCRYPT_RIJNDAEL_256`. Ejemplo:

```
$this->encrypt->set_cipher(MCRYPT_BLOWFISH);
```

Por favor visite php.net para tener un alista de cifradores disponibles.

Si prefieres probar manualmente si su servidor soporta Mcrypt puedes usar:

```
echo ( ! function_exists('mcrypt_encrypt')) ? 'Nop' : 'Sip';
```

**$this->encrypt->set_mode();**

Te permite establecer un modo MCRYPT. Por defecto usa `MCRYPT_MODE_ECB`. Ejemplo:

```
$this->encrypt->set_mode(MCRYPT_MODE_CFB);
```

Por favor visite php.net para tener un alista de modos disponibles.

**$this->encrypt->sha1();**

La funcion de codificacion SHA1. Provee una cadena y la retorna un hash de 160 bit de una sola forma. Nota: SHA1, es como el MD5 es no-decodificable. Ejemplo:

```
$hash = $this->encrypt->sha1('alguna cadena');
```

Muchas instalaciones de PHP tienen soporte para SHA1 por defecto, por tanto lo unico que necesitas para codificar es usar la funcion nativa:

```
$hash = sha1('Alguna Cadena');
```

Si tu servidor no soporta SHA1 puedes usar la funcion proveida.

### $this->encrypt->encode_from_legacy($orig_data, $legacy_mode = MCRYPT_MODE_ECB, $key = '');

Le permite recodificar los datos que se encriptaron originalmente con CodeIgniter 1.x para que sean compatibles con la biblioteca Encrypt de CodeIgniter 2.x. Solamente es necesario usar este método si tiene datos encriptados almacenados permanentemente como en un archivo o base de datos en un servidor que soporte Mcrypt. "Light" usa encriptación tal como datos de sesión encriptados o flashdata encriptados transitorios que no necesitan intervención de su parte. Sin embargo, las sesiones existentes encriptadas se destruirán ya que los datos encriptados antes de 2.x no se decodificaban.

**¿Por qué solamente un método para recodificar los datos, en lugar de mantener los métodos anteriores tanto para codificación como para decodificación?** Los algoritmos de la biblioteca Encrypt se mejoraron en CodeIgniter 2.x, tanto en rendimiento como seguridad y no queremos incentivar que se sigan usando métodos viejos. Por supuesto, puede extender la biblioteca Encryption si lo desea y reemplazar los métodos nuevos con los viejos y mantener una compatibilidad total con los datos encriptados con CodeIgniter 1.x, pero esta es una decisión que, en todo caso, tiene que hacer con cuidado y deliberadamente un desarrollador.

```
$new_data = $this->encrypt->encode_from_legacy($old_encrypted_string);
```

| Parámetro | Por Defecto | Descripción |
|---|---|---|
| **$orig_data** | n/a | Datos originales encriptados con la biblioteca Encryption de CodeIgniter 1.x |
| **$legacy_mode** | MCRYPT_MODE_ECB | Modo de Mcrypt que se usó para generar los datos encriptados originales. El valor por defecto de CodeIgniter 1.x era MCRYPT_MODE_ECB y se asumirá este valor, a menos que se lo anule con este parámetro. |
| **$key** | n/a | Clave de encriptación. Se lo especifica normalmente en su archivo de configuración como se describe anteriormente. |

**Clase de Manipulación de Imagen**

La clase de Manipulación de Imagen de CodeIgniter le permite realizar las siguientes acciones:

- Redimensión de Imagen
- Creación de Thumbnail
- Recorte de Imagen
- Rotación de Imagen
- Marca de Agua en Imagen

Todas las tres librerías de imágenes principales son soportadas: GD/GD2, NetPBM, e ImageMagick

**Nota:** Marca de Agua sólo está disponible usando la librería GD/GD2. Además, aunque las otras librerías son soportadas, GD es requerida para que el programa calcule las propiedades de la imagen. El procesamiento de la imagen, sin embargo, será realizado con la librería que haya especificado.

**Inicializar la Clase**

Como la mayoría de las otras clases en CodeIgniter, la clase de imagen es inicializada en su controlador usando la función $this->load->library:

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

Una vez que la librería es cargada estará lista para el uso. El objeto de la librería de imagen que usará para llamar todas las funciones es: $this->image_lib

**Procesar una Imagen**

Sin importar el tipo de procesamiento que quiera realizar (redimencionar, recortar, rotar o marca de agua), el proceso general es idéntico. Establecerá algunas preferencias correspondientes a la acción que tenga intenciones de realizar, entonces llamará a una de las cuatro funciones de procesamiento disponibles. Por ejemplo, para crear un thumbnail de una imagen hará esto:

```
$config['image_library'] = 'GD';
$config['source_image'] = '/ruta/a/imagen/mifoto.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();
```

El código anterior le dice a la función image_resize que busque por una imagen llamada *mifoto.jpg* ubicada en la carpeta source_image, entonces cree un thumbnail que sea de 75 X 50 pixeles usando la image_library GD2. Ya que la opción maintain_ratio está habilitada, el thumb será tan cercano al width y height de destino como sea posible mientras preserve la proporción del aspecto original. El thumbnail será llamado *mifoto_thumb.jpg*

**Nota:** Para que la clase de imagen pueda hacer algún procesamiento, la carpeta contenedora de los archivos de imagen debe tener permisos de 777.

**Nota:** El procesamiento de imágenes puede requerir una cantidad considerable de memoria del servidor para algunas operaciones. Si experimenta errores por agotamiento de memoria mientras procesa imágenes, puede necesitar limitar sus tamaños máximos y/o ajustar los límites de memoria de PHP.

**Funciones de Procesamiento**

Hay cuatro funciones de procesamiento:

- $this->image_lib->resize()
- $this->image_lib->crop()
- $this->image_lib->rotate()
- $this->image_lib->watermark()
- $this->image_lib->clear()

Estas funciones devuelve un buleano TRUE si funcionan y FALSE si fallan. Si fallan puede recuperar el mensaje de error usando esta función:

```
echo $this->image_lib->display_errors();
```

Una buena costumbre es usar la función de procesamiento condicionalmente, mostrando un error cuando falla, así:

```
if ( ! $this->image_lib->resize())
{
    echo $this->image_lib->display_errors();
}
```

Nota: Puede opcionalmente especificar el formato HTML a ser aplicado a los errores, enviando las etiquetas de apertura/clausura en la función, así:

```
$this->image_lib->display_errors('<p>', '</p>');
```

**Preferencias**

Las preferencias que se describen debajo, le permiten adaptar el procesamiento de imágenes para satisfacer sus necesidades.

Note que no todas las preferencias están disponibles para cada función. Por ejemplo, las preferencias de hachas x/y sólo están disponibles para recortar imagen. De la misma manera, Las preferencias de ancho y alto no tienen efecto al recortar. La columna "disponibilidad" indica que función soporta una preferencia determinada.

Disponibilidad (columna Disp.):

- *R* - Redimensionar Imagen
- *C* - Recortar Imagen
- *X* - Rotar Imagen
- *W* - Marca de Agua

| Preferencia | Valor por Defecto | Opciones | Descripción | Disponibilidad |
|---|---|---|---|---|
| **image_library** | GD2 | GD, GD2, ImageMagick, NetPBM | Establece la librería de imagen a ser usada. | R, C, X, W |

| | | | | |
|---|---|---|---|---|
| **library_path** | Ninguna | Ninguna | Establece la ruta del servidor a su librería ImageMagick o NetPBM. Si usa una de estas librerías debe suministrar la ruta. | R, C, X |
| **source_image** | Ninguna | Ninguna | Establece el nombre/ruta de la imagen de origen. La ruta debe ser una ruta relativa o absoluta del servidor, no una URL. | R, C, S, W |
| **dynamic_output** | FALSE | TRUE/FALSE (buleano) | Determina si el nuevo archivo de imagen debe ser escrito a disco o generado dinámicamente. Nota: Si elije la configuración dinámica, sólo una imagen puede ser mostrada a la vez, y no puede ser posicionada en la página. Simplemente muestra la imagen dinámicamente a su explorador, junto con los encabezados de imagen. | R, C, X, W |
| **quality** | 90% | 1 - 100% | Establece la calidad de la imagen. Mayor calidad implica mayor tamaño del archivo. | R, C, X, W |
| **new_image** | Ninguna | Ninguna | Establece el nombre/ruta de destino de la imagen. Usará esta preferencia cuando cree una copia de una imagen. La ruta debe ser una ruta relativa o absoluta del servidor, no una URL. | R, C, X, W |
| **width** | Ninguna | Ninguna | Establece el ancho que quiere establecerle a la imagen. | R, C |
| **height** | Ninguna | Ninguna | Establece el alto que quiere establecerle a la imagen. | R, C |
| **create_thumb** | FALSE | TRUE/FALSE (boolean) | Le dice a la función de procesamiento de imagen que cree un thumb. | R |
| **thumb_marker** | _thumb | Ninguna | Especifica el indicador de thumbnail. Será insertado justo antes de la extensión del archivom así que mifoto.jpg será mifoto_thumb.jpg | R |
| **maintain_ratio** | TRUE | TRUE/FALSE (boolean) | Especifica si mantener la proporción del aspecto original cuando redimensiona o usar valores duros. | R |
| **master_dim** | auto | auto, width, height | Especifica que usar como la hacha maestra cuando redimensiona o crea thumbs. Por ejemplo, digamos que quiere redimensionar una imagen a 100 X 75 pixeles. Si su imagen de origen no permite redimensionamiento perfecto a esas dimensiones, esta configuración determina que hacha debe ser usada como valor duro. "auto" establece el hacha automáticamente basado en si la imagen es más alta que ancha o viceversa. | R |
| **rotation_an** | Ninguna | 90, 180, | Especifica el ángulo de rotación cuando se | X |

| | | | | |
|---|---|---|---|---|
| **gle** | | 270, vrt, hor | rotan imagenes. Note que PHP rota contra las agujas del reloj, así que una rotación de 90 grados a la derecha debe ser especificada como 270. | |
| **x_axis** | Ninguna | Ninguna | Establece la cordenada X en pixeles para recortar una imagen. Por ejemplo, una configuración de 30 cortará una imagen 30 pixeles desde la izquierda. | C |
| **y_axis** | Ninguna | Ninguna | Establece la cordenada X en pixeles para recortar una imagen. Por ejemplo, una configuración de 30 cortará una imagen 30 pixeles desde arriba. | C |

**Estableciendo preferencias en un archivo de configuración**

Si prefiere no establecer preferencias usando los métodos anteriores, puede en su lugar ponerlos en un archivo de configuración. Simplemente cree un nuevo archivo llamado *image_lib.php* y el arreglo *$config* en ese archivo. Entonces guarde el archivo en: *config/image_lib.php* y será usado automáticamente. NO necesitará usar la función $this->image_lib->initialize si guarda sus preferencias en un archivo de configuración.

**$this->image_lib->resize()**

La función de redimensión de imagen le permite redimensionar la imagen original, crear una copia (con o sin redimensión), o crear una imagen thumbnail.

Por propósitos prácticos no hay diferencia entre crear una copia y crear un thumbnail excepto que un thumb tendrá el marcador thumbnail como parte de su nombre (por ejemplo, mifoto_thumb.jpg).

Todas las preferencias listadas en la tabla anterior están disponibles por esta función excepto estas tres: rotation, x_axis, and y_axis.

**Creando un Thumbnail**

La función de redimensión creará un archivo thumbnail (y preserva el original) si establece esta preferencia como TRUE:

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

Esta preferencia sólo determina un thumbnail es creado o no.

**Creando una Copia**

La función de redimensión creará una copia del archivo de imagen (y preservará el original) su establece una ruta y/o un nuevo nombre de archivo usando esta preferencia:

```
$config['new_image'] = '/ruta/a/nueva_imagen.jpg';
```

Nota acerca de esta preferencia:

- Si sólo el nombre de la nueva imagen es especificado será ubicada en la misma carpeta que el original
- Si sólo la ruta es especificada, la nueva imagen será ubicada en el destino con el mismo nombre que el original.

- Si ambos, la ruta y nombre de la imagen, son especificados será ubicado en su propia destinación y con el nuevo nombre dado.

**Redimensionando la Imagen Original**

Si ninguno de las dos preferencias listadas arriba (create_thumb, y new_image) son usadas, la función de redimensión usará la imagen original como destino de procesamiento.

**$this->image_lib->crop()**

La función de recortar funciona casi idénticamente a la función de redimensión excepto que requiere que establezca preferencias para las hachas X e Y (en pixeles) especificando donde cortar, así:

```
$config['x_axis'] = '100';
$config['y_axis'] = '40';
```

Todas las preferencias listadas en la tabla anterior están disponibles excepto estas: rotation, width, height, create_thumb, new_image.

Aquí hay un ejemplo mostrando como puede recortar una imagen:

```
$config['image_library'] = 'imagemagick';
$config['library_path'] = '/usr/X11R6/bin/';
$config['source_image'] = '/ruta/a/imagen/mifoto.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();
}
```

Nota: Sin una interface visual es difícil recortar imagenes, así que esta función no es muy útil a menos que tenga intenciones de construir dicha interface. Esto es exactamente lo que hicimos para el módulo de galería de fotos en ExpressionEngine, el CMS que desarrollamos. Agregamos una Interface de Usuario que permite que el area a recortar sea seleccionada.

**$this->image_lib->rotate()**

La función de rotación de imagen requiere que el ángulo de rotación sea establecido a través de esta preferencia:

```
$config['rotation_angle'] = '90';
```

Hay 5 opciones de rotación:

1. 90 - rota 90 grados contra las agujas del reloj.
2. 180 - rota 180 grados contra las agujas del reloj.
3. 270 - rota 270 grados contra las agujas del reloj..
4. hor - Voltea la imagen horizontalmente.
5. vrt - Voltea la imagen verticalmente.

Aquí hay un ejemplo mostrando como puede rotar una imagen:

```
$config['image_library'] = 'netpbm';
$config['library_path'] = '/usr/bin/';
$config['source_image'] = '/ruta/a/imagen/mifoto.jpg';
$config['rotation_angle'] = 'hor';

$this->image_lib->initialize($config);

if ( ! $this->image_lib->rotate())
{
    echo $this->image_lib->display_errors();
}
```

**$this->image_lib->clear()**

La función "clear" restablece todos los valores por defecto usados cuando se procesa una imagen. Usted podría querer llamarla si está procesando imágenes en un ciclo.

```
$this->image_lib->clear();
```

**Marca de Agua**

La característica Marca de Agua requiere la librería GD/GD2.

**Dos tipos de Marca de Agua**

Hay dos tipos de marca de agua que puede usar:

- **Texto**: El mensaje de marca de agua será generado usando texto, ya sea con una fuente True Type que especifique, o usando la salida de texto nativo que la librería GD soporta. Si usa la versión True Type, su instalación de GD debe ser compilada con soporte True Type (la mayoría lo son, pero no todos).
- **Revestimiento**: El mensaje de marca de agua será generado al revestir la imagen (usualmente un PNG o GIF transparent) conteniendo su marca de agua sobre la imagen de origen.

**Marca de agua en una imagen**

Al igual que con la otra funciones el procesamiento general para marca de agua involucra establecer las preferencias correspondientes a la acción que intente realizar, entonces llamar a la función "watermark". Aquí hay un ejemplo:

```
$config['source_image'] = '/ruta/a/imagen/mifoto.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();
```

El ejemplo anterior usará una fuente de 16 píxeles para crear el texto "Copyright 2006 - John Doe". La marca de agua será posicionada abajo/al centro de la imagen, 20 píxeles desde el fondo de la imagen.

**Nota:** Para que la clase de imagen pueda hacer cualquier procesamiento, el archivo de imagen debe tener permisos de 777.

### Preferencias de Marca de Agua

Esta tabla muestra las preferencias que están disponibles para ambos tipos de marcas de agua (texto o revestimiento)

| Preferencia | Valor por defecto | Opciones | Descripción |
|---|---|---|---|
| **wm_type** | text | type, overlay | Establece el tipo de marca de agua que debe ser usado. |
| **source_image** | Ninguna | Ninguna | Establece el nombre/ruta de la imagen de origen. La ruta debe ser relativa o una ruta absoluta del servidor, no una URL. |
| **dynamic_output** | FALSE | TRUE/FALSE (boolean) | Determina si el nuevo archivo de imagen debe ser escrito a disco o generado dinámicamente. Nota: Si elije la configuración dinámica, sólo una imagen puede ser mostrada a la vez, y no puede ser posicionada en la página. Simplemente muestra la imagen dinámicamente a su explorador, junto con los encabezados de imagen. |
| **quality** | 90% | 1 - 100% | Establece la calidad de la imagen. Mientras más alta sea la calidad, mayor será el tamaño del archivo. |
| **padding** | Ninguna | Un número | La cantidad de espaciado, establecido en píxeles, que serán aplicados a la marca de agua para establecer la distancia desde el borde de sus imágenes. |
| **wm_vrt_alignment** | bottom | top, middle, bottom | Establece el alineamiento vertical para la marca de agua de la imagen. |
| **wm_hor_alignment** | center | left, center, right | Establece el alineamiento horizontal para la marca de agua de la imagen. |
| **wm_vrt_offset** | Ninguna | Ninguna | Puede especificar un punto de comienzo vertical (en pixeles) para aplicar a la posición de la marca de agua. El punto normalmente mueve la marca de agua hacia abajo, excepto que haya establecido alineamiento como "down" (abajo) entonces el valor de comienzo moverá la marca de agua hacia arriba en la imagen. |
| **wm_hor_offset** | Ninguna | Ninguna | Puede especificar un punto de comienzo horizontal (en pixeles) para aplicar a la posición de la marca de agua. El punto normalmente mueve la marca de agua a la derecha, excepto que haya establecido alineamiento como "right" (a la derecha) entonces el valor de comienzo moverá la marca de agua a la izquierda de la imagen. |

### Preferencia de Texto

Esta tabla muestra las preferencias que están disponibles para el tipo de texto de la marca de agua.

| Preferencia | Valor por defecto | Opciones | Descripción |
|---|---|---|---|
| **wm_text** | Ninguna | Ninguna | El texto que quiere mostrar como marca de agua. Típicamente este será una nota de derechos registrados. |
| **wm_font_path** | Ninguna | Ninguna | La ruta del servidor a la fuente True Type que quiera usar. Si no usa esta opción, la fuente nativa de la GD será usada. |
| **wm_font_size** | 16 | Ninguna | El tamaño del texto. Nota: si no está usando la opción True Type anterior, el número es establecido usando un rango de 1 - 5. De otra forma, puede usar cualquier tamaño de pixel válido para la guente que está usando. |
| **wm_font_color** | ffffff | Ninguna | El color de la fuente, especificada en hexadecimal. Nota, debe usar los 6 caracteres de valor hexadecimal completos (por ejemplo, 993300), en vez de la versión abreviada de tres caracteres (por ejemplo, fff). |
| **wm_shadow_color** | Ninguna | Ninguna | el color de la sombra, especificada en hexadecimal. Si deja esta opción vacía, no se usará sombra. Nota, debe usar los 6 caracteres de valor hexadecimal completos (por ejemplo, 993300), en vez de la versión abreviada de tres caracteres (por ejemplo, fff). |
| **wm_shadow_distance** | 3 | Ninguna | La distancia (en pixeles) desde la fuente a la que la sombra debe aparecer. |

**Preferencias de Revestimiento**

Esta tabla muestra las preferencias que están disponibles para el tipo de revestimiento de la marca de agua.

| Preferencia | Valor por defecto | Opciones | Descripción |
|---|---|---|---|
| **wm_overlay_path** | Ninguna | Ninguna | La ruta del servidor a la imagen que quiere usar como marca de agua. Requerido sólo si usa el método de revestimiento. |
| **wm_opacity** | 50 | 1 - 100 | Opacidad de la imagen. Puede especificar la opacidad (por ejemplo, transparencia) de su imagen de marca de agua. Esto permite a la marca de agua ser débil y no oscurecer completamente los detalles de la imagen detrñas de ella. Una opacidad del 50% es típica. |
| **wm_x_transp** | 4 | Un número | Si su imagen de marca de agua es una imagen PNG o GIF, puede especificar un color a la imagen a ser "transparente". Esta opción (junto con la próxima) le permitirá especificar ese color. Esto funciona expecificando la coordenada de pixel "X" e "Y" (medida desde arriba a la izquierda) dentro de la imagen que corresponde a un pixel representativo del color que quiere transparentar. |
| **wm_y_transp** | 4 | Un número | Junto con la opción anterior, esto le permite especificar la coordenada a un pixel representativo del color que quiere transparentar. |

**Clase Calendar**

La clase Calendar permite crear dinamicamente calendarios. Sus calendarios pueden formatearse a través del uso de una plantilla calendario, permitiendo un 100% de control sobre todos los aspectos de su diseño. Además, puede pasar datos a las celdas de su calendario.

**Inicializando la clase**

Similar a la mayoría de las clases en codeIgniter, la clase calendar es inicializada en su controlador usando la función $this->load->library:

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

Una vez cargado, el objeto Calendario estará disponible usando: $this->calendar

**Mostrando un calendario**

Éste es un ejemplo muy simple que enseña como puede mostrar un calendario:

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

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

El código anterior generará un calendario para el mes/año actual basado en la hora de su servidor. Para mostrar un calendario para un mes y año específicos deberá pasar esta infromación a la función de generación de calendario:

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

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

El código de arriba generará un calendario que muestra el mes de junio del año 2006. El primer parámetro especifica el año, el segundo parámetro especifica el mes.

**Pasando datos a las celdas de tu calendario**

Agregar datos a las celdas de tu calendario involucra crear un arreglo asociativo en el cual los índices corresponden a los días que desea llenar y el valor del arreglo contiene los datos. El arreglo es pasado al tercer parámetro de la función generar calendario. Considere este ejemplo:

```
$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);
```

Usando el ejemplo anterior, los días número 3, 7, 13 y 26 se convertirán en enlaces que apuntan a las URLs provistas.

**Nota:** Por defecto se adume que el array contiene links. En la sección que explica la plantilla calendario verá como puede personalizar el modo en que los datos son pasados, de manera que puede pasar tipos diferentes de información

### Ajustando las Preferencias de Visualización

Hay siete preferencias que puede ajustar para controlar varios aspectos del calendario. Las preferencias son ajustadas pasando un arreglo de preferencias en el segundo parámetro de la funcion loading. Este es un ejemplo:

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

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

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

El código citado anteriormente haría el calendario comenzar en sábado, usa la cabezera de mes "largo", y los nombres de días "pequeños". Más información con respecto a las preferencias a continuación.

| Preferencia | Valor por defecto | Opciones | Descripción |
|---|---|---|---|
| **template** | None | None | Un cadena que contiene su plantilla calendario. Ver la seccion plantilla debajo. |
| **local_time** | time() | None | Un timestamp Unix que corresponde al tiempo actual. |
| **start_day** | sunday | Any week day (sunday, monday, tuesday, etc.) | Ajusta el día de la semana con que el calendario debería iniciar. |
| **month_type** | long | long, short | Determina que versión del nombre del mes usar en la cabecera. long = January, short = Jan. |
| **day_type** | abr | long, short, abr | Determina que versión de los nombre de los días de la semana usar en la cabecera de la columna. long = Domingo, short = Dom, abr = Do. |
| **show_next_prev** | FALSE | TRUE/FALSE (boolean) | Determina si se mostrarán los links que permiten ir al mes siguiente/previo. Ver información sobre esta característica debajo. |
| **next_prev_url** | None | A URL | Ajusta el basepath usado en los links del calendario siguiente/previo. |

### Mostrando los Links Mes Siguiente/Previo

Permitir a su calendario incrementar/decrementar dinámicamente a través de los links next/previous requiere que establezca su código de calendario similar a este ejemplo:

```
$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));
```

Advertirá algunas cosas acerca del ejemplo citado anteriormente:

- Debe ajustar el "show_next_prev" a TRUE.
- Debe proveer la URL al controlador que contiene su calendario en la preferencia "next_prev_url" preference.
- Debe proveer el "Año" y el "Mes" a la función de generación de calendario mediante los segmenteos URI donde aparecen (Nota: La clase calendario automáticamente agrega el año/mes a la URL base provista.).

**Creando una Plantilla Calendario**

Creando una plantilla calendario puede tener 100% de control sobre el diseño de su calendario. Cada componente de su calendario estará contenido dentro de un par de pseudo-variables como se muestra aquí:

```
$prefs['template'] = '

    {table_open}<table border="0" cellpadding="0" cellspacing="0">{/table_open}

    {heading_row_start}<tr>{/heading_row_start}

    {heading_previous_cell}<th><a href="{previous_url}">&lt;&lt;</a></th>{/heading_previous_cell}
    {heading_title_cell}<th colspan="{colspan}">{heading}</th>{/heading_title_cell}
    {heading_next_cell}<th><a href="{next_url}">&gt;&gt;</a></th>{/heading_next_cell}

    {heading_row_end}</tr>{/heading_row_end}

    {week_row_start}<tr>{/week_row_start}
    {week_day_cell}<td>{week_day}</td>{/week_day_cell}
    {week_row_end}</tr>{/week_row_end}

    {cal_row_start}<tr>{/cal_row_start}
    {cal_cell_start}<td>{/cal_cell_start}

    {cal_cell_content}<a href="{content}">{day}</a>{/cal_cell_content}
    {cal_cell_content_today}<div class="highlight"><a href="{content}">{day}</a></div>{/cal_cell_content_today}

    {cal_cell_no_content}{day}{/cal_cell_no_content}
    {cal_cell_no_content_today}<div class="highlight">{day}</div>{/cal_cell_no_content_today}

    {cal_cell_blank} {/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();
```

## Clase Form Validation

CodeIgniter provee una clase para preparación de datos y validación de formularios que ayuda a minimizar la cantidad de código que se escribe.

### Introducción

Antes de explicar el enfoque de CodeIgniter para validar datos, describiremos el escenario ideal:

1. Se muestra un formulario.
2. Usted completa los datos y lo envía.
3. Si el envío tiene algo inválido, o falta algo que sea obligatorio, el formulario se muestra nuevamente con los datos que Ud completó junto con un mensaje de error que describe el problema.
4. Este proceso continua hasta que se envíe un formulario válido.

En el receptor, el script debe:

1. Verificar los datos obligatorios.
2. Verificar que los datos son del tipo correcto y coinciden con el criterio correcto. Por ejemplo, si se envía un usuario, tiene que ser válido y contener solamente caracteres permitidos. Tiene que tener una longitud mínima y no exceder la longitud máxima. El usuario tiene que existir, no puede ser una palabra reservada, etc.
3. Por seguridad, descontaminar los datos.
4. Preformatear los datos si es necesario (¿Se necesita recortar espacios al inicio o final? ¿Codificación HTML? Etc.)
5. Preparar los datos para insertarlos en la base de datos.

Aunque el proceso anterior no es terriblemente complejo, normalmente requiere de una cantidad significativa de código, mostrar mensajes de error. Normalmente varias estructuras de control se colocan dentro del formulario HTML. La validación de formularios, aunque es simple de crear, generalmente es muy tediosa y enmarañada de implementar

### Tutorial de Validación de Formularios

Lo que sigue es un tutorial "práctico" para implementar la Validación de Formularios de CodeIgniter.

Para implementar la validación de formularios necesitará tres cosas:

1. Un archivo de Vista que contenga un formulario.
2. Un archivo de Vista que contiene un mensaje de "éxito" para mostrarse cuando el envío sea exitoso.
3. Una función Controlador para recibir y procesar los datos enviados.

Vamos a crear esas tres cosas, usando un formulario de registro como ejemplo.

### El formulario

Usando un editor de texto, crear un formulario llamado mi_form.php. Dentro suyo, ubicar este código y guardarlo en su carpeta `application/views/`:

```
<html>
<head>
<title>Mi Formulario</title>
</head>
<body>

<?php echo validation_errors(); ?>

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

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

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

<h5>Confirmar contraseña</h5>
<input type="text" name="passconf" value="" size="50" />

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

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

</form>

</body>
</html>
```

**La Página de Éxito**

Usando un editor de texto, crear un formulario llamado form_success.php. Dentro suyo, colocar este código y guardarlo en su carpeta `application/views/`:

```
<html>
<head>
<title>Mi Formulario</title>
</head>
<body>

<h3>Se envió correctamente su formulario!</h3>

<p><?php echo anchor('form', '¡Inténtelo otra vez!'); ?></p>

</body>
</html>
```

**El Controlador**

Usando un editor de texto, crear un controlador llamado form.php. Dentro suyo, colocar este código y

guardarlo en su carpeta `application/controllers/`:

```
<?php

class Form extends CI_Controller {

    function index()
    {
        $this->load->helper(array('form', 'url'));

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

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

**Pruébelo!**

Para probar su formulario, visite su sitio usando una URL similar a esta:

`example.com/index.php/`*form*`/`

Si envía el formulario debería ver recargar el formulario. Esto es porque no se estableció ninguna regla de validación todavía.

**Como todavía no se le dijo a la Clase Form_validation que valide algo todavía, devuelve** `FALSE` **(booleano) por defecto. La función** `run()` **solamente devuelve** `TRUE` **si se aplicaron satisfactoriamente las reglas sin que ninguna fallara.**

**Explicación**

Advertirá varias cosas acerca de las páginas anteriores:

El formulario (mi_form.php) es un formulario web estándar con algunas excepciones:

1. Usa un helper form para crear la apertura del formulario. Técnicamente esto no es necesario. Podría crear el formulario usando HTML estándar. Sin embargo, el beneficio de usar el helper es que genera la URL de acción por Ud, basado en la URL en su archivo de configuración. Esto hace su aplicación más portable en caso que su URL cambie.
2. En la parte superior del formulario advertirá la siguiente llamada de función: `<?php echo validation_errors(); ?>`

   Esta función devolverá cualquier mensaje de error enviado de regreso por el validador. Si no hay mensajes devuelve una cadena vacía.

El controlador (form.php) tiene una función: index(). Esta función inicializa la clase form_validation y carga los *helpers form* y *URL* usados por sus archivos de vista. También `ejecuta` la rutina de validación. Basado en si la validación fue exitosa o no, presenta tanto el formulario como la página de éxito.

**Establecer Reglas de Validación**

CodeIgniter le permite establecer tantas reglas de validación como necesite para un campo dado, pornerlas en cascada, e incluso preparar y preprocesar los datos de los campos al mismo tiempo. Para establecer las reglas de validación usará la función set_rules():

```
$this->form_validation->set_rules();
```

La función anterior toma como entrada **tres** parámetros:

1. Nombre del campo - el mismo que le dio al campo del formulario.
2. Un nombre "humano" para este campo, que deberá insertarse en el mensaje de error. Por ejemplo, si el campo se llama "usuario" puede darle un nombre humano como "Nombre de Usuario". **Nota:** Si quisiera que el nombre del campo sea almacenado en un archivo de idioma, por favor lea Traducir los Nombres de Campo.
3. Las reglas de validación para este campo de formulario.

Este es un ejemplo. En su controlador (form.php), agregue este código justo debajo de la función de inicialización de la validación:

```
$this->form_validation->set_rules('username', 'Usuario', 'required');
$this->form_validation->set_rules('password', 'Contraseña',
'required');
$this->form_validation->set_rules('passconf', 'Confirmar Contraseña',
'required');
$this->form_validation->set_rules('email', 'Email', 'required');
```

Su controlador debería verse así:

```
<?php
class Form extends CI_Controller {
    function index()
    {
        $this->load->helper(array('form', 'url'));

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

        $this->form_validation->set_rules('username', 'Usuario',
'required');
        $this->form_validation->set_rules('password', 'Contraseña',
'required');
        $this->form_validation->set_rules('passconf', 'Confirmar
Contraseña', 'required');
        $this->form_validation->set_rules('email', 'Email',
'required');

        if ($this->form_validation->run() == FALSE)
        {
```

```
            $this->load->view('mi_form');
        }
        else
        {
            $this->load->view('form_success');
        }
    }
} ?>
```

Ahora envíe el formulario con los campos en blanco y debería ver los mensajes de error. Si envía el formulario con todos los campos llenos, verá la página de éxito.

**Nota:** Cuando hay un error, los campos del formulario no se vuelven a llenar con los datos. Haremos esto en breve.

### Establecer Reglas Usando un Array

Antes de seguir, debe notarse que la función que establece la regla se le puede pasar un array si prefiere establecer todas las reglas en una sola acción. Si usa este enfoque, deberá llamar a las claves del array según se indica:

```
$config = array(
               array(
                     'field'   => 'username',
                     'label'   => 'Usuario',
                     'rules'   => 'required'
                  ),
               array(
                     'field'   => 'password',
                     'label'   => 'Contaseña',
                     'rules'   => 'required'
                  ),
               array(
                     'field'   => 'passconf',
                     'label'   => 'Confirmar Contraseña',
                     'rules'   => 'required'
                  ),
               array(
                     'field'   => 'email',
                     'label'   => 'Email',
                     'rules'   => 'required'
                  )
            );

$this->form_validation->set_rules($config);
```

### Reglas en Cascada

CodeIgniter le permite agrupar varias reglas juntas. Probémoslo. Cambie sus reglas en el tercer parámetro de la función que establece las reglas por esto:

```
$this->form_validation->set_rules('username', 'Usuario',
'required|min_length[5]|max_length[12]');
$this->form_validation->set_rules('password', 'Contraseña',
'required|matches[passconf]');
$this->form_validation->set_rules('passconf', 'Confirmar Contraseña',
'required');
$this->form_validation->set_rules('email', 'Email',
'required|valid_email');
```

El código anterior establece las siguiente reglas:

1. El campo usuario no tiene que ser menor que 5 caracteres y no mayor que 12.
2. El campo contraseña tiene que coincidir con el campo confirmar contraseña.
3. El campo email tiene que contener una dirección de email válida.

Pruébelo! envíe su formulario sin los datos adecuados y verá los nuevos mensajes de error que corresponden a sus nuevas reglas. Hay muchas reglas disponibles, sobre las que puede leer en la referencia de validación.

**Preparar Datos**

Además de las funciones de validación como las usadas antes, también puede preparar sus datos de otras varias formas. Por ejemplo, puede configurar las reglas así:

```
$this->form_validation->set_rules('username', 'Usuario',
'trim|required|min_length[5]|max_length[12]|xss_clean');
$this->form_validation->set_rules('password', 'Contraseña',
'trim|required|matches[passconf]|md5');
$this->form_validation->set_rules('passconf', 'Confirmar Contraseña',
'trim|required');
$this->form_validation->set_rules('email', 'Email',
'trim|required|valid_email');
```

En el ejempo anterior, eliminamos los espacios al comienzo y fin de los campos, convertimos la contraseña a MD5 y aplicamos la función xss_clean() al usuario, la que remueve datos maliciosos.

**Se puede usar como regla cualquier función nativa de PHP que acpete solamente un parámetro, como** htmlspecialchars**,** trim**,** MD5**, etc.**

**Nota:** En general, tendrá que usar las funciones de preparación **después** de las reglas de validación, por lo que si hay un error, los datos originales se mostrarán en el formulario.

**Volver a Llenar el Formulario**

Hasta ahora sólo hemos estado tratando con errores. Es tiempo de volver a llenar los campos del formulario con los datos enviados. CodeIgniter ofrece varias funciones helper que le permiten hacer esto. La que usará mayormente es:

```
set_value('nombre_de_campo')
```

Abra su archivo de vista mi_form.php y actualice el **valor** de cada campo usando la función set_value():

**No olvide incluir cada nombre de campo en las funciones** set_value()**!**

```
<html>
<head>
<title>Mi Formulario</title>
</head>
<body>

<?php echo validation_errors(); ?>

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

<h5>Usuario</h5>
<input type="text" name="username" value="<?php echo
set_value('username'); ?>" size="50" />

<h5>Contraseña</h5>
<input type="text" name="password" value="<?php echo
set_value('password'); ?>" size="50" />

<h5>Confirmar contraseña</h5>
<input type="text" name="passconf" value="<?php echo
set_value('passconf'); ?>" size="50" />

<h5>Email</h5>
<input type="text" name="email" value="<?php echo set_value('email');
?>" size="50" />

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

</form>

</body>
</html>
```

Ahora recargue la página y envíe el formulario para que dispare un error. Sus campos de formulario deberían llenarse nuevamente.

**Nota:** La sección Referencia de Funciones (más abajo) contiene funciones que le permiten volver a llenar menús <select>, botones de radio y casillas de verificación.

**Nota Importante:** Si usa un array como nombre del campo de formulario, tiene que proporcionarlo como un array a la función. Ejemplo:

```
<input type="text" name="colors[]" value="<?php echo
set_value('colors[]'); ?>" size="50" />
```

Para mayor información, vea más abajo la sección Usar Arrays como Nombres de Campo.

### Callbacks: sus Propias Funciones de Validación

El sistema de validación soporta callbacks para sus propias funciones de validación. Esto le permite extender la clase de validación para ajustarla a sus necesidades. Por ejemplo, si necesita ejecutar una consulta de base de datos para ver si un usuario está eligiendo un nombre único, puede crear una

función callback que haga eso. Creemos un ejemplo para esto.

En su controlador, cambie la regla "username" por esta:

```
$this->form_validation->set_rules('username', 'Usuario',
'callback_username_check');
```

Luego agregue una nueva función llamada username_check a su controlador. Así es como se debería ver su controlador ahora:

```
<?php
class Form extends CI_Controller {
    function index()
    {
        $this->load->helper(array('form', 'url'));

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

        $this->form_validation->set_rules('username', 'Usuario',
'callback_username_check');
        $this->form_validation->set_rules('password', 'Contraseña',
'required');
        $this->form_validation->set_rules('passconf', 'Confirmar
Contraseña', 'required');
        $this->form_validation->set_rules('email', 'Email',
'required');

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

    function username_check($str)
    {
        if ($str == 'prueba')
        {
            $this->form_validation->set_message('username_check', 'El
campo %s no puede contener la palabra "prueba"');
            return FALSE;
        }
        else
        {
            return TRUE;
        }
    }
}
```

```
?>
```

Recargue su formulario y envíelo con la palabra "test" como nombre de usuario. Puede ver que el dato del campo de formulario se pasa a su función callback para procesarlo.

**Para invocar un callback tan solo ponga el nombre de la función en una regla, con "callback_" como prefijo de la regla.**

También puede procesar el dato del formulario que pasa a su callback y recuperarlo. Si su callback devuelve cualquier otra que cosa que un booleano TRUE/FALSE, se asume que esos datos son los recientemente procesados datos de formulario.

### Establecer Mensajes de Error

Todos los mensajes de error nativos se ubican en el siguiente archivo de idioma: language/english/form_validation_lang.php.

Para establecer sus propios mensajes personalizados, puede tanto editar el archivo, como usar la siguiente función:

```
$this->form_validation->set_message('regla', 'Mensaje de error');
```

Donde *regla* corresponde al nombre de una regla en particular y *Mensaje de error* es el texto que quiere mostrar.

Si incluye %s en su cadena de error, se reemplazará con el nombre "humano" que usó para el campo cuando estableció su regla.

En el ejemplo de "callback" anterior, el mensaje de error se estableció al pasar el nombre de la función:

```
$this->form_validation->set_message('username_check');
```

También puede anular cualquier mensaje de error que se encuentre en el archivo de idioma. Por ejemplo, para cambiar el mensaje para la regla "required", hará esto:

```
$this->form_validation->set_message('required', 'Su mensaje personalizado aquí');
```

### Traducir los Nombres de Campo

Si quisiera almacenar el nombre "humano" que le pasó a la función set_rules() en un archivo de idioma y, por lo tanto, hacer que el nombre sea traducible, aquí se muestra como:

Primero, prefije el nombre "humano" con lang:, como en el ejemplo:

```
$this->form_validation->set_rules('first_name', 'lang:first_name', 'required');
```

Luego, almacene el nombre en uno de sus array de archivos de idioma (sin el prefijo):

```
$lang['first_name'] = 'First Name';
```

**Nota:** Si almacena su ítem de array en un archivo de idioma que CI no carga automáticamente, necesitará recodar cargarlo en su controlador usando:

```
$this->lang->load('file_name');
```

Para mayor información acerca de los archivos de idioma, vea la página de la Clase Lang.

**Cambiar los Delimitadores de Error**

Por defecto, la Clase Form_validation agrega una etiqueta de párrafo (<p>) alrededor de cada mensaje que se muestra. Se puede cambiar esos delimitadores, tanto global como individualmente.

1. **Cambiar los Delimitadores Globalmente**

   Para cambiar globalmente los delimitadores de error en su función controlador, apenas después de cargar la Clase Form_validation, agregue esto:

   ```
   $this->form_validation->set_error_delimiters('<div
   class="error">', '</div>');
   ```

   En este ejemplo cambiamos los delimitadores a etiquetas div.

2. **Cambiar los Delimitadores Individualmente**

   A cada una de las dos funciones que generan errores que se muestran en este tutorial, se les puede proporcionar sus propios delimitadores según se muestra:

   ```
   <?php echo form_error('field name', '<div class="error">',
   '</div>'); ?>
   ```

   O:

   ```
   <?php echo validation_errors('<div class="error">', '</div>');
   ?>
   ```

**Mostrar los Errores Individualmente**

Si prefiere mostrar un mensaje de error cerca de cada campo de formulario en lugar de una lista, puede usar la función form_error().

Pruébelo! Cambie su formulario para que luzca así:

```
<h5>Usuario</h5>
<?php echo form_error('username'); ?>
<input type="text" name="username" value="<?php echo
set_value('username'); ?>" size="50" />

<h5>Contraseña</h5>
<?php echo form_error('password'); ?>
<input type="text" name="password" value="<?php echo
set_value('password'); ?>" size="50" />

<h5>Confirmar Contraseña</h5>
<?php echo form_error('passconf'); ?>
<input type="text" name="passconf" value="<?php echo
set_value('passconf'); ?>" size="50" />

<h5>Email</h5>
<?php echo form_error('email'); ?>
<input type="text" name="email" value="<?php echo set_value('email');
?>" size="50" />
```

Si no hay errores, no se muestra nada. Si hay un error, aparecerá un mensaje.

**Nota Importante:** Si usa un array como nombre del campo de formulario, tiene que proporcionarlo como un array a la función. Ejemplo:

```
<?php echo form_error('options[size]'); ?>
<input type="text" name="options[size]" value="<?php echo
set_value("options[size]"); ?>" size="50" />
```

Para mayor información, lea más abajo la sección Usar Arrays como Nombres de Campo.

### Guardar Conjuntos de Reglas de Validación en un Archivo de Configuración

Una funcionalidad interesante de la Clase Form_validation es que le permite almacenar todas sus reglas de validación para toda su aplicación en un archivo de configuración. Puede organizar estas reglas en "grupos". Estos grupos pueden cargarse sea en forma automática cuando se llama un controlador/función coincidente, o manualmente llamando a cada una según se necesite.

### Cómo Guardar sus Reglas

Para almacenar sus reglas de validación, simplemente cree un archivo llamado `form_validation.php` en su carpeta application/config/. En ese archivo colocará un array llamado `$config` con sus reglas. Como se mostró antes, el array de validación tendrá este prototipo:

```
$config = array(
               array(
                     'field'   => 'username',
                     'label'   => 'Usuario',
                     'rules'   => 'required'
                  ),
               array(
                     'field'   => 'password',
                     'label'   => 'Contraseña',
                     'rules'   => 'required'
                  ),
               array(
                     'field'   => 'passconf',
                     'label'   => 'Confirmar Contraseña',
                     'rules'   => 'required'
                  ),
               array(
                     'field'   => 'email',
                     'label'   => 'Email',
                     'rules'   => 'required'
                  )
            );
```

Su regla de validación se cargará automáticamente y se usará cuando llame a la función run().

Por favor advierta que al array **TIENE** que llamarse $config.

**Crear un Conjunto de Reglas**

Para organizar sus reglas en "conjuntos" se necesita que las coloque en "sub arrays". Considere el siguiente ejemplo, que muestra dos conjuntos de reglas. Llamamos arbitrariamente a esas dos reglas "signup" y "email". Puede darles el nombre que guste:

```
$config = array(
                  'signup' => array(
                                        array(
                                                'field' => 'username',
                                                'label' => 'Username',
                                                'rules' => 'required'
                                        ),
                                        array(
                                                'field' => 'password',
                                                'label' => 'Password',
                                                'rules' => 'required'
                                        ),
                                        array(
                                                'field' => 'passconf',
                                                'label' =>
'PasswordConfirmation',
                                                'rules' => 'required'
                                        ),
                                        array(
                                                'field' => 'email',
                                                'label' => 'Email',
                                                'rules' => 'required'
                                        )
                                        ),
                  'email' => array(
                                        array(
                                                'field' =>
'emailaddress',
                                                'label' =>
'EmailAddress',
                                                'rules' =>
'required|valid_email'
                                        ),
                                        array(
                                                'field' => 'name',
                                                'label' => 'Name',
                                                'rules' =>
'required|alpha'
                                        ),
                                        array(
                                                'field' => 'title',
                                                'label' => 'Title',
                                                'rules' => 'required'
                                        ),
```

```
                                        array(
                                                'field' => 'message',
                                                'label' => 'MessageBody',
                                                'rules' => 'required'
                                            )
                                        )
                );
```

**Llamar a un Grupo de Reglas Específico**

Para llamar a un grupo específico, deberá pasar su nombre a la función `run()`. Por ejemplo, para llamar a la regla `signup`, hará esto:

```
if ($this->form_validation->run('signup') == FALSE)
{
   $this->load->view('mi_form');
}
else
{
   $this->load->view('form_success');
}
```

**Asociar una Función Controlador con un Grupo de Reglas**

Un método alternativo (y más automático) de llamar a un grupo de reglas es darle nombre de acuerdo a la clase/función controlador que piense usar con él. Por ejemplo, digamos que tiene un controlador llamado `Member` y una función llamada `signup`. Así es como la clase se podría ver:

```
<?php

class Member extends CI_Controller {

   function signup()
   {
      $this->load->library('form_validation');

      if ($this->form_validation->run() == FALSE)
      {
         $this->load->view('mi_form');
      }
      else
      {
         $this->load->view('form_success');
      }
   }
}
?>
```

En su archivo de configuración de validación, le dará nombre a su grupo de reglas `member/signup`:

```
$config = array(
```

```
            'member/signup' => array(
                                    array(
                                            'field' => 'username',
                                            'label' => 'Usuario',
                                            'rules' => 'required'
                                        ),
                                    array(
                                            'field' => 'password',
                                            'label' => 'Contraseña',
                                            'rules' => 'required'
                                        ),
                                    array(
                                            'field' => 'passconf',
                                            'label' => 'Confirmar
Contraseña',
                                            'rules' => 'required'
                                        ),
                                    array(
                                            'field' => 'email',
                                            'label' => 'Email',
                                            'rules' => 'required'
                                        )
                                    )
                );
```

Cuando un grupo de reglas se llama igual que una clase/función controlador, se usará automáticamente cuando sea invocada la función run() function desde esa clase/función.

**Usar Arrays como Nombres de Campo**

La Clase Form_validation soporta el uso de arrays como nombres de campo. Considere este ejemplo:

```
<input type="text" name="options[]" value="" size="50" />
```

Si usa un array como nombre de campo, tiene que usar el **MISMO** nombre de array en las Funciones Helper que necesitan nombre de campo y como su nombre de campo Regla de Validación.

Por ejemplo, para establecer una regla para el campo anterior, usaría:

```
$this->form_validation->set_rules('options[]', 'Options',
'required');
```

O, para mostrar un error para el campo anterior, usaría:

```
<?php echo form_error('options[]'); ?>
```

O para volver a llenar el campo, usaría:

```
<input type="text" name="options[]" value="<?php echo
set_value('options[]'); ?>" size="50" />
```

También puede usar arrays multidimensionales como nombres de campo. Por ejemplo:

```
<input type="text" name="options[size]" value="" size="50" />
```

O aún:

```
<input type="text" name="sports[nba][basketball]" value="" size="50"
/>
```

Como con nuestro primer ejemplo, tiene que usar el mismo nombre en las funciones helper:

```
<?php echo form_error('sports[nba][basketball]'); ?>
```

Si está usando casillas de verificación (u otro campo) que tiene opciones múltiples, no se olvide de dejar un corchete vacío después de cada opción, para que todas las selecciones sean agregadas al array POST:

```
<input type="checkbox" name="options[]" value="rojo" />
<input type="checkbox" name="options[]" value="azul" />
<input type="checkbox" name="options[]" value="verde" />
```

Or if you use a multidimensional array:

```
<input type="checkbox" name="options[color][]" value="rojo" />
<input type="checkbox" name="options[color][]" value="azul" />
<input type="checkbox" name="options[color][]" value="verde" />
```

También incluirá corchetes cuando use una función helper:

```
<?php echo form_error('options[color][]'); ?>
```

**Referencia de Reglas**

La siguiente es una lista de todas las reglas nativas que están disponibles para usarse:

| Regla | Parámetro | Descripción | Ejemplo |
|---|---|---|---|
| **required** | No | Devuelve FALSE si el elemento de formulario está vacío. | |
| **is_unique** | Si | Devuelve FALSE si el elemento del formulario ya existe en la tabla y el nombre del campo en el parámetro. | is_unique[table.field] |
| **matches** | Si | Devuelve FALSE si el elemento de formulario no coincide con el parámetro. | matches[form_item] |
| **min_length** | Si | Devuelve FALSE si el elemento de formulario es más corto que el valor del parámetro. | min_length[6] |
| **max_length** | Si | Devuelve FALSE si el elemento de formulario es más largo que el valor del parámetro. | max_length[12] |
| **exact_length** | Si | Devuelve FALSE si el elemento de formulario no es exactamente el valor del parámetro. | exact_length[8] |
| **greater_than** | Si | Devuelve FALSE si el elemento de formulario es menor que el valor del parámetro o no es numérico. | greater_than[8] |
| **less_than** | Si | Devuelve FALSE si el elemento de formulario es mayor que el valor del parámetro o no es numérico. | less_than[8] |
| **alpha** | No | Devuelve FALSE si el elemento de formulario contiene algo que no es un caracter alfabético. | |
| **alpha_numeric** | No | Devuelve FALSE si el elemento de formulario contiene algo que no es un caracter alfanumérico. | |
| **alpha_dash** | No | Devuelve FALSE si el elemento de formulario contiene algo que no es un caracter alfanumérico, guión de subrayado o guión común. | |

| | | | |
|---|---|---|---|
| **numeric** | No | Devuelve FALSE si el elemento de formulario contiene algo que no es un caracter numérico. | |
| **integer** | No | Devuelve FALSE si el elemento de formulario contiene algo que no es un entero. | |
| **decimal** | Si | Devuelve FALSE si el elemento de formulario no es exactamente el valor del parámetro. | |
| **is_natural** | No | Devuelve FALSE si el elemento de formulario contiene algo que no es un número natural: 0, 1, 2, 3, etc. | |
| **is_natural_no_zero** | No | Devuelve FALSE si el elemento de formulario contiene algo que no es un número natural, pero no cero: 1, 2, 3, etc. | |
| **valid_email** | No | Devuelve FALSE si el elemento de formulario no contiene una dirección de email válida. | |
| **valid_emails** | No | Devuelve FALSE si cualquier valor provisto en una lista separada por comas no es un email válido. | |
| **valid_ip** | No | Devuelve FALSE la IP proporcionada no es válida. Acepta un parámetro opcional de "IPv4" o "IPv6" para especificar un formato IP | |
| **valid_base64** | No | Devuelve FALSE si la cadena proporcionada contiene algo que no es un caracter Base64. | |
| **regex_match** | Si | Devuelve FALSE si el elmento de formulario contiene algo que no este en la expresión regular. | regex_match[/expr_reg/] |

**Nota:** Estas reglas también se pueden llamar desde funciones discretas. Por ejemplo:

```
$this->form_validation->required($string);
```

**Nota:** También puede usar funciones nativas de PHP que permiten un solo parámetro.

**Referencia de Preparaciones**

La siguiente es una lista de todas las funciones de preparación que están disponibles para usarse:

| Nombre | Parámetro | Descripción |
|---|---|---|
| **xss_clean** | No | Ejecuta los datos mediante la función de Filtrado XSS, descripta en la página Clase Input. |
| **prep_for_form** | No | Convierte los caracteres especiales para que los datos HTML se puedan mostrar en un campo de formulario sin romperlo. |
| **prep_url** | No | Agrega "http://" a las URLs si falta. |
| **strip_image_tags** | No | Quita el HTML de las etiquetas de imagen, dejando la URL en crudo. |
| **encode_php_tags** | No | Convierte las etiquetas PHP a entidades. |

**Nota:** También puede usar funciones nativas de PHP que permiten un parámetro, como `trim`, `htmlspecialchars`, `urldecode`, etc.

**Referencia de Funciones**

Las siguientes funciones están pensadas para usarse en sus funciones controladoras.

**$this->form_validation->set_rules();**

Le permite establecer reglas de validación, como se describe en las secciones anteriores del tutorial:

- Establecer Reglas de Validación
- Guardar Grupos de Reglas de Validación en un Archivo de Configuración

**$this->form_validation->run();**

Ejecuta las rutinas de validación. Devuelve el booleano **TRUE** en caso de éxito y **FALSE** en caso de falla. Opcionalmente, puede pasar el nombre del grupo de validación mediante la función, como se describe en: Guardar Grupos de Reglas de Validación en un Archivo de Configuración..

**$this->form_validation->set_message();**

Le permite establecer mensajes de error. Vea más arriba Establecer Mensajes de Error.

**Referencia de Helpers**

Las siguientes funciones helper están disponibles para usarse en los archivos de vistas que contienen sus formularios. Advierta que éstos son funciones procedurales, por lo tanto **no** necesitan que les anteponga $this->form_validation.

**form_error()**

Muestra un mensaje de error individual asociado con el nombre del campo suministrado a la función. Ejemplo:

```
<?php echo form_error('username'); ?>
```

Se pueden especificar opcionalmente los delimitadores de error. Vea más arriba la sección Cambiar los Delimitadores de Error.

**validation_errors()**

Muestra todos los mensajes de error como una cadena. Ejemplo:

```
<?php echo validation_errors(); ?>
```

Se pueden especificar opcinalmente los delimitadores de error. Vea más arriba la sección Cambiar los Delimitadores de Error.

**set_value()**

Le permite establecer un valor de un campo de texto o un textarea. Tiene que proporcionar el nombre del campo mediante el primer parámetro de la función. El segundo prámetro (opcional) le permite establecer un valor por defecto para el formulario. Ejemplo:

```
<input type="text" name="quantity" value="<?php echo set_value('quantity', '0'); ?>" size="50" />
```

El formulario anterior mostrará "0" cuando se cargue por primera vez.

**set_select()**

Si usa un menú <select>, esta función le permite mostrar el ítem de menú que se seleccionó. El primer parámetro tiene que contener el nombre del menú select, el segundo parámetro tiene que contener el valor de cada ítem y el tercer parámetro (opcional) le permite establecer un ítem como valor por defecto (usar booleano TRUE/FALSE). Ejemplo:

Example:

```
<select name="myselect">
<option value="one" <?php echo set_select('mi_select', 'uno', TRUE); ?> >One</option>
<option value="two" <?php echo set_select('mi_select', 'dos'); ?> >Two</option>
<option value="three" <?php echo set_select('mi_select', 'tres'); ?> >Three</option>
</select>
```

**set_checkbox()**

Le permite mostrar una casilla de verificación en el estado en que se envió. El primer parámetro tiene que contener el nombre de la casilla de verificación, el segundo parámetro tiene que contener su valor y el tercer parámetro (opcional) le permite establecer un ítem como valor por defecto (usar booleano TRUE/FALSE). Ejemplo:

```
<input type="checkbox" name="mi_check[]" value="1" <?php echo
set_checkbox('mycheck[]', '1'); ?> />
<input type="checkbox" name="mi_check[]" value="2" <?php echo
set_checkbox('mycheck[]', '2'); ?> />
```

**set_radio()**

Le permite mostrar botones de radio en el estado en que fueron enviados. Esta función es idéntica a la función **set_checkbox()** anterior.

```
<input type="radio" name="mi_radio" value="1" <?php echo set_radio('myradio',
'1', TRUE); ?> />
<input type="radio" name="mi_radio" value="2" <?php echo set_radio('myradio',
'2'); ?> />
```

**Clase Input**

La Clase de Entrada sirve para dos propósitos:

1. Pre-procesa datos de entrada global por seguridad.
2. Provee alguna funciones auxiliares para recuperar datos de entradas y pre-procesarlos.

**Nota:** Esta clase es inicializada automáticamente por el sistema, así que no hay necesidad de que lo haga manualmente.

**Filtros de Seguridad**

La función de filtro de seguridad es llamada automáticamente cuando un nuevo controlador es invocado. Hace lo siguiente:

- Destruye todo el arreglo global GET. Ya que CodeIgniter no utiliza cadenas GET, no hay razón para permitirla.
- Destruye todas las variables globales en el caso de que register_globals esté habilitado.
- Filtra las claves de POST/COOKIE, permitiendo sólo caracteres alfanuméricos (y algunos pocos más).
- Provee filtro XSS (Cross-site Scripting Hacks). Esto puede ser habilitado globalmente, o a pedido.
- Estandariza las caracteres de nueva línea a \n

**Filtro XSS**

La Clase Input tiene la habilidad de filtrar la entrada automáticamente para evitar ataques cross-site scripting. Si desea que el filtrado se ejecute automáticamente cada vez que encuentra datos POST o COOKIE, puede habilitarlo abriendo su archivo `application/config/config.php` y configurando esto:

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

Por favor, referirse a la documentación de la Clase Security para mayor información sobre el Filtrado XSS en su aplicación.

**Usar Datos POST, COOKIE, or SERVER**

CodeIgniter viene con tres funciones asistentes que le permiter recuperar ítems POST, COOKIE or SERVER. La principal ventaja de usar las funciones provistas en vez de traer el ítem directamente ($_POST['algo']) es que la función verificará si el ítem existe y devuelve false (buleano) si no. Esto le permite convenientemente usar datos sin tener que probar si un ítem existe primero. En otras palabras, normalmente puede hacer algo así:

```
if ( ! isset($_POST['algo']))
{
    $algo = FALSE;
}
else
{
    $algo = $_POST['algo'];
}
```

Con las funciones de CodeIgniter puede simplemente hacer esto:

```
$algo = $this->input->post('algo');
```

Las tres funciones son:

- $this->input->post()
- $this->input->cookie()
- $this->input->server()

**$this->input->post()**

El primer parámetro contendrá el nombre del ítem POST que está buscando:

```
$this->input->post('algun_dato');
```

La función devuelve FALSE (buleano) si el ítem que intenta recuperar no existe.

El segundo opcional parámetro le permite correr los datos a través del filtro XSS. Es habilitada al establecer el segundo parámetro como un booleano TRUE;

```
$this->input->post('algun_dato', TRUE);
```

Para devolver un array de todos los ítems POST, llamarla sin parámetros.

Para devolver todos los ítems POST y pasarlos a través del filtro XSS, deje el primer parámetro en blanco y establezca el segundo parámetro a un booleano.

La función devuelve FALSE (booleano) si no hay ítems en el POST.

```
$this->input->post(); // devuelve todos los ítems POST con Filtrado
XSS
$this->input->post(NULL, FALSE); // devuelve todos los ítems POST sin
Filtrado XSS
```

**$this->input->get()**

Esta función es idéntica a la función post, solo que recupera datos GET:

```
$this->input->get('algun_dato', TRUE);
```

Para devolver un array de todos los ítems, llamarla sin parámetros.

Para devolver todos los ítems GET y pasarlos a través del filtro XSS, deje el primer parámetro en blanco y establezca el segundo parámetro al booleano TRUE.

La función devuelve FALSE (booleano) si no hay ítems en el GET.

```
$this->input->get(); // devuelve todos los ítems GET con Filtrado XSS
$this->input->get(NULL, FALSE); // devuelve todos los ítems items sin
Filtrado XSS
```

**$this->input->get_post()**

Esta función buscará datos a traves de los flujos post y get, primero en post y luego en get:

```
$this->input->get_post('algun_dato', TRUE);
```

**$this->input->cookie()**

Esta función es idéntica a la función post, sólo que recupera datos de cookie:

```
$this->input->cookie('algun_dato', TRUE);
```

**$this->input->server()**

Esta función es idéntica a las funciones anteriores, sólo que recupera datos del servidor:

```
$this->input->server('algun_dato');
```

**$this->input->set_cookie()**

Establece una cookie conteniendo los valores que especifique. Hay dos formas de pasarle información a esta función: Método de Array y Parámetros Discretos.

**Método de Array**

Al usar este método, se pasa un array asociativo al primer parámetro:

```
$cookie = array(
    'name'   => 'Nombre de la Cookie',
    'value'  => 'Valor de la Cookie',
    'expire' => '86500',
    'domain' => '.algun-dominio.com',
    'path'   => '/',
    'prefix' => 'mi_prefijo_',
    'secure' => TRUE
);

$this->input->set_cookie($cookie);
```

**Notas:**

• Solamente el nombre y el valor son obligatorios. Para borrar una cookie establezca la caducidad ('expire') en blanco.

• La caducidad se establece en **segundos**, que se agregarán a la hora actual. No incluya la hora, sino solamente la cantidad de segundos desde *ahora* en los que desea que la cookie sea válida. Si la caducidad se establece a cero, la cookie durará solamente el tiempo en el que el navegador esté abierto.

• Para las cookies de todo el sitio, independientemente de cómo sea solicitado su sitio, agregar su URL al **dominio** comenzando con un punto, así: .su-dominio.com

• Normalmente la ruta no es necesaria porque la función establece una ruta de servidor.

• Solamente se necesita el prefijo si necesita evitar colisiones de nombres con otras cookies que se llaman igual en su servidor.

• Solamente se necesita el booleano 'secure' si quiere hacer una cookie segura, estableciéndolo a TRUE.

**Parámetros Discretos**

Si lo prefiere, puede establecer la cookie pasándole datos al usar parámetros individuales:

```
$this->input->set_cookie($name, $value, $expire, $domain, $path,
$prefix, $secure);
```

**$this->input->cookie()**

Le permite recuperar una cookie. El primer parámetro contendrá el nombre de la cookie que está buscando (incluyendo cualquier prefijo):

```
cookie('alguna_cookie');
```

La función devuelve FALSE (booleano) si el ítem que está intentando recuperar no existe.

El segundo parámetro (opcional) le permite pasar los datos a través del filtro XSS. Se habilita estableciendo el segundo parámetro al booleano TRUE.

```
cookie('alguna_cookie', TRUE);
```

**$this->input->ip_address()**

Devuelve la dirección IP del usuario actual. Si la dirección IP no es válida, la función devolverá la IP: 0.0.0.0.

```
echo $this->input->ip_address();
```

**$this->input->valid_ip(*$ip*)**

Toma como entrada una dirección IP y devuelve TRUE o FALSE (booleano) según sea válida o no. Nota: La función $this->input->ip_address() anterior valida la IP automáticamente.

```
if ( ! $this->input->valid_ip($ip))
{
     echo 'Inválida';
}
else
{
     echo 'Válida';
}
```

Acepta un segundo parámetro opcional de cadena "IPv4" o "IPv6" para especificar un formato de IP. El valor por defecto controla ambos formatos.

**$this->input->user_agent()**

Devuelve el agente de usuario (navegador web) que usa el usuario actual. Devuelve FALSE si no está disponible.

```
echo $this->input->user_agent()
```

Vea la Clase User_agent para obtener información de cómo extraer de la cadena del agente de usuario.

**$this->input->request_headers()**

Útil si está corriendo en un entorno que no es Apache donde apache_request_headers() no está soportado.

```
$headers = $this->input->request_headers();
```

**$this->input->get_request_header();**

Devuelve un miembro simple del array de encabezados de solicitud.

```
$this->input->get_request_header('algun-encabezado', TRUE);
```

**$this->input->is_ajax_request()**

Comprueba si el encabezado de servidor *HTTP_X_REQUESTED_WITH* está establecido y devuelve una respuesta booleana.

**$this->input->is_cli_request()**

Comprueba si la constante *STDIN* está establecida, que es una forma segura de probar si PHP se está ejecutando desde la línea de comandos.

```
$this->input->is_cli_request()
```

**Clase FTP**

La Clase FTP de CodeIgniter permite que los archivos sean transferidos a un servidor remoto. Los archivos remotos pueden también ser movidos, renombrados, y borrados. La Clase FTP también incluye una función de "espejado" que permite que permite a todo un directorio local ser recreado en forma remota a través de FTP.

**Nota:** Los protocolos FTP SFTP y SSL no están soportados, solo FTP estándar.

**Inicializando la Clase**

Al igual que la mayoría de las otras clases en CodeIgniter, la clase FTP se inicializa en sus controlador usando la función $this->load->library:

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

Una vez cargado, el objeto FTP estará disponible utilizando: $this->ftp

**Ejemplos de Uso**

En este ejemplo, una conexión se abre con el servidor FTP, y un archivo local es leído y cargado en modo ASCII.Los permisos de archivo se establecen a 755. Nota: La configuración de permisos requiere 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();
```

En este ejemplo se recupera una lista de archivos desde el servidor.

```
$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();
```

En este ejemplo, se refleja un directorio local en el servidor.

```
$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();
```

**Referencia de Función**

**$this->ftp->connect()**

Se conecta y registra en el servidor FTP. Las preferencias de conexión se establecen pasando un arreglo a la función, o almacenándolas en un archivo de configuración.

Aquí hay un ejemplo que muestra cómo establecer las preferencias manualmente:

```
$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);
```

**Establacer las Preferencias de FTP en un Archivo de Configuracion**

Si Ud. prefiere, puede almacenar sus preferencias de FTP en un archivo de configuración. Simplemente crea un nuevo archivo llamado *ftp.php*, agrega el arreglo *$config* en ese archivo. A continuación guarde el archivo en *config/ftp.php* y se usará automáticamente.

**Opciones de Conexión Disponibles:**

- **hostname** - el nombre del host FTP. Por lo general, algo como:  ftp.example.com
- **username** - el nombre de usuario FTP.
- **password** - la contraseña FTP.
- **port** - el número de puerto. Establecido a 21 por defecto.
- **debug** - `TRUE/FALSE` (boolean). Ya sea para permitir o no la depuración para mostrar

mensajes de error.

- **passive** - `TRUE/FALSE` (boolean). Ya sea para usar o no el modo pasivo. Pasivo se establece automáticamente por defecto.

**$this->ftp->upload()**

Carga un archivo a su servidor. Usted debe proporcionar la ruta local y la ruta remota, opcionalmente se puede establecer el modo y los permisos. Ejemplo:

```
$this->ftp->upload('/local/path/to/myfile.html',
'/public_html/myfile.html', 'ascii', 0775);
```

**Las opciones de Modo son:** `ascii`, `binary`, and `auto` (el valor por defecto). Si `auto` es usado, será el modo base en la extensión de archivos del archivo fuente.

Los permisos están disponibles si está corriendo PHP 5 y se pueden pasar como un valor `octal` en el cuarto parámetro.

**$this->ftp->download()**

Descarga un archivo desde su servidor. Tiene que proporcionar las rutas remota y local. Opcionalmente puede establecer el modo. Ejemplo:

```
$this->ftp->download('/public_html/myfile.html',
'/local/path/to/myfile.html', 'ascii');
```

**Las opciones de modo son:**`ascii`, `binary`, and `auto` (valor por defecto). Si se usa `auto`, basará el modo en la extensión del archivo origen.

Devuelve FALSE si la descarga no se realiza satisfactoriamente incluyendo si PHP no tiene permiso para escribir el archivo local).

**$this->ftp->rename()**

Permite renombrar un archivo. Suministre el nombre/ruta del archivo fuente y el nombre/ruta del nuevo archivo.

```
// Renombra green.html a blue.html
$this->ftp->rename('/public_html/foo/green.html',
'/public_html/foo/blue.html');
```

**$this->ftp->move()**

Le permite mover un archivo. Suministre la ruta fuente y la de destino:

```
// Mueve blog.html desde "joe" a "fred"
$this->ftp->move('/public_html/joe/blog.html',
'/public_html/fred/blog.html');
```

Nota: si el nombre del archivo destino es diferente el archivo será renombrado.

**$this->ftp->delete_file()**

Le permite borrar un archivo. Suministre la ruta fuente con el nombre del archivo.

```
$this->ftp->delete_file('/public_html/joe/blog.html');
```

**$this->ftp->delete_dir()**

Le permite borrar un directorio y todo lo que contiene. Suministre la ruta fuente al directorio con una barra diagonal.

**Importante** Tenga MUCHO CUIDADO con esta función. Ésta borrará recursivamente **todo** en la ruta suministrada, incluidas las sub-carpetas y todos los archivos. Hacer uso de la misma estando absolutamente seguro de que la ruta es correcta. Trate de usar la función `list_files()` en primer lugar para verificar que su ruta es correcta.

```
$this->ftp->delete_dir('/public_html/path/to/folder/');
```

**$this->ftp->list_files()**

Le permite obtener una lista de archivos de su servidor retornados como un arreglo. Usted debe proporcionar la ruta al directorio deseado.

```
$list = $this->ftp->list_files('/public_html/');

print_r($list);
```

**$this->ftp->mirror()**

Lee recursivamente una carpeta local y todo lo que contiene (incluidos las sub-carpetas) y crea un espejo a través de FTP basado en él. Cualquiera que sea la estructura de directorios de la ruta del archivo original será recreado en el servidor. Ud. debe suministrar una ruta fuente y una ruta destino:

```
$this->ftp->mirror('/path/to/myfolder/', '/public_html/myfolder/');
```

**$this->ftp->mkdir()**

Le permite crear un directorio en su servidor. Suministre la ruta final en el nombre de la carpetaque desea crear, con una barra diagonal. Los permisos pueden ser establecidos pasando un valor `octal` en el segundo parámetro (si está ejecutando PHP 5).

```
// Crea una carpeta llamada "bar"
$this->ftp->mkdir('/public_html/foo/bar/', 0777);
```

**$this->ftp->chmod()**

Le permite establecer permisos de archivo. Suministre la ruta del archivo o carpeta que desea modificar los permisos en:

```
// Chmod "bar" a 777
$this->ftp->chmod('/public_html/foo/bar/', 0777);
```

**$this->ftp->close();**

Cierra la conexión con el servidor. Es recomendable que utilice esto cuando haya terminado de cargar.

**Clase Pagination**

La Clase Paginación de CodeIgniter es muy fácil de usar, y es 100% personalizable, ya sea dinámicamente o a través de las preferencias almacenadas.

Si no está familiarizado con el término "paginación", se refiere a los vínculos que le permite navegar de una página a otra, como este:

```
« First  < 1 2 3 4 5 >  Last »
```

**Ejemplo**

Este es un ejemplo sencillo que muestra cómo crear una de paginación en sus funciones del controller:

```
$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();
```

**Notas:**

El arreglo *$config* contiene sus variables de configuración. estas son pasadas a la función $this->pagination->initialize como se muestra arriba. A pesar de que hay una veintena de items que usted puede configurar, el mínimo que necesita son los tres mostrados. Aquí hay una descripción de lo que representan esos items:

- **base_url** Esta es la URL completa a la clase/función del controlador que contiene su número de páginas. En el ejemplo anterior, ésta apunta a un controlador denominado "Test" y a una función llamada "page". Tenga en cuenta que Ud. puede modificar el trazado de su URI si necesita una estructura diferente.
- **total_rows** Este número representa la cantidad total de filas en el conjunto de resultados que está creando para el número de páginas. Normalmente, este será el número total de filas que retorno su consulta a la base de datos.
- **per_page** El número de elementos que tiene intención de mostrar por página. En el ejemplo anterior, Ud. desea mostrar 20 elementos por página.

La funcición *create_links()* retorna una cadena vacía cuando no hay paginación para mostrar.

**Establecer preferencias en un archivo de configuración**

Si prefiere no definir las preferencias utilizando el método anterior, puede ponerlas en un archivo de configuración. Basta con crear un nuevo archivo llamado *pagination.php*, y agregar el arreglo *$config* en ese archivo. A continuación guarde el archivo en: *config/pagination.php* y será utilizado automáticamente. Ud. No necesita usar la función $this->pagination->initialize si guarda sus preferencias en un archivo de configuración.

**Personalizando la Paginación**

La siguiente es una lista de todas las preferencias que se pueden pasar a la función de inicialización para ajustar la pantalla.

**$config['uri_segment'] = 3;**

La función determina automáticamente cual segmento de su URI contiene el número de página. Si usted necesita algo diferente se puede especificar.

**$config['num_links'] = 2;**

El número de links en "digitos" que Ud. desea antes y después del número de página seleccionado. Por ejemplo, el número 2 indica que habrá dos dígitos en ambos lados, como en el ejemplo de enlaces en la parte superior de esta página.

**Agregando Marcas de Cierre**

Si desea rodear toda la paginación con algunas marcas lo puede hacer con estas dos preferencias:

**$config['full_tag_open'] = ' ';**

La etiqueta de apertura situada en la parte izquierda de todo el resultado.

**$config['full_tag_close'] = ' ';**

La etiqueta de cierre situada en la parte derecha de todo el resultado.

**Personalizando el enlace "Primero"**

**$config['first_link'] = 'Primero';**

El texto que le gustaría que se muestre en el "primer" enlace de la izquierda.

**$config['first_tag_open'] = ' ';**

La etiqueta de apertura para el enlace "first".

**$config['first_tag_close'] = ' ';**

La etiqueta de cierre para el enlace "first".

**Personalizando el enlace "Último"**

**$config['last_link'] = 'Último';**

El texto que le gustaría que se muestre en el "último" enlace de la derecha.

**$config['last_tag_open'] = ' ';**

La etiqueta de apertura para el enlace "last".

**$config['last_tag_close'] = ' ';**

La etiqueta de cierre para el enlace "last".

**Personalizando el enlace "Siguiente"**

**$config['next_link'] = '>';**

El texto que le gustaría que se muestre en el enlace de página "siguiente".

**$config['next_tag_open'] = ' ';**

La etiqueta de apertura para el enlace "next".

**$config['next_tag_close'] = ' ';**

La etiqueta de cierre para el enlace "next".

**Personalizando el enlace "Anterior"**

**$config['prev_link'] = '<';**

El texto que le gustaría que se muestre en el enlace de página "anterior".

**$config['prev_tag_open'] = ' ';**

La etiqueta de apertura para el enlace "previous".

**$config['prev_tag_close'] = ' ';**

La etiqueta de cierre para el enlace "previous".

**Personalizando el enlace "Página Actual"**

**$config['cur_tag_open'] = '';**

La etiqueta de apertura para el enlace "current".

**$config['cur_tag_close'] = '';**

La etiqueta de cierre para el enlace "current".

**Personalizando el enlace "Dígito"**

**$config['num_tag_open'] = ' ';**

La etiqueta de apertura para el enlace "digit".

**$config['num_tag_close'] = ' ';**

La etiqueta de cierre para el enlace "digit".

### Ocultar las Páginas

Si quiere que no se listen páginas específicas (por ejemplo, quiere solamente los enlaces "siguiente" y "anterior"), puede suprimir su presentación al agregar:

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

### Agregar una Clase a cada Ancla

Si quiere agregar un atributo de clase a cada enlace presentado por la Clase Pagination, puede establecer el índice "anchor_class" del array config igual al nombre de clase que desea.

**Clase Session**

La Clase Session le permite mantener el "estado" de un usuario y seguir su actividad mientras navegue su sitio. La clase de Sesión guarda información de sesión para cada usuario como datos serializados (y opcionalmente encriptados) en una cookie. También guarda los datos de sesión en una tabla de base de datos para seguridad adicional, ya que permite que el ID de la sesión en la cookie del usuario sea contrastada contra la guardada. Por defecto sólo la cookie es guardada. Si elije usar la opción de base de datos, necesitará crear la tabla de sesión como se indica debajo.

**Nota:** La Clase Session **NO** utiliza las sesiones nativas de PHP. Genera sus propios datos de sesión, ofreciendo más flexibilidad para los desarrolladores.

**Nota:** Aún cuando no use sesiones encriptadas, tiene que establecer una clave de encriptación en su archivo de configuración, que se usa para ayudar a evitar la manipulación de los datos de sesión.

**Inicializando una Sesión**

Las sesiones típicamente correrán globalmente con cada carga de la página, así que la clase de sesión debe ser o inizializada en el constructor de su controlador, o puede ser auto-cargada por el sistema. Para la mayor parte, la clase de sesión correrá desatendida en el transfondo, así que simplemente inicializar la clase causará leer, crear y actualizar sesiones.

Para inicializar la clase de Sesión manualmente en el constructor del controlador, use la función $this->load->library:

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

Una vez cargada, el objeto de la librería de Sesión estará disponible usando: $this->session

**¿Cómo funcionan las sesiones?**

Cuando una página es cargada, la clase de sesión verificará si datos de sesión válidos existen en la cookie de sesión del cliente. Si los datos de sesión **no** existen (o si ha expirado) una nueva sesión será creada y guardada en la cookie. Si una sesión existe, su información será actualizada y la cookie será actualizada. Con cada actualización, el session_id (identificador de sesión) será regenerado.

Es importante para usted entender que una vez inicializada, la clase de Sesión corre automáticamente. No hay nada que necesite hacer para que el comportamiento anterior ocurre. Puede, como verá luego, trabajar con datos de sesión o incluso agregar sus propios datos a la sesión del usuario, pero el proceso de lectura, escritura y actualización de sesión es automático.

**Que son los Datos de Sesión?**

Una *sesión*, hasta donde le concierne a CodeIgniter, es simplemente un arreglo conteniendo la siguiente información:

- El identificador de sesión único (esto es estadísticamente una cadena aleatoria con muy fuerte entropia, una encriptación con MD5 por portabilidad, y es regenerado (por defecto) cada cinco minutos)
- La dirección IP del usuario
- Los datos del Agente del Usuario (los primeros 50 caracteres de la cadena de datos del explorador)

- La marca de tiempo de la "última actividad".

Los datos anteriores son guardados en una cookie como un arreglo serializado con este prototipo:

```
[array]
(
     'session_id'    => encriptación aleatoria,
     'ip_address'    => 'cadena - dirección IP del usuario',
     'user_agent'    => 'cadena - datos del agente del usuario',
     'last_activity' => marca de tiempo
)
```

Si tiene la opción de encriptación habilitada, el arreglo será encriptado antes de ser guardado en la cookie, haciendo los datos altamente seguros e impermeables de ser leídos o alterados por alguien. Más información acerca de la encriptación puede ser encontrada aquí, aunque la clase de Sesión se ocupará de la inicialización y encriptación de datos automáticamente.

Nota: Las cookies de sesión son sólo actualizadas cada cinco minutos por defecto para reducir la carga de procesamiento. Si recarga repetidamente una página, notará que el tiempo de "last activity" sólo se actualizará si cinco munitos o más han pasado desde la última vez que la cookie fue escrita. Este tiempo es configurable cambiando la línea de $config['time_to_update'] en su archivo system/config/config.php .

**Recuperando Datos de Sesión**

Cualquier pieza de información desde el arreglo de sesión está disponible usando la siguiente función:

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

Donde `item` es la clave del arreglo correspondiente al item que desea traer. Por ejemplo, para traer el ID de sesión hará lo siguiente:

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

**Nota:** La función devuelve FALSE (booleano) si el item al que intenta acceder no existe.

**Agregando Datos de Sesión Especiales**

Un aspecto útil del arreglo de sesión es que puede agregar sus propios datos a él y ellos serán guardados en la cookie del usuario. ¿Por qué querría hacer esto? Aquí hay un ejemplo:

Digamos que un usuario en particular ingresó a su sitio. Una vez autenticado, puede agregar su nombre de usuario y dirección de email a la cookie de sesión, haciendo ese dato globalmente disponible sin tener que correr una consulta de base de datos cuando lo necesite.

Para agregar sus datos al arreglo de sesión debe pasar un arreglo que contenga sus nuevos datos a esta función:

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

Donde `$arreglo` es un arreglo asociativo conteniendo sus nuevos datos. Aquí hay un ejemplo:

```
$nuevosdatos = array(
                   'nombre_de_usuario'  => 'johndoe',
                   'email'     => 'johndoe@algun-sitio.com',
                   'ingresado' => TRUE
```

```
                );

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

Si quiere agregar un dato a la vez, set_userdata() también soporta esta sintaxis.

```
$this->session->set_userdata('algun_nombre', 'algun_valor');
```

**Nota:** Las cookies pueden guardar sólo 4KB de datos, así que sea cuidadoso de no exceder la capacidad. La El proceso de encriptación en particular produce una cadena de datos más largo que la original, así que sea cuidadoso siguiente cuanto datos están guardando.

**Removiendo Datos de Sesión**

Simplemente como set_userdata() puede ser usado para agregar información en la sesión, unset_userdata() puede ser usado para remover, al pasar la clase de sesión. Por ejemplo, si quiere remover 'algun_nombre' de la información de sesión:

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

Esta función puede también puede recibir un arreglo asociativo de items para borrar.

```
$arreglo_items = array('nombre_de_usuario' => '', 'email' => '');

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

**Flashdata**

CodeIgniter soporta "flashdata", o datos de sesión que sólo estarán disponibles para el próximo pedido al servidor. Esto puede ser muy útil, y son típicamente usados para informar o mensajes de estado (por ejemplo: "registro 2 borrado").

Nota: Las variables flash son prefijadas con "flash_" así que evite esos prefijos en sus propios nombres de sesión.

Para agregar flashdata:

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

También puede pasar un arreglo a set_flashdata(), de la misma manera que con set_userdata().

Para leer una variable flashdata:

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

Si encuentra que necesita preservar una variable flashdata durante un pedido adicional, puede hacerlo usando la función keep_flashdata().

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

**Guardando Datos de Sesión en la Base de Datos**

Mientras el arreglo de datos de sesión guardada en la cookie del usuario contenga un identificador de sesión, a menos que guarde los datos de sesión en una base de datos, no hay forma de validarlo. Para algunas aplicaciones que requieren poco o nada de seguridad, la validación del ID de sesión puede no ser necesaria, pero si la aplicación requiere seguridad, la validación es obligatoria.

Cuando los datos de sesión están disponibles en una base de datos, cada vez que una sesión válida es

encontrada en la cookie del usuario, una consulta a la base de datos es hecha para compararla. Si el identificador de sesión no coincide, la sesión es destruida. Los identificadores de sesión nunca pueden ser actualizados, sólo pueden ser generados cuando se crea una nueva sesión.

Para guardar sesiones, debe crear una tabla de base de datos primero. Aquí está el prototipo básico (para MySQL) requerido por la clase de sesión:

```
CREATE TABLE IF NOT EXISTS  `ci_sessions` (
        session_id varchar(40) DEFAULT '0' NOT NULL,
        ip_address varchar(45) DEFAULT '0' NOT NULL,
        user_agent varchar(120) NOT NULL,
        last_activity int(10) unsigned DEFAULT 0 NOT NULL,
        user_data text NOT NULL,
        PRIMARY KEY (session_id),
        KEY `last_activity_idx` (`last_activity`)
);
```

**Nota:** Por defecto, la tabla es llamada ci_sessions, pero puede nombrarla como desee siempre y cuando actualice el archivo `application/config/config.php` para que contenga el nombre que ha elegido. Una vez que haya creado su tabla de base de datos puede habilitar la opción de base de datos en su archivo config.php como sigue:

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

Una vez habilitada, la clase de Sesión guardará datos de sesión en la base de datos.

Esté seguro que haya especificado el nombre de la tabla en su archivo de configuración también:

```
$config['sess_table_name'] = 'ci_sessions";
```

**Nota:** La clase de Sesión tiene un recolector de basura incluido que limpia las sesiones expiradas para que usted no necesito escribir su propia rutina para hacerlo.

### Destruir una sesión

Para limpiar la sesión actual:

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

### Preferencias de Sesión

Encontrará las siguientes preferencias relacionadas a la Sesión en su archivo `application/config/config.php`:

| Preferencia | Por defecto | Opciones | Descripción |
|---|---|---|---|
| **sess_cookie_name** | ci_session | Ninguna | Nombre con el que quiere guardar la sesión. |
| **sess_expiration** | 7200 | Ninguna | TLa cantidad de segundos que quisiera que la sesión dure. El valor por defecto es de 2 horas (7200 segundos). Si quisiera que la sesión no expire, establecer el valor a cero: 0 |
| **sess_expire_on_close** | FALSE | TRUE/FALSE (booleano) | Si provocar que la sesión expire automáticamente cuando se cierra la ventana del navegador. |
| **sess_encrypt_** | FALSE | TRUE/FALSE | Si encriptar o no los datos de sesión. |

| | | | |
|---|---|---|---|
| **cookie** | | (booleano) | |
| **sess_use_database** | FALSE | TRUE/FALSE (booleano) | Si guardar o no los datos de sesión en una base de datos. Tiene que crear la tabla antes de habilitar esta opción. |
| **sess_table_name** | ci_sessions | Cualquier nombre válido de tabla SQ | Nombre de la tabla de base de datos para la sesión. |
| **sess_time_to_update** | 300 | Tiempo en segundos | Esta opción controla cuan frecuentemente la Clase Session se regenerará a sí misma y creará un nuevo ID de sesión. |
| **sess_match_ip** | FALSE | TRUE/FALSE (booleano) | Si tiene que coincidir la dirección IP del usuario al leer los datos de sesión. Advierta que algunos ISP cambian la IP dinámicamente, por lo que si quiere una sesión que no expire, probablemente tenga que establecer esto a FALSE. |
| **sess_match_useragent** | TRUE | TRUE/FALSE (booleano) | Si tiene que coincidir el Agente de Usuario al leer los datos de sesión. |

**Clase Upload**

La Clase Upload de CodeIgniter le permite subir archivos. Puede establecer varias preferencias, restringir el tipo y tamaño de los archivos.

**El Proceso**

Subir un archivo involucra el siguiente proceso general:

- Un formulario de carga es mostrado, permitiendole al usuario seleccionar un archivo y subirlo.
- Cuando el formulario es enviado, el archivo es subido a la destinación que especifique.
- En el camino, el archivo es validado para estar seguro que es permitido basado en las preferencias que haya establecido.
- Una vez cargado, al usuario se le mostrará un mensaje de éxito.

Para demostrar este proceso aquí hay un pequeño tutorial. Luego encontrará información de referencia.

**Crear el Formulario de Carga**

Usando un editor de texto, crear un formulario llamado formulario_carga.php. En él, ubique este código y guardelo en su carpeta `applications/views/`:

```
<html>
<head>
<title>Formulario de Carga</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>
```

Notará que estamos usando el asistente de formulario para crear la etiqueta de apertura de formulario. La carga de archivos requiere un formulario "multipart", así que el asistente crea la sintaxis apropiada para usted. También notará que hay un variable $error. Esto es para que podamos mostrar mensajes de error en el caso que el usuario haga algo mal.

**La página de éxito**

Usando un editor de texto, debes crear un formulario llamado upload_success.php. En el, coloque este código y guárdelo dentro de su carpeta `applications/views/`:

```
<head>
<title>Formulario de Carga</title>
</head>
<body>
<h3>Su archivo fue exitosamente subido!</h3>
```

```
<ul>
<?php foreach($upload_data as $item => $value):?>
<li><?=$item;?>: <?=$value;?></li>
<?php endforeach; ?>
</ul>
<p><?=anchor('upload', 'Subir otro archivo!'); ?></p>
</body>
</html>
```

**El controlador**

Usando un editor de texto, debe crear un controlador llamado upload.php. En el, coloque el siguiente código y guárdelo en su carpeta `applications/controllers/`:

```
<?php
class Upload extends CI_Controller {

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

        function index()
        {
                $this->load->view('formulario_carga', 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('formulario_carga',
$error);
                }
                else
                {
                        $data = array('upload_data' => $this->upload-
```

```
>data());

                        $this->load->view('upload_success', $data);
                }
        }
}
?>
```

**La carpeta Upload**

Ud. necesitará una carpeta de destino para sus imágines subidas. Cree una carpeta en la raiz de su instalación CodeIgniter llamado uploads y establezca los permisos de archivo a 777.

**Pruébelo!**

Para probar su propio formulario, visite su sitio usando una URL similar a esta:

`www.tu-sitio.com/index.php/`*`upload`*`/`

You should see an upload form. Try uploading an image file (either a jpg, gif, or png). If the path in your controller is correct it should work.

**Guía de Referencia**

**Inicializar la Clase Upload**

Como la mayoría de las clases en CodeIgniter, la Clase Upload inicializa en su controlador usando la función $this->load->library:

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

Una vez que la clase Upload está cargada, el objeto estará disponible usando: $this->upload

**Establecer Preferencias**

Similar a otras bibliotecas, controlará lo que está permitido subir basado en sus preferencias. En el controlador que construyó antes, establecer estas preferencias:

```
$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);

// Alternativamente puede establecer preferencias llamando a la
función
// initialize(). Útil si carga automáticamente la clase:
// $this->upload->initialize($config);
```

Las preferencias anteriores son bastante claras por sí mismas. Debajo está la tabla que describe todas las preferencias disponibles.

**Preferencias**

The following preferences are available. The default value indicates what will be used if you do not specify that preference.

| Preferencia | Valor por Defecto | Opciones | Descripción |
|---|---|---|---|
| **upload_path** | Ninguno | Ninguna | Ruta a la carpeta donde se deberían ubicar los archivos subidos. La carpeta tiene que ser escribible y la ruta puede ser absoluta o relativa. |
| **allowed_types** | Ninguno | Ninguna | Tipo mime correspondiente a los tipos de archivo que se permiten subir. Normalmente la extensión del archivo se puede usar como tipo mime. Separe varios tipos con barras verticales. |
| **file_name** | Ninguno | Nombre de archivo deseado | Si lo establece, CodeIgniter renombrará el archivo subido con este nombre. La extensión provista en el nombre de archivo tiene que ser la de un tipo de archivo permitido. |
| **overwrite** | FALSE | TRUE/FALSE (booleano) | Establecida a TRUE, si existe un archivo con el mismo nombre que el que está cargando, se sobrescribirá. Establecida a FALSE, se le agregará un número al nombre de archivo si existe otro con el mismo nombre. |
| **max_size** | 0 | Ninguna | Tamaño máximo (en kilobytes) que el archivo puede tener. Establecer a cero para sin límite. Nota: La mayoría de las instalaciones de PHP tiene su propio límite, según se especifica en el archivo php.ini. Normalmente por defecto es 2 MB (o 2048 KB). |
| **max_width** | 0 | Ninguna | Ancho máximo (en pixeles) que el archivo puede tener. Establecer a cero para sin límite. |
| **max_height** | 0 | Ninguna | Alto máximo (en pixeles) que el archivo puede tener. Establecer a cero para sin límite. |
| **max_filename** | 0 | Ninguna | Longitud máxima que un nombre de archivo puede tener. Establecer a cero para sin límite. |
| **encrypt_name** | FALSE | TRUE/FALSE (booleano) | Establecida a TRUE se convertirá el nombre del archivo a una cadena encriptada aleatoriamente. Puede ser útil si quisiera que la persona que suba el archivo no pueda discernir su nombre. |
| **remove_spaces** | TRUE | TRUE/FALSE (booleano) | Establecida a TRUE, cualquier espacio en el nombre del archivo se convertirá en guión de subrayado. Se recomienda. |

**Establecer Preferencias en un Archivo de Configuración**

Si prefiere no establecer las preferencias usando este método, en su lugar puede ponerlas dentro de un archivo de configuración. Simplemente cree un nuevo archivo llamado *upload.php* y agregue el array *$config* a ese archivo. Luego guarde el archivo en *config/upload.php* y se lo utilizará automáticamente. NO necesitará usar la función $this->upload->initialize si guarda sus preferencias en un archivo de configuración.

**Referencia de Funciones**

Están disponibles las siguiente funciones:

**$this->upload->do_upload()**

Realiza la subida basada en las preferencias que se establecieron. Nota: Por defecto la rutina de subida espera que los archivos vengan en un campo de formulario llamado userfile y el formulario tiene que ser de tipo "multipart":

```
<form method="post" action="some_action" enctype="multipart/form-
data" />
```

Si quisiera establecer su propio nombre de archivo, simplemente pase su valor a la función do_upload():

```
$field_name = "algun_nombre_de_campo";
$this->upload->do_upload($field_name);
```

**$this->upload->display_errors()**

Recupera cualquier mensaje de error si la función do_upload() devuelve FALSE. La función no hace eco automáticamente, devuelve los datos para que pueda asignarlos si resulta necesario.

**Formatear Errores**

Por defecto, la función anterior envuelve cualqueir error dentro de etiquetas

. Puede establecer sus propios delimitadores con esto:

```
$this->upload->display_errors('<p>', '</p>');
```

**$this->upload->data()**

Esta es una función helper que devuelve un array que contiene todos los datos relacionados al archivo que subió. Este es el prototipo del array:

```
Array
(
        [file_name] => mi_pic.jpg
        [file_type] => image/jpeg
        [file_path] => /ruta/a/su/subida/
        [full_path] => /ruta/a/su/subida/jpg.jpg
        [raw_name] => mi_pic
        [orig_name] => mi_pic.jpg
        [client_name] => mi_pic.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"
```

)

**Explicación**

Esta es una explicación de los ítems del array anterior

| Item | Descripción |
|---|---|
| **file_name** | Nombre del archivo que se subió, incluyendo la extensión |
| **file_type** | Tipo Mime del archivo |
| **file_path** | Ruta absoluta del servidor al archivo |
| **full_path** | Ruta absoluta del servidor incluyendo el nombre de archivo |
| **raw_name** | Nombre del archivo sin la extensión |
| **orig_name** | Nombre original del archivo. Solamente es útil si usa la opción de nombre encriptado |
| **client_name** | Nombre de archivo como lo proporciona el agente de usuario, antes de cualquier preparación o incremento del nombre de archivo |
| **file_ext** | Extensión de archivo con el punto |
| **file_size** | Tamaño del archivo en kilobytes |
| **is_image** | Si el archivo es o no una imagen. 1 = imagen. 0 = no lo es |
| **image_width** | Ancho de la imagen |
| **image_height** | Altura de la imagen |
| **image_type** | Tipo de imagen. Normalmente la extensión sin el punto |
| **image_size_str** | Una cadena que contiene ancho y alto. Útil para ponerlo en una etiqueta de imagen |

**Clase Output**

La Clase Output es una clase pequeña con una función principal: Enviar la página web terminada al navegador. Es responsable de almacenar en caché las páginas web, si se usa esta funcionalidad.

**Nota:** El sistema inicializa automáticamente esta clase, por lo que no hay necesidad de hacerlo manualmente.

Bajo circunstancias normales no advertirá la Clase Output, ya que funciona de forma transparente sin su intervención. Por ejemplo, cuando usa la clase Load para cargar un archivo de vista, se pasa automáticamente a la Clase Output, a la cual llamará automáticamente CodeIgniter al final de la ejecución del sistema. Sin embargo, es posible que Ud intervenga manualmente con la salida si lo necesita, usando cualesquiera de las siguientes dos funciones:

**$this->output->set_output();**

Le permite establecer manualmente la cadena de salida final. Ejemplo de Uso:

```
$this->output->set_output($data);
```

**Importante:** Si establece manualmente la salida, tiene que ser la última cosa hecha en la función que la llama. Por ejemplo, si arma una página en una de sus funciones controlador, no establezca la salida hasta el fin.

**$this->output->set_content_type();**

Le permite establecer el tipo mime de su página para que sirva datos JSON, JPEG's, XML, etc. fácilmente.

```
$this->output
        ->set_content_type('application/json')
        ->set_output(json_encode(array('foo' => 'bar')));

$this->output
        ->set_content_type('jpeg') // También podría usar ".jpeg" que
tendrá el punto eliminado antes de buscar en config/mimes.php
->set_output(file_get_contents('files/algo.jpg'));
```

**Importante:** Asegurarse que ninguna cadena no-mime que pase a este método existe en config/mimes.php o no tendrá efecto.

**$this->output->get_output();**

Le permite recuperar manualmente cualquier salida que haya sido enviada para almacenar en la clase Output. Ejemplo de Uso:

```
$string = $this->output->get_output();
```

Advierta que los datos solamente se recuperarán con esta función si previamente fueron enviados a la clase Output por una de las funciones de CodeIgniter, como *$this->load->view()*.

**$this->output->append_output();**

Agrega datos en la cadena de salida. Ejemplo de Uso:

```
$this->output->append_output($data);
```

**$this->output->set_header();**

Le permite establecer manualmente los encabezados de servidor, que la clase Output enviará cuando imprima la pantalla final renderizada. Ejemplo:

```
$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");
$this->output->set_header("Pragma: no-cache");
```

**$this->output->set_status_header(*code*, '*text*');**

Le permite establecer manualmente el encabezado de estado del servidor. Ejemplo:

```
$this->output->set_status_header('401');
// Establecer el encabezado como: No Autorizado
```

Ver aquí la lista completa de encabezados.

**$this->output->enable_profiler();**

Le permite habiitar/deshabilitar el Perfilador, el cual mostrará pruebas de desempeño y otros datos al final de las páginas con fines de depuración y optimización.

Para habilitar el perfilador, ubique la siguiente función en cualquier parte dentro de sus funciones controlador:

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

Cuando esté habilitado, se generará un informe y se lo mostrará al final de las páginas.

Para deshabilitar el perfilador, usará:

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

**this->output->set_profiler_sections()**

Le permite habilitar/deshabilitar secciones específicas del Perfilador cuando esté habilitado. Referirse a la documentación del Perfilador para mayor información.

**$this->output->cache()**

La biblioteca output de CodeIgniter también controla el almacenamiento en caché. Para mayor información, vea la documentación del almacenamiento en caché.

**Analizar las Variables de Ejecución**

CodeIgniter analizará las seudo-variables *{elapsed_time}* y *{memory_usage}* en su salida por defecto. Para deshabilitar esto, establecer la propiedad de clase *$parse_exec_vars* a *FALSE* en su controlador.

```
$this->output->parse_exec_vars = FALSE;
```

**Configuración de Base de Datos**

CodeIgniter tiene un archivo de configuración que permite almacenar los valores de conexión (usuario, contraseña, nombre de base de datos). El archivo de configuración se localiza en:

```
application/config/database.php
```

Los parametros de configuración son guardados en un arreglo multidimensional con este prototipo:

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "root";
$db['default']['password'] = "";
$db['default']['database'] = "database_name";
$db['default']['dbdriver'] = "mysql";
$db['default']['dbprefix'] = "";
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = FALSE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = "";
$db['default']['char_set'] = "utf8";
$db['default']['dbcollat'] = "utf8_general_ci";
$db['default']['swap_pre'] = "";
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;
```

La razon por la que usamos un arreglo multidimensional y no uno más sencillo es permitir el almacenamiento de múltiple parámetros de conexión. Por ejemplo si queremos trabajar con múltiples entornos (desarrollo, producción, prueba, etc.) bajo la misma instalación, podemos un grupo de parametro de configuración para cada uno, luego cambiar entre cada uno según sea necesario. Por ejemplo, para configurar un entorno de prueba podemos hacer esto:

```
$db['test']['hostname'] = "localhost";
$db['test']['username'] = "root";
$db['test']['password'] = "";
$db['test']['database'] = "database_name";
$db['test']['dbdriver'] = "mysql";
$db['test']['dbprefix'] = "";
$db['test']['pconnect'] = TRUE;
$db['test']['db_debug'] = FALSE;
$db['test']['cache_on'] = FALSE;
$db['test']['cachedir'] = "";
$db['test']['char_set'] = "utf8";
$db['test']['dbcollat'] = "utf8_general_ci";
$db['test']['swap_pre'] = "";
$db['test']['autoinit'] = TRUE;
$db['test']['stricton'] = FALSE;
```

Luego para usar este grupo de parámetros (del entorno de prueba) cambiamos el valor de la variable que se encuentra en el archivo de configuración, así:

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

Nota: El nombre "test" es arbitrario. Puede ser cualquiera que querramos. Por defecto usamos el valor

"default" para la primer conexión, pero esto también puede ser cambiado por algo más relevante para el proyecto.

**Active Record**

La Clase Active Record es globalmente activada o desactivada estableciendo la variable $active_record en el archivo de configuración para la base de datos, en TRUE/FALSE. Si no vamos a usar la Clase Active Record, establecerla en FALSE hará que se utilicen menos recursos cuando se inicialicen las clases de base de datos.

```
$active_record = TRUE;
```

**Note:** que algunas clases de CodeIgniter como las de sessiones (sessions) requieren que Active Records esté activado para conseguir cierta funcionalidad.

**Explicacion de los valores:**

- **hostname** - El nombre de anfitrion (hopst) de tu servidor de base de datos. A menudo es "localhost".
- **username** - El nombre de usuario para acceder a la base de datos.
- **password** - El password usado para conectar a la base de datos.
- **database** - El nombre de la base de datos a la que se conectará.
- **dbdriver** - El tipo de Base de Datos. por ejemplo: mysql, postgre, obdc, etc. Debe ser especificado en minúsculas.
- **dbprefix** - Un prefijo opcional que será agregado al nombre de la tabla cuando se ejecuten consultas Active Record. Esto permite a multiples instalaciones de CodeIgniter compartir la base de datos.
- **pconnect** - TRUE/FALSE (boolean) -Si se debe usar una conexion permanente.
- **db_debug** - TRUE/FALSE (boolean) -Si los errores de la base de datos deben ser desplegados.
- **cache_on** - TRUE/FALSE (boolean) -Si el cacheo de consultas debe ser activado, vea tambien Clase Database Caching.
- **cachedir** - La ruta absoluta a la carpeta del cache de consultas del servidor.
- **char_set** - El juego de caracteres a ser usado para comunicarse con la base de datos.
- **dbcollat** - El cotejo de caracteres usado en comunicaciones con la base de datos.
- **swap_pre** - Un *prefijo de tabla* por defecto que se tiene que intercambiar con dbprefix. Esto es útil para aplicaciones distribuidas donde puede correr consultas escritas manualmente, y se necesita que el prefijo siga siendo personalizable por el usuario.
- **autoinit** - Si conectar o no automáticamente a la base de datos cuando se carga la biblioteca. Si está establecido a FALSE, la conexión tomará lugar antes de ejecutar la primera consulta.
- **stricton** - TRUE/FALSE (booleano) - Si forzar conexiones en "Modo Estricto", bueno para asegurar el SQL estricto mientras se desarrolla una aplicación.
- **port** - El numero de puerto de la base de datos. Actualmente usado solamente por el driver de Postgre. Para usar este valor debes agregar una linea al arreglo de configuraciones de la base de datos. `$db['default']['port'] = 5432;`

**Nota:** Dependiendo de qué plataforma de base de datos que está utilizando (MySQL, postgres, etc) no todos los valores serán necesarios. Por ejemplo, cuadno se usa SQLite no necesitaras proveer un usuario o password, y el nombre de la base de datos sera la ruta al archivo de la base de datos. La información anterior, asume que usas MySQL.

**Conectar a una Base de Datos**

Hay dos formas de conectarse a una base de datos:

**Conectando automáticamente**

La característica de "auto conexión" cargará e instanciará la clase de base de datos en cada página cargada. Para permitir "auto conexión", agregue la palabra *database* al arreglo "library", como se indica en el siguiente archivo:

`application/config/autoload.php`

**Conectando Manualmente**

Si sólo algunas páginas requieren conexión a la base de datos, puedes conectar manualmente agregando esta línea de código en cualquier función donde sea necesario, o en el constructor de tu clase para hacer a la base de datos disponible globalmente en esa clase.

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

Si la función de arriba **no** contiene ninguna información en el primer parámetro, conectará al grupo especificado en tu archivo de configuración de base de datos. Para la mayoría de las personas, este es el método preferido de uso.

**Parámetros Disponibles**

1. Los valores de conexión de la base de datos, pasado tanto como un array o como una cadena DSN.
2. TRUE/FALSE (booleano). Si devolver el ID de conexión (ver más abajo Conectar a Varias Bases de Datos).
3. TRUE/FALSE (booleano). Si habilitar la Clase Active Record. Establecido a TRUE por defecto.

**Conectar Manualmente a una Base de Datos**

El primer parámetro de esta función puede **opcionalmente** ser usado para especificar un grupo en particular de tu archivo de configuración, o puedes envíar valores de conexión para uan base de datos que no esté especificada en tu archivo de configuración. Ejemplos:

Para seleccionar un grupo específico de tu archivo de configuración, puede hacer esto:

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

Donde `nombre_grupo` es el nombre del grupo de la conexión de tu archivo de configuración.

Para conectar manualmente a una base de datos deseada, puedes pasar un arreglo de valores:

```
$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['cache_on'] = FALSE;
$config['cachedir'] = "";
$config['char_set'] = "utf8";
$config['dbcollat'] = "utf8_general_ci";

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

Para información en cada uno de esos valores por favor vea la página de configuración.

O puedes envíar valores de tu base de datos como un "Data Source Name". DSNs debe tener este prototipo:

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

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

Para anular los valores de configuración por defecto al conectar con una cadena DSN, agregar las variables de configuración como un query string.

```
$dsn =
'dbdriver://username:password@hostname/database?char_set=utf8&dbcolla
t=utf8_general_ci&cache_on=true&cachedir=/path/to/cache';

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

**Conectando a Múltiples Bases de Datos**

Si necesita conectar a más de una base de datos simultáneamente puede hacer como sigue:

```
$DB1 = $this->load->database('grupo_uno', TRUE);
$DB2 = $this->load->database('grupo_dos', TRUE);
```

Nota: Cambié las palabras "grupo_uno" y "grupo_dos" al nombre del grupo específico al que se está conectando (o puede pasar los valores de conexión como se indico antes).

Cuando se establece el segundo parámetro como TRUE (booleano) la función devolverá el objeto de base de datos.

Cuando se conecta de esta forma, deberá usar el nombre del objeto para ejecutar comandos en vez de la sintaxis usada a través de esta guía. En otras palabras, en vez de ejecutar comandos con:

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

En vez use:

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

**Reconectar / Mantener la Conexión Viva**

Si se excede el tiempo de espera de inactividad del servidor de base de datos mientras está realizando

alguna tarea pesada de PHP (por ejemplo, procesando una imagen), debería considerar hacer ping al servidor usando el método reconnect() antes de enviar otras consultas, el cual puede mantener la conexión viva o restablecerla.

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

### Cerrar Manualmente la Conexión

Mientras que CodeIgniter se encarga inteligentemente de cerrar las conexiones de bases de datos, la conexión se puede cerrar explícitamente.

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

**Generar Resultados de Consultas**

Hay varias formas de generar los resultados de una consulta:

**result()**

Esta función retorna el resultado de la consulta como un arreglo de **objetos**, o **un arreglo vacío** en el caso de fracaso. Normalmente puede utilizar esto en un bucle foreach, de esta manera:

```
$query = $this->db->query("SU CONSULTA");

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

La anterior función es un alias de result_object().

Si ejecuta consultas que podrían no producir un resultado, lo animamos a probar el resultado de esto primero:

```
$query = $this->db->query("SU CONSULTA");

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

También puede pasar una cadena a result(), la que representa una clase a instanciar por cada objeto de resultado (**nota:** se tiene que cargar esta clase).

```
$query = $this->db->query("SELECT * FROM users;");

foreach ($query->result('User') as $user)
{
   echo $row->name; // call attributes
   echo $row->reverse_name(); // or methods defined on the 'User'
class
}
```

**result_array()**

Esta función retorna el resultado de la consulta como un arreglo puro, o un arreglo vacío cuando no se producen resultados. Normalmente puede utilizar esto en un bucle foreach, de esta manera:

```
$query = $this->db->query("SU CONSULTA");
```

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

**row()**

Esta función retorna una única fila como resultado. Si su consulta tiene mas de una fila, ésta solo retorna la primera fila. El resultado es retornado como un **objeto**. Aquí hay un ejemplo de su uso:

```
$query = $this->db->query("SU CONSULTA");

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

   echo $row->title;
   echo $row->name;
   echo $row->body;
}
```

Si desea retornar una fila específica puede presentar el número de la fila como un dígito en el primer parámetro:

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

También puede agregar un segundo parámetro de cadena, que es el nombre de la clase con la que instanciar la fila:

```
$query = $this->db->query("SELECT * FROM users LIMIT 1;");

$query->row(0, 'User')
echo $row->name; // call attributes
echo $row->reverse_name(); // or methods defined on the 'User' class
```

**row_array()**

Idéntica a la función *row()* anterior, excepto que retorna un arreglo. Por ejemplo:

```
$query = $this->db->query("SU CONSULTA");

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

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

Si desea retornar una fila específica puede presentar el número de la fila como un dígito en el primer parámetro:

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

Además, se puede caminar hacia adelante / atrás / primera / última fila a través de sus resultados utilizando estas variaciones:

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

Por defecto, retorna un objeto a menos que ponga la palabra "array" en el parámetro:

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

**Funciones Helper de Resultados**

**$query->num_rows()**

El número de filas retornado por la consulta. Nota: En este ejemplo, $query es la variable que el objeto resultante de la consulta es asignado a:

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

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

**$query->num_fields()**

El número de CAMPOS (columnas) retornado por la consulta. Asegúrese de llamar a la función utilizando su objeto resultante de la consulta:

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

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

**$query->free_result()**

Se libera la memoria asociada con el resultado y elimina los recursos ID resultantes. Normalmente PHP libera la memoria automáticamente al finalizar la ejecución de scripts. Sin embargo, si está ejecutando una gran cantidad de consultas en un script en particular, puede que desee liberar el resultado después de cada resultado de consulta que se ha generado, con el fin de reducir los consumos de memoria. Por ejemplo:

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

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

```
}
$query->free_result(); // El objeto resultante $query dejará de estar
disponible

$query2 = $this->db->query('SELECT name FROM some_table');

$row = $query2->row();
echo $row->name;
$query2->free_result(); // El objeto resultante $query2 dejará de
estar disponible
```

**Funciones Helper de Consultas**

**$this->db->insert_id()**

El número de ID de cuando se realizo una inserción a la base de datos.

**$this->db->affected_rows()**

Muesta el número de filas afectadas, cuando se realiza una consulta de tipo de "escritura" (insert, update, etc.).

Nota: En MySQL "DELETE FROM TABLE" devuelve 0 filas afectadas. La clase de base de datos tiene un pequeño hack que le permite devolver el número correcto de filas afectadas. Por defecto este hack está habilitado pero se puede apagar en el archivo de driver de la base de datos.

**$this->db->count_all();**

Permite determinar el número de filas en una tabla en particular. El primer parámetro es el nombre de la tabla. Ejemplo:

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

// Produce un entero, como 25
```

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

Imprime la plataforma de base de datos que está corriendo (MySQL, MS SQL, Postgre, etc...):

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

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

Imprime la versión de base de datos que está corriendo:

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

**$this->db->last_query();**

Devuelve la última consulta que fue ejecutada (la cadena de la consulta, no el resultado). Example:

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

// Produce: SELECT * FROM sometable....
```

Las dos funciones siguientes ayudan a simplificar el proceso de escritura de INSERTs Y UPDATEs de base de datos.

**$this->db->insert_string();**

Esta función simplifica el proceso de escritura de inserciones de base de datos. Devuelve una cadena SQL de inserción correctamente formada. Ejemplo:

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

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

El primer parámetro es el nombre de la tabla. El segundo un arreglo asociativo con los datos que serán insertados. El ejemplo anterior produce:

```
INSERT INTO nombre_tabla (nombre, email, url) VALUES ('Rick',
'rick@your-site.com', 'www.your-site.com')
```

Nota: Los valores son automáticamente escapados, produciendo consultas más seguras.

**$this->db->update_string();**

Esta función simplifica el proceso de escritura de actualizaciones de base de datos. Devuelve una cadena SQL de actualización correctamente formada. Ejemplo:

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

$condicion = "autor_id = 1 AND estado = 'activo'";

$str = $this->db->update_string('nombre_tabla', $data, $condicion);
```

El primer parámetro es el nombre de la tabla, el segundo un arreglo asociativo con los datos que serán insertados, y el tercer parámetro es la cláusula "where". El ejemplo anterior produce:

```
UPDATE nombre_tabla SET name = 'Rick', email = 'rick@your-site.com',
url = 'www.your-site.com' WHERE autor_id = 1 AND estado = 'activo'
```

Nota: Los valores son automáticamente escapados, produciendo consultas más seguras.

**Clase Active Record**

CodeIgniter usa una versión modificada del Patrón de Base de Datos Active Record. Este patrón permite obtener, insertar y actualizar información in tu base de datos con mínima codificación. En algunos casos, sólo una o dos líneas de código son necesarias para realizar una acción de base de datos. CodeIgniter no requiere que cada tabla de la base de datos sea un propio archivo de clase. Se permite una interface más simplificada

Más allá de la simplicidad, un beneficio mayor de usar la Active Record es que te permite crear una aplicación independiente de la base de datos que usa, ya que la sintaxis de consulta es generada por cada adaptador de base de datos. También permite queries más seguras, ya que los valores son escapadas automáticamente por el sistema.

**Nota:** Si tienes intenciones de usar tus propias consultas, puedese deshabilitar esta clase en tu archivo de configuración de base de datos, permitiendo a la librería de la base de datos y adaptadores usar menos recursos

**Seleccionar Datos**

Las siguientes funciones permiten construir una sentencia **SELECT** SQL.

**Nota: Si está usando PHP 5, puede usar métodos en cadena para una sintaxis más compacta. Esto es descripto al final de la página.**

**$this->db->get();**

Ejecuta la consulta de selección y devuelve el resultado. Puede ser usado solo, para traer todos los registros de una tabla:

```
$consulta = $this->db->get('mitabla');

// Produce: SELECT * FROM mitabla
```

El segundo y tercer parámetro permiten establecer cláusulas de límite y principio:

```
$consulta = $this->db->get('mitabla', 10, 20);

// Produce: SELECT * FROM mitabla LIMIT 20, 10 (en MySQL. Otras bases
de datos tienen pequeñas diferencias en la sintaxis)
```

Notará que la funcion de arriba es asignada a la variable llamada `$consulta`, que puede ser usada para mostrar los resultados:

```
$consulta = $this->db->get('mitabla');

foreach ($consulta->result() as $fila)
{
    echo $fila->titulo;
}
```

**$this->db->get_where();**

Identica a la función de arriba, excepto que permite agregar una cl´usula "where" en el segundo parámetro, en vez de usar la función db->where():

```
$consulta = $this->db->get_where('mitabla', array('id' => $id),
$limite, $principio);
```

Por favor lea sobre la función where más abajo para más información

Nota: get_where() era anteriormente conocida como getwhere(), que ha sido eliminada

**$this->db->select();**

Permite escribir la porción de SELECT de tu consulta:

```
$this->db->select('titulo, contenido, fecha');

$consulta = $this->db->get('mitabla');

// Produce: SELECT titulo, contenido, fecha FROM mitabla
```

**Nota:** Si está seleccionando todo (*) de una tabla, no es necesario usar esta función. Cuando se omite, CodeIgniter asume que desa SELECT *

$this->db->select() acepta un opcional segundo parámetro. Si se establece como FALSE, CodeIgniter no intentará proteger los nombres de campos u tablas. Esto es útil si necesita una consulta de selección compuesta.

```
$this->db->select('(SELECT SUM(pagos.monto) FROM pagos WHERE
pagos.factura_id=4') AS monto_pagado', FALSE);
$consulta = $this->db->get('mitabla');
```

**$this->db->select_max();**

Escribe una porción "SELECT MAX(campo)" de tu consulta. Opcionalmente, puedes incluir un segunda parámetro para renombrar el campo resultante.

```
$this->db->select_max('edad');
$consulta = $this->db->get('miembros');
// Produce: SELECT MAX(edad) as edad FROM miembros

$this->db->select_max('edad', 'edad_miembro');
$query = $this->db->get('miembros');
// Produce: SELECT MAX(edad) as edad_miembro FROM miembros
```

**$this->db->select_min();**

Escriba una porción "SELECT MIN(campo)" de tu consulta. Como en select_max(), puedes opcionalmente incluir un segundo parámetro para renombrar el campo resultante.

```
$this->db->select_min('edad');
$consulta = $this->db->get('miembros');
// Produce: SELECT MIN(edad) as edad FROM miembros
```

**$this->db->select_avg();**

Escriba una porción "SELECT AVG(campo)" de tu consulta. Como en select_max(), puedes opcionalmente incluir un segundo parámetro para renombrar el campo resultante.

```
$this->db->select_avg('edad');
$consulta = $this->db->get('miembros');
// Produce: SELECT AVG(edad) as edad FROM miembros
```

**$this->db->select_sum();**

Escriba una porción "SELECT SUM(campo)" de tu consulta. Como en select_max(), puedes opcionalmente incluir un segundo parámetro para renombrar el campo resultante.

```
$this->db->select_sum('edad');
$consulta = $this->db->get('miembros');
// Produce: SELECT SUM(edad) as edad FROM miembros
```

**$this->db->from();**

Permite escribir la porción FROM de la consulta:

```
$this->db->select('titulo, contenido, fecha');
$this->db->from('mitabla');

$consulta = $this->db->get();

// Produce: SELECT titulo, contenido, fecha FROM mitabla
```

Nota: Como se mostró antes, la porción FROM de tu consulta puede ser especificada en la función $this->db->get(), así que use el método que prefiera

**$this->db->join();**

Permite escribir una porción JOIN de la consulta:

```
$this->db->select('*');
$this->db->from('blogs');
$this->db->join('comentarios', 'comentarios.id = blogs.id');

$query = $this->db->get();

// Produce:
// SELECT * FROM blogs
// JOIN comentarios ON comentarios.id = blogs.id
```

Multiples llamados a la función pueden ser hechos si necesita multiples joins en una consulta.

Si necesita algo distinto al natural JOIN puede especificarlo a través del tercer parámetro de la función. Las opciones son: left, right, outer, inner, left outer, and right outer.

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

// Produce: LEFT JOIN comentarios ON comentarios.id = blogs.id
```

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

Está función habilita establecer cláusulas **WHERE** usando uno de cuátro métodos:

**Nota:** Todos los valores pasados a esta función son escapados automáticamente, produciendo consultas más seguras.

1. **Método simpre de clave/valor:** `$this->db->where('nombre', $nombre);`

   ```
   // Produce: WHERE nombre = 'Jose'
   ```

   Note que el signo igual es agregado para uested.

   Si utiliza multiples llamadas a la función, ellos serán encadenados juntos con un *AND* entre ellos:

   ```
   $this->db->where('nombre', $nombre);
   $this->db->where('titulo', $titulo);
   $this->db->where('estado', $estado);

   // WHERE nombre = 'Jose' AND titulo = 'jefe' AND estado =
   'activo'
   ```

2. **Método especial de clave/valor:**

   Se puede incluir un operador en el primer parámetro con el objeto de controlar la comparación:

   ```
   $this->db->where('nombre !=', $nombre);
   $this->db->where('id <', $id);

   // Produce: WHERE nombre != 'Jose' AND id < 45
   ```

3. **Método de arreglo asociativo:** `$arreglo = array('nombre' => $nombre, 'titulo' => $titulo, 'status' => $status);`

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

   // Produce: WHERE nombre = 'Joe' AND titulo = 'boss' AND status
   = 'active'
   ```

   Puede incluir operadores propios cuando se usa este método también:

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

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

4. **Cadena especial:**

   Se puede escribir cláusulas propias manualmente:

   ```
   $where = "nombre='Jose' AND estado='jefe' OR estado='activo'";

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

**$this->db->or_where();**

Está función es idéntica a la de arriba, excepto que multiples instancias son unidas por OR:

```
$this->db->where('nombre !=', $nombre);
$this->db->or_where('id >', $id);

// Produce: WHERE nombre != 'Joe' OR id > 50
```

Nota: or_where() era anteriormente conocida como orwhere(), la cual ha sido deprecada.

**$this->db->where_in();**

Genera WHERE campo IN ('item', 'item') unido con AND si corresponde

```
$nombres = array('Frank', 'Todd', 'James');
$this->db->where_in('username', $nombres);
// Produce: AND WHERE username IN ('Frank', 'Todd', 'James')
```

**$this->db->or_where_in();**

Genera WHERE campo IN ('item', 'item') unido con OR si corresponde

```
$nombres = array('Frank', 'Todd', 'James');
$this->db->or_where_in('username', $nombres);
// Produce: OR WHERE username IN ('Frank', 'Todd', 'James')
```

**$this->db->where_not_in();**

Genera WHERE campo NOT IN ('item', 'item') unido con AND si corresponde

```
$nombres = array('Frank', 'Todd', 'James');
$this->db->where_not_in('username', $nombres);
// Produce: AND WHERE username NOT IN ('Frank', 'Todd', 'James')
```

**$this->db->or_where_not_in();**

Genera WHERE campo NOT IN ('item', 'item') unido con OR si corresponde

```
$nombres = array('Frank', 'Todd', 'James');
$this->db->or_where_not_in('username', $nombres);
// Produces: OR WHERE username NOT IN ('Frank', 'Todd', 'James')
```

**$this->db->like();**

Está función permite generar cláusulas **LIKE**, útiles para realizar búsquedas.

**Nota:** Todos los valores pasados a esta función son escapados automáticamente.

1. **Método simple de clave/valor:** `$this->db->like('titulo', 'match');`

   ```
   // Produce: WHERE titulo LIKE '%match%'
   ```

   Si utiliza múltiples llamados a la función, ellos serán encadenados con *AND* entre ellos:

   ```
   $this->db->like('titulo', 'match');
   $this->db->like('body', 'match');

   // WHERE titulo LIKE '%match%' AND body LIKE '%match%
   ```

   Si desea

controlar donde la wildcard (%) es ubicada, puede utilizar un tercer parámetro opcional. Las opciones son 'before', 'after' y 'both' (por defecto). `$this->db->like('titulo', 'match', 'before');`

```
// Produces: WHERE titulo LIKE '%match'

$this->db->like('titulo', 'match', 'after');
// Produces: WHERE titulo LIKE 'match%'

$this->db->like('titulo', 'match', 'both');
// Produces: WHERE titulo LIKE '%match%'
```

2. **Método de arreglo asociativo:** `$array = array('titulo' => $match, 'pagina1' => $match, 'pagina2' => $match);`

```
$this->db->like($array);

// WHERE titulo LIKE '%match%' AND pagina1 LIKE '%match%' AND
pagina2 LIKE '%match%'
```

**$this->db->or_like();**

Está función es idéntica a la anterior, excepto que multiples instances son unidas por OR:

```
$this->db->like('titulo', 'match');
$this->db->or_like('body', $match);

// WHERE titulo LIKE '%match%' OR body LIKE '%match%'
```

Nota: or_like() era anteriormente conocida como orlike(), la cual ha sido deprecada.

**$this->db->not_like();**

Esta función es idéntica a **like()**, excepto que genera sentencias NOT LIKE:

```
$this->db->not_like('titulo', 'match');

// WHERE titulo NOT LIKE '%match%
```

**$this->db->or_not_like();**

Esta función es idéntica a **not_like()**, excepto que muchas instancias son unidas por OR:

```
$this->db->like('titulo', 'match');
$this->db->or_not_like('body', 'match');

// WHERE titulo LIKE '%match% OR body NOT LIKE '%match%'
```

**$this->db->group_by();**

Permite escribir porciones GROUP BY de la consulta:

```
$this->db->group_by("titulo");
```

```
// Produces: GROUP BY titulo
```

También se puede pasar un arreglo de multiples valores:

```
$this->db->group_by(array("titulo", "fecha"));

// Produce: GROUP BY titulo, fecha
```

Nota: group_by() era anteriormente conocida como groupby(), la cual ha sido deprecada.

**$this->db->distinct();**

Agrega la palabra clave "DISTINCT" a la consulta

```
$this->db->distinct();
$this->db->get('tabla');

// Produce: SELECT DISTINCT * FROM tabla
```

**$this->db->having();**

Permite escribir la porción HAVING de la consulta:

```
$this->db->having('user_id = 45');
// Produce: HAVING user_id = 45

$this->db->having('user_id', 45);
// Produce: HAVING user_id = 45
```

También puede pasar un arreglo de multiples valores:

```
$this->db->having(array('titulo =' => 'Mi titulo', 'id <' => $id));

// Produce: HAVING titulo = 'Mi titulo', id < 45
```

**$this->db->or_having();**

Idéntica a having(), sólo que separa múltiples cláusulas con "OR".

**$this->db->order_by();**

Permite establecer una cláusula de ORDER BY. El primer parámetro contiene el nombre de la columna por la que desea ordenar. El segundo parámetro establece la dirección del resultado. Las opciones son `asc` or `desc`, or `random`.

```
$this->db->order_by("titulo", "desc");

// Produce: ORDER BY titulo DESC
```

También puede pasar su propia cadena en el primer parámetro:

```
$this->db->order_by('titulo desc, nombre asc');

// Produce: ORDER BY titulo DESC, nombre ASC
```

O múltiples llamados a la función pueden ser hechos si necesita múltiples campos.

```
$this->db->order_by("titulo", "desc");
$this->db->order_by("nombre", "asc");

// Produces: ORDER BY titulo DESC, nombre ASC
```

Nota: order_by() era anteriormente conocida como orderby(), la cual a sido deprecada.

Note: ordernamiento aleatorio no es soportado actualmetne en los "drivers" Oracle o MSSQL. Estos serán predeterminados como 'ASC'.

**$this->db->limit();**

Permite limitar el número de filas que desea que devuelva la consulta:

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

// Produce: LIMIT 10
```

El segundo parámetro permite establecer el inicio del resultado.

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

// Produce: LIMIT 20, 10 (in MySQL. Otras bases de datos tienen
pequeñas diferencias de sintaxis.)
```

**$this->db->count_all_results();**

Permite determinar el número de filas de una consulta Active Record en particular. Las consultas aceptan las restricciones de Active Record tales como where(), or_where(), like(), or_like(), etc. Ejemplo:

```
echo $this->db->count_all_results('mi_tabla');
// Produce un entero, como 25

$this->db->like('titulo', 'match');
$this->db->from('mi_tabla');
echo $this->db->count_all_results();
// Produce un entero, como 17
```

**$this->db->count_all();**

Permite determinar el número de filas de una tabla en particular. Permits you to determine the number of rows in a particular table. Envíe el nombre de la tabla como primer parámetro. Por ejemplo:

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

// Produce un entero, como 25
```

**Insertar Datos**

**$this->db->insert();**

Genera una cadena de inserción basado en los datos que se suministren, y ejectua la consulta. Se puede pasar un **arreglo** o un **objeto** a la función. Aquí hay un ejemplo, usando un arreglo:

```
$data = array(
               'titulo' => 'Mi titulo' ,
               'nombre' => 'Mi nombre' ,
               'fecha' => 'Mi fecha'
            );

$this->db->insert('mitabla', $data);

// Produces: INSERT INTO mitabla (titulo, nombre, fecha) VALUES ('Mi
titulo', 'Mi nombre', 'Mi fecha')
```

El primer parámetro contiene el nombre de la table, la segunda es un arreglo de valores.

Aquí hay un ejemplo usando un objeto:

```
/*
    class Myclass {
        var $titulo = 'Mi titulo';
        var $contenido = 'Mi Contenido';
        var $fecha = 'Mi Fecha';
    }
*/

$objeto = new Myclass;

$this->db->insert('mitabla', $objeto);

// Produce: INSERT INTO mitabla (titulo, contenido, fecha) VALUES
('Mi titulo', 'Mi Contenido', 'Mi Fecha')
```

El primer parámetro contendrá el nombre de la tabla, el segundo es un arreglo asociativa de valores.

**Nota:** Todos los valores son escapados automáticamente produciendo consultas más seguras.

**$this->db->insert_batch();**

Genera una cadena insert de SQL basada en los datos provistos, y ejecuta la consulta. A la función se le puede pasar tanto un **array** como un **objeto**. Aquí hay un ejemplo usando un array:

```
$data = array(
   array(
      'title' => 'Mi titulo' ,
      'name' => 'Mi Nombre' ,
      'date' => 'Mi fecha'
   ),
   array(
```

```
      'title' => 'Otro titulo' ,
      'name' => 'Otro Nombre' ,
      'date' => 'Otra fecha'
   )
);

$this->db->update_batch('mi_tabla', $data);

// Produces: INSERT INTO mi_tabla (titulo, nombre, fecha) VALUES ('Mi
titulo', 'Mi nombre', 'Mi fecha'), ('Otro titulo', 'Otro nombre',
'Otra fecha')
```

El primer parámetro contendrá el nombre de la tabla y el segundo es un array asociativo de valores.

**Nota:** Todos los valores pasados a esta función se escapan, produciendo consultas más seguras.

**$this->db->set();**

Esta función habilita permite establecer valores para insertar o actualizar.

**Puede ser usado en vez de pasar un arreglo de datos directamente a las funciones de insert o update:**

```
$this->db->set('nombre', $nombre);
$this->db->insert('mitabla');

// Produce: INSERT INTO mitabla (nombre) VALUES ('{$nombre}')
```

Si utiliza múltiples llamados a la función, ellos serán ensamblados apropiadamente basados en si está insertando o actualizando:

```
$this->db->set('nombre', $nombre);
$this->db->set('titulo', $titulo);
$this->db->set('estado', $estado);
$this->db->insert('mytable');
```

**set()** también aceptará un opcional tercer parámetro ($escape), que prevendrá datos de ser escapado si es establecido como FALSE. Para ilustrar la diferencia, aquí está set() usado con y sin el parámetro de escape.

```
$this->db->set('campo', 'campo+1', FALSE);
$this->db->insert('mitabla');
// resulta INSERT INTO mitabla (campo) VALUES (campo+1)

$this->db->set('campo', 'campo+1');
$this->db->insert('mitabla');
// resulta INSERT INTO mitabla (campo) VALUES ('campo+1')
```

También puede pasar un arreglo asociativo a esta función:

```
$arreglo = array('nombre' => $nombre, 'titulo' => $titulo, 'estado'
=> $estado);

$this->db->set($arreglo);
$this->db->insert('mitabla');
```

O un objeto:

```
/*
    class Miclase {
        var $titulo = 'Mi titulo';
        var $content = 'Mi Content';
        var $date = 'Mi Date';
    }
*/

$objeto = new Miclase;

$this->db->set($objeto);
$this->db->insert('mitabla');
```

**Actualizar Datos**

**$this->db->update();**

Genera una cadena de actualización y corre la consulta basado en los datos suministrados. Puede pasar una **arreglo** o un **objeto** a la función. Aquí hay un ejemplo, usando un arreglo:

```
$data = array(
               'titulo' => $titulo,
               'nombre' => $nombre,
               'fecha' => $fecha
            );

$this->db->where('id', $id);
$this->db->update('mitabla', $data);

// Produces:
// UPDATE mitabla
// SET titulo = '{$titulo}', nombre = '{$nombre}', fecha = '{$fecha}'
// WHERE id = $id
```

O puede suministrar un objeto:

```
/*
    class Miclase {
        var $titulo = 'My titulo';
        var $contenido = 'My Contenido';
        var $fecha = 'My Fecha';
    }
*/

$objeto = new Miclase;

$this->db->where('id', $id);
$this->db->update('mitabla', $objeto);
```

```
// Produce:
// UPDATE mitabla
// SET titulo = '{$titulo}', nombre = '{$nombre}', fecha = '{$fecha}'
// WHERE id = $id
```

**Nota:** Todos los valores son escapados automáticamente produciendo consulta más seguras.

Notará el uso de la función $this->db->where(), permitiendo establecer una cláusula WHERE. Opcionalmente, puede pasar información directamente como una cadena:

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

O como un arreglo:

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

También se puede usar la función $this->db->set() descripta anteriormente, cuando se efectúan actializaciones.

**Borrar Datos**

**$this->db->delete();**

Genera una cadena de eliminación SQL y ejecuta la consulta.

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

// Produce:
// DELETE FROM mitabla
// WHERE id = $id
```

El primer parámetro es el nombre de la tabla, el segundo la cláusula WHERE. También puede usar las funciones where() o or_where() en vez de pasar los datos como segundo parámetro de la función:

```
$this->db->where('id', $id);
$this->db->delete('mitabla');

// Produce:
// DELETE FROM mitabla
// WHERE id = $id
```

Un arreglo de nombres de tablas puede ser pasada a delete() si desea eliminar datos de más de una tabla.

```
$tablas = array('tabla1', 'tabla2', 'tabla3');
$this->db->where('id', '5');
$this->db->delete($tablas);
```

Si desea eliminar todos los datos de una tabla, puede usar la función truncate(), o empty_table().

**$this->db->empty_table();**

Genera una cadena SQL de eliminación y ejecuta la consulta.

```
$this->db->empty_table('mitabla');

// Produce:
// DELETE FROM mitabla
```

**$this->db->truncate();**

Genera una cadena SQL de truncado y ejecuta una consulta.

```
$this->db->from('mitabla');
$this->db->truncate();
// or
$this->db->truncate('mitabla');

// Produce:
// TRUNCATE mitabla
```

**Nota:** Si el comando TRUNCATE no está disponible, truncate() ejecutará "DELETE FROM tabla".

## Métodos en cadena

Métodos en cadena permite simplificar tu sintaxis conectando múltiples funciones. Considere este ejemplo:

```
$this->db->select('titulo')->from('mitabla')->where('id', $id)->limit(10, 20);

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

**Note:** Métodos en cadena sólo funciona con PHP 5.

## Active Record Caching

Mientras no es un verdadero cacheo, Active Record permite salvar (o "cache") ciertas partes de tus consultas para reusar después. Normalmente, cuando una llamada a Active Record es completada, toda la información es reinizializada para el próximo llamado. Con cacheo, puede prevenir esta reinicialización, y reusar la información fácilmente.

Las llamadas a cache son acumulables. Si ejecuta 2 llamadas a cached select(), y luego 2 llamadas a no cached select(), esto ersultará en 4 llamadas a select(). Hay tres funciones de cacheo disponibles:

**$this->db->start_cache()**

Esta función debe ser llamada para comenzar a cachear. Todas las consultas Active Record del tipo correcto (ver debajo para las consultas soportadas) son guardadas para uso posterior.

**$this->db->stop_cache()**

Esta función puede ser llamada para parar el cacheo.

**$this->db->flush_cache()**

Esta función elimina todos los items cacheados por la Active Record.

Aquí hay un ejemplo de uso:

```
$this->db->start_cache();
$this->db->select('campo1');
$this->db->stop_cache();
$this->db->get('tabla');
// Results in:
// SELECT `campo1` FROM (`tabla`)

$this->db->select('campo2');
$this->db->get('tabla');
// Results in:
// SELECT `campo1`, `campo2` FROM (`tabla`)

$this->db->flush_cache();

$this->db->select('campo2');
$this->db->get('tabla');
// Results in:
// SELECT `campo2` FROM (`tabla`)
```

**Nota:** Los siguientes campos pueden ser cacheados: 'select', 'from', 'join', 'where', 'like', 'groupby', 'having', 'orderby', 'set'

## REFERENCIAS

https://ellislab.com/codeigniter/user-guide/

http://escodeigniter.com/guia_usuario/

http://www.desarrolloweb.com/manuales/manual-codeigniter.html