

LIBRO CON APUNTES Y MARCADO CORRIGE PARTES

Revisado

admin , 09:20:03, 6-3-2017



CodeIgniter Guía del Usuario en Español Versión 2.1.3

Rev. 01 – 10/2012

Tabla de Contenido

Información Básica	5
Bienvenido a CodeIgniter	6
Acuerdo de licencia de CodeIgniter	7
Créditos	8
Instalación	9
Descargar CodeIgniter	10
Instrucciones de Instalación	11
Introducción	12
Primeros Pasos con CodeIgniter	13
CodeIgniter de un Vistazo	14
Funcionalidades de CodeIgniter	16
Diagrama de Flujo de la Aplicación	17
Modelo-Vista-Controlador	18
Objetivos de Diseño y Arquitectura	19
Tutorial	20
Introducción	21
Páginas Estáticas	22
Sección de Noticias	25
Crear Ítems de Noticias	29
Conclusión	32
Temas Generales	33
Las URLs de CodeIgniter	34
Controladores	36
Nombres Reservados	42
Vistas	43
Modelos	47
Funciones Helper	50
Usar las Bibliotecas de CodeIgniter	53
Crear sus Propias Bibliotecas	54
Usar Drivers de CodeIgniter	58
Crear Drivers	59
Crear Clases del Núcleo	60

Temas Generales (cont.)

Hooks - Extender el Núcleo del Framework	63
Carga Automática de Recursos	65
Funciones Comunes	66
Ruteo URI	68
Manejo de Errores	71
Almacenamiento en Caché de Páginas Web	73
Perfilar su Aplicación	74
Administrador Aplicaciones	76
Sintaxis Alternativa de PHP para Archivos de Vistas	78
Seguridad	80
Estilo y Sintaxis Generales	82
Escribir Documentación	93

Referencia de Clases **94**

Clase Benchmark	95
Clase Calendar	97
Clase Cart	101
Clase Config	106
Clase Database	109
Clase Email	158
Clase Encrypt	163
Clase Form_validation	166
Clase FTP	184
Clase Image_lib	188
Clase Input	196
Clase Javascript	201
Clase Lang	206
Clase Load	208
Clase Migration	212
Clase Output	214
Clase Pagination	217
Clase Parser	221
Clase Security	225
Clase Session	226
Clase Table	231
Clase TrackBack	236
Clase Typography	240
Clase Unit_test	242
Clase Upload	245

Referencia de Clases (cont.)

Clase URI	251
Clase User_agent	254
Clases XML-RPC y Servidor XML-RPC	258
Clase Zip	267
Driver de Almacenamiento en Caché	271

Referencia de Helpers 274

Helper Array	275
Helper CAPTCHA	280
Helper Cookie	280
Helper Date	281
Helper Directory	286
Helper Download	288
Helper Email	289
Helper File	290
Helper Form	293
Helper HMTL	301
Helper Inflector	307
Helper Language	309
Helper Number	310
Helper Path	311
Helper Security	312
Helper Smiley	313
Helper String	318
Helper Text	319
Helper Typography	322
Helper URL	323
Helper XML	328

Anexos 329

Anexo I: Actualizar desde una Versión Anterior	330
Anexo II: Registro de Cambios	350



Bienvenido a CodeIgniter

CodeIgniter es un framework para desarrollo de aplicaciones - un conjunto de herramientas - para gente que **construye sitios web** usando PHP. Su objetivo es permitirle desarrollar proyectos mucho más rápido que lo que podría hacer si escribiera el código desde cero, proveyéndole un rico conjunto de bibliotecas para tareas comunes, así como y una interfaz sencilla y una estructura lógica para acceder a esas bibliotecas. CodeIgniter le permite enfocarse creativamente en su proyecto al minimizar la cantidad de código necesaria para una tarea dada.

¿Para quien es CodeIgniter?

CodeIgniter es para Ud si:

- Necesita un framework con una pequeña impronta.
- Necesita un desempeño excepcional.
- Necesita amplia compatibilidad con cuentas estándar de alojamiento que corren una variedad de versiones de PHP y configuraciones.
- Necesita un framework que casi no necesite configuración.
- Necesita un framework que no le obligue a usar la línea de comandos.
- Necesita un framework que no le obligue a adquirir reglas de codificación restrictivas.
- No está interesado en bibliotecas monolíticas de gran tamaño como PEAR.
- No quiere verse forzado a aprender un lenguaje de plantillas (aunque hay un motor de plantillas disponible si desea uno).
- Evita la complejidad, favoreciendo las soluciones simples.
- Necesita una documentación clara y completa.

Requisitos del Servidor

- [PHP](#) versión 5.1.6 o más reciente.
- Se necesita una base de datos en la mayoría de los casos de programación de aplicaciones web. Las bases de datos que se soportan actualmente son MySQL (4.1+), MySQLi, MS SQL, Postgres, Oracle, SQLite, y ODBC.

Acuerdo de Licencia de CodeIgniter

Copyright (c) 2008 - 2011, EllisLab, Inc.
Todos los derechos reservados.

Esta licencia es un acuerdo legal entre Ud y EllisLab Inc. para el uso del software CodeIgniter (el "Software"). Al obtener el Software, Ud está de acuerdo a cumplir con los términos y condiciones de esta licencia.

Uso Permitido

Se le permite usar, copiar, modificar y distribuir el Software y su documentación, con o sin modificaciones, para cualquier propósito, siempre que se cumplan los siguientes requisitos:

1. Se incluya una copia de esta licencia con la distribución.
2. Las redistribuciones del código fuente deben conservar el aviso de copyright en todos los archivos de código fuente.
3. Las redistribuciones en formato binario deben reproducir el aviso de copyright en la documentación y/o otros materiales suministrados con la distribución.
4. Todos los archivos que han sido modificados deben llevar avisos indicando la naturaleza de los cambios y los nombres de aquellos que los cambió.
5. Los productos derivados del Software debe incluir un reconocimiento de que son derivados de CodeIgniter en su documentación y/o otros materiales suministrados con la distribución.
6. Los productos derivados del Software no se pueden llamar "CodeIgniter", ni puede aparecer "CodeIgniter" en su nombre, sin la previa autorización por escrito de EllisLab, Inc.

Indemnización

Usted acepta indemnizar y mantener inocentes a los autores del Software y todos los colaboradores por los daños directos, indirectos, incidentales o consecuentes reclamaciones de terceros, acciones o demandas, así como cualquier gasto, responsabilidades, daños, asentamientos o cargos derivados de su uso o mal uso del Software, o una violación de cualquiera de los términos de esta licencia.

Renuncia de Garantía

EL SOFTWARE SE PROVEE "COMO ESTÁ", SIN GARANTÍA DE NINGÚN TIPO, EXPRESA O IMPLÍCITA, INCLUYENDO, PERO NO LIMITADA A, GARANTÍAS DE CALIDAD, RENDIMIENTO, NO INFRACCIÓN, COMERCIALIZACIÓN O IDONEIDAD PARA UN PROPÓSITO PARTICULAR.

Limitaciones de Responsabilidad

UD ASUME TODOS LOS RIESGOS ASOCIADOS CON LA INSTALACIÓN Y USO DE ESTE SOFTWARE. EN NINGÚN CASO LOS AUTORES O TENEDORES DEL COPYRIGHT DEL SOFTWARE SERÁN RESPONSABLES POR RECLAMOS, DAÑOS O CUALQUIER OTRA RESPONSABILIDAD QUE SE PRESENTE DE O EN RELACIÓN CON EL SOFTWARE. LOS TENEDORES DE LA LICENCIA SON RESPONSABLES UNICAMENTE DE DETERMINAR LA CONVENIENCIA DEL USO Y ASUMIR TODOS LOS RIESGOS ASOCIADOS CON SU USO, INCLUYENDO PERO NO LIMITADO A LOS RIESGOS DE LOS ERRORES DEL PROGRAMA, DAÑO AL EQUIPO, PÉRDIDA DE DATOS O PROGRAMAS DE SOFTWARE, O LA FALTA DE DISPONIBILIDAD O INTERRUPCIÓN DE OPERACIONES.

Créditos

CodeIgniter fue desarrollado originalmente por [Rick Ellis](#) (CEO de [EllisLab, Inc.](#)). El framework se escribió para obtener buen rendimiento en el mundo real, donde muchas de las bibliotecas de clases, helpers, y subsistemas se tomaron prestados del código base de [ExpressionEngine](#).

Actualmente, es el Equipo de Desarrollo de ExpressionEngine quien lo desarrolla y mantiene. El desarrollo de vanguardia está encabezado por el selecto grupo de colaboradores del Reactor Team.

Una mención especial para Ruby on Rails por inspirarnos a crear un framework PHP, por poner los frameworks en la conciencia general de la comunidad web.

Acerca de la Guía de Usuario en Español

La traducción fue realizada en la primavera austral de 2012, liberándola con la esperanza de serle útil a quien la lea.

Octubre de 2012
Fernando "seacat" Velo
seacat.area@gmail.com

Esta Guía se distribuye con licencia



<http://creativecommons.org/licenses/by-sa/3.0/>

Convenciones Tipográficas

A lo largo de todo el texto se usan estas convenciones tipográficas:

Ejemplos de código

Notas, advertencias y otras cuestiones de importancia

Rutas y clases: application/config/routes.php

Funciones: \$this->agent->is_browser()

Variables y constantes: \$system_folder

Construcciones del lenguaje: <h3>

Expresiones destacadas



Instalación

Descargar CodeIgniter

La última versión de CodeIgniter se puede descargar de la página web <http://www.codeigniter.com>.

Instrucciones de Instalación

CodeIgniter se instala en cuatro pasos:

- Descomprima el paquete.
- Suba las carpetas y archivos de CodeIgniter al servidor. Normalmente el archivo **index.php** será la raíz.
- Abra el archivo **application/config/config.php** con un editor de texto y establezca su URL base. Si tiene intención de usar encriptación o sesiones, establezca su clave de encriptación.
- Si tiene intención de usar una base de datos, abra el archivo **application/config/database.php** con un editor de texto y establezca los parámetros de la base de datos.

Si desea incrementar la seguridad ocultando la ubicación de sus archivos de CodeIgniter, puede renombrar las carpetas **system** y **application** a algo más privado. Si los renombra, tendrá que abrir su archivo **index.php** principal y configurar las variables **\$system_folder** y **\$application_folder** con los nuevos nombres que haya elegido.

Para mayor seguridad, las carpetas **system** y cualquier **application** se tendrán que ubicar por encima de la raíz web para que no sean accesibles directamente mediante el navegador. Por defecto se incluyen en cada carpeta un archivo **.htaccess** para ayudar a evitar el acceso directo, pero es mejor sacarlos del acceso público completamente en caso de cambiar la configuración del servidor web o no soportar los archivos **.htaccess**.

Después de moverlos, abra su archivo **index.php** principal y configure las variables **\$system_folder** y **\$application_folder** preferiblemente con la ruta completa, por ejemplo '**/www/mi_usuario/system**'.

Una medida adicional para tomar en los entornos de producción es deshabilitar el reporte de error de PHP y cualquier otra funcionalidad dependiente exclusivamente del desarrollo. En CodeIgniter, esto se puede hacer configurando la constante **ENVIRONMENT**, que se describe mejor en la página de Seguridad.

¡Eso es todo!

Si Ud es nuevo en CodeIgniter, por favor lea la sección Primeros Pasos de la Guía del Usuario para comenzar a aprender como construir aplicaciones PHP dinámicas. ¡Disfrútelo!

Resolución de Problemas

Si encuentra que no importa lo que ponga en su URL solamente se carga la página por defecto, es posible que su servidor no soporte la variable **PATH_INFO** necesaria para entregar las URLs amigables con los motores de búsqueda. Como primer paso, abra el archivo **application/config/config.php** y busque la información **URI Protocol**. Se recomienda que pruebe un par de parámetros alternativos. Si aún no funciona después de haberlo intentado, necesitará forzar a CodeIgniter a agregar un signo de pregunta en sus URLs. Para hacer esto, abra el archivo **application/config/config.php** y cambie esto:

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

Por esto:

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



Introducción

Primeros Pasos con CodeIgniter

Cualquier software requiere de algún esfuerzo de aprendizaje. Hemos hecho todo lo posible para minimizar la curva de aprendizaje, mientras que el proceso sea lo más agradable posible.

El primer paso es instalar CodeIgniter, y luego leer todos los temas de la sección **Introducción** de la Tabla de Contenido.

Lo siguiente es leer cada una de las páginas de **Temas Generales** en orden. Cada tema se basa en el anterior e incluye ejemplos de código que se le anima a probar.

Una vez que comprendió lo básico, estará listo para explorar las páginas **Referencia de Clases** y **Referencia de Helpers** para aprender a usar las bibliotecas nativas y los archivos de helpers.

Siéntase libre de aprovechar nuestros [Foros de la Comunidad](#) si tiene preguntas o problemas y nuestra Wiki para ver ejemplos de código publicados por otros usuarios.

CodeIgniter de un Vistazo

CodeIgniter es un Framework para Aplicaciones

CodeIgniter es un conjunto de herramientas para gente que construyen aplicaciones web usando PHP. Su objetivo es permitirle desarrollar proyectos mucho más rápido que lo que podría hacer si escribiera código desde cero, al proveer un rico conjunto de bibliotecas para tareas comúnmente necesarias, tanto como una interfaz sencilla y una estructura lógica para acceder a esas bibliotecas. CodeIgniter le permite enfocarse creativamente en su proyecto al minimizar la cantidad de código necesario para una tarea dada.

CodeIgniter es Libre

CodeIgniter está liberado bajo licencias open source del estilo Apache/BSD, así que puede usarlo donde desee. Para mayor información lea el acuerdo de licencia.

CodeIgniter es Liviano

Es realmente liviano. El núcleo del sistema sólo requiere algunas bibliotecas muy pequeñas. Esto está en marcado contraste con muchos frameworks que requieren muchos más recursos. Las bibliotecas adicionales se cargan dinámicamente bajo pedido, basado en sus necesidades para un proceso dado, por lo que el sistema base es muy ligero y bastante rápido.

CodeIgniter es Rápido

Es verdaderamente rápido. Lo desafiamos a que encuentre un framework que tenga mejor desempeño que CodeIgniter.

CodeIgniter Usa M-V-C

CodeIgniter usa el enfoque Modelo-Vista-Controlador, que permite una gran separación entre la lógica y la presentación. Es particularmente bueno para proyectos en los que los diseñadores trabajan en sus archivos de plantillas, ya que el código en estos archivos será mínimo. Describimos MVC en más detalle en su propia página.

CodeIgniter Genera URLs Claras

Las URLs generadas por CodeIgniter son claras y amigables con los motores de búsqueda. En lugar de usar el enfoque estándar "query string" característico de sistemas dinámicos, CodeIgniter usa el enfoque basado en segmentos:

ejemplo.com/noticias/articulo/345

Nota: Por defecto el archivo **index.php** está incluido en la URL pero se puede quitarlo usando un sencillo archivo **.htaccess**.

CodeIgniter Trae un Montón de Paquetes

CodeIgniter viene con una gama completa de bibliotecas que facilitan las tareas de desarrollo web más comúnmente usadas, como acceso a base de datos, envío de correo electrónico, validación de datos de formularios, manejo de sesiones, manipulación de imágenes, trabajo con datos XML-RPC y mucho más.

CodeIgniter es Extensible

El sistema se puede extender fácilmente a través de sus propias bibliotecas, helpers, extensiones de clases o

sistema de hooks.

CodeIgniter No Necesita un Motor de Plantillas

Aunque CodeIgniter viene con un sencillo motor de plantillas que se puede usar opcionalmente, Ud no está forzado a usar uno. Los Motores de Plantillas simplemente no pueden igualar el desempeño del PHP nativo, y la sintaxis que hay que aprender para usar un motor de plantillas normalmente es solo marginalmente más fácil que aprender los fundamentos de PHP. Considere este bloque de código PHP:

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

Compárelo con el seudo-código usado por un motor de plantillas:

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

Sí, el ejemplo del motor de plantillas es un poco más claro, pero viene con el precio del desempeño ya que hay que convertir el seudo-código de vuelta en PHP para ejecutarlo. Como uno de nuestros objetivos es *máximo* desempeño, optamos por no obligar a usar un motor de plantillas.

CodeIgniter está Completamente Documentado

Los programadores aman programar y odian escribir documentación. No somos diferentes, por supuesto, pero como la documentación es **tan importante** como el código en sí mismo, estamos comprometidos a hacerlo. Nuestro código fuente es extremadamente claro y bien comentado también.

CodeIgniter tiene una Amigable Comunidad de Usuarios

Puede encontrar a nuestra creciente comunidad de usuarios participando de nuestros [Foros de la Comunidad](#).

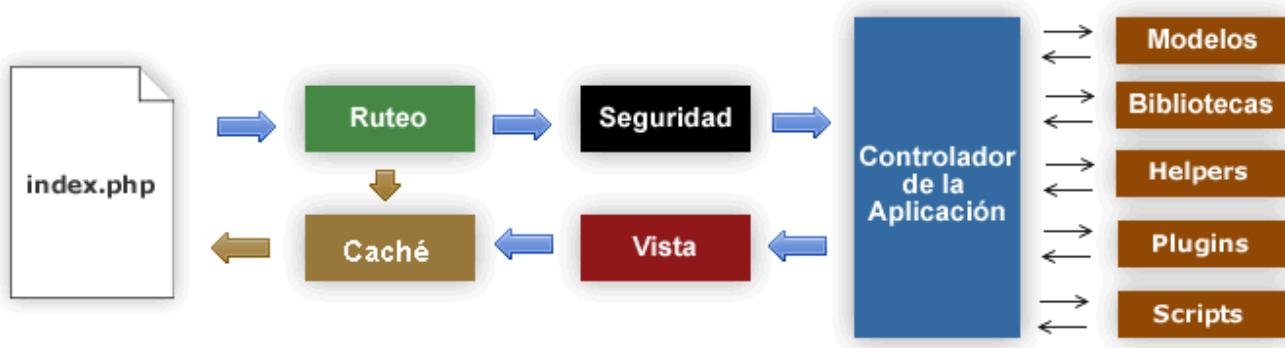
Funcionalidades de CodeIgniter

Las funcionalidades por sí mismas son una forma muy pobre de juzgar a una aplicación, ya que no dicen nada acerca de la experiencia del usuario, o si está diseñada intuitiva e inteligentemente. Las funcionalidades no revelan nada acerca de la calidad del código o su rendimiento, atención a los detalles o las prácticas de seguridad. La única forma real de juzgar una aplicación es probarla y llegar a conocer su código. Instalar CodeIgniter es un juego de niños, por lo que lo animamos a hacerlo. Por el momento, esta es la lista de funcionalidades principales de CodeIgniter.

- Sistema basado en Modelo-Vista-Controlador
- Extremadamente liviano
- Clases de base de datos con soporte para varias plataformas
- Soporte para base de datos con Active Record
- Validación de datos y formularios
- Seguridad y filtrado XSS
- Administración de sesiones
- Clase para enviar Email. Soporta adjuntos, Email de HTML/Texto, varios protocolos (sendmail, SMTP, y Mail) y más.
- Biblioteca de manipulación de imágenes (recorte, redimensión, rotación, etc). Soporta GD, ImageMagick, y NetPBM
- Clase para subir archivos
- Clase para FTP
- Localización
- Paginación
- Encriptación de datos
- Evaluación de rendimiento
- Caché de página completa
- Historial de errores
- Perfilado de la aplicación
- Clase para Calendarios
- Clase para Agente del Usuario
- Clase para codificación Zip
- Clase de motor de plantillas
- Clase para Trackback
- Biblioteca XML-RPC
- Clase para pruebas de unidad
- URLs amigables para los motores de búsqueda
- Ruteo URI flexible
- Soporte para hooks y extensiones de clase
- Amplia biblioteca de funciones "helper"

Diagrama de Flujo de la Aplicación

El siguiente gráfico ilustra como los datos fluyen a través del sistema:



1. El **index.php** sirve como el controlador frontal, inicializando los recursos básicos que necesita CodeIgniter para ejecutar.
2. El Ruteador examina la solicitud HTTP para determinar que debería hacer con ella.
3. Si existe el archivo de caché, se lo envía directamente al navegador, sin pasar por la ejecución normal del sistema.
4. Seguridad. Antes que se cargue el controlador de la aplicación, por razones de seguridad se filtran la solicitud HTTP y cualquier otro dato enviado por el usuarios.
5. El controlador carga el modelo, las bibliotecas del núcleo, helpers, y cualquier otro recurso requerido para procesar una solicitud específica.
6. La Vista terminada se procesa y se envía al navegador para que se pueda ver. Si el caché está habilitado, la vista se cachea primero para que las siguientes solicitudes que la necesiten puedan ser servidas.

Modelo-Vista-Controlador

CodeIgniter está basado en el patrón de desarrollo Modelo-Vista-Controlador. MVC es un enfoque de software que separa la lógica de la aplicación de la presentación. En la práctica, le permite a sus páginas web contener mínimo código ya que la presentación está separada del código PHP.

- El **Modelo** representa sus estructuras de datos. Tipicamente sus clases del modelo contendrán funciones que los ayudarán a devolver, insertar y actualizar información de su base de datos.
- La **Vista** es la información que se presenta al usuario. Una vista será normalmente una página web, pero en CodeIgniter, una vista también puede ser un fragmento de página como el encabezado o pie de página. También puede ser una página RSS, o cualquier otro tipo de "página".
- El **Controlador** sirve como un intermediario entre el Modelo, la Vista y cualquier otro recurso necesario para procesar la solicitud HTTP y generar una página web.

CodeIgniter tiene un enfoque bastante poco estricto de MVC, ya que los Modelos no son obligatorios. Si no necesita la separación añadida o encuentra que mantener modelos requiere más complejidad de la que quiere, puede ignorarlos y construir su aplicación mínimamente usando Controladores y Vistas. CodeIgniter también le permite incorporar sus propios scripts, o inclusive desarrollar bibliotecas del núcleo para el sistema, permitiéndole trabajar en una forma que tenga más sentido para Ud.

Objetivos de Diseño y Arquitectura

Nuestro objetivo para CodeIgniter es **máximo desempeño, capacidad y flexibilidad en el paquete más pequeño y ligero posible.**

Para alcanzar esta meta nos comprometemos a usar evaluaciones de desempeño, a refactorizar y simplificar cada paso del proceso de desarrollo rechazando cualquier cosa que nos aleje del objetivo establecido.

Desde el punto de vista técnico y arquitectónico, CodeIgniter se creó con los siguientes objetivos:

- **Instaciación Dinámica.** En CodeIgniter, los componentes se cargan y las rutinas se ejecutan solamente cuando se necesita, en lugar de hacerlo globalmente. No se hacen suposiciones para el sistema respecto a lo que puede ser necesario más allá de los mínimos recursos del núcleo, por lo que el sistema es muy liviano por defecto. Los eventos, disparados por la solicitud HTTP, los controladores y las vistas que Ud diseñe determinarán lo que se invoque.
- **Poco Acoplamiento.** El acoplamiento es el grado en que los componentes de un sistema dependen unos de otros. Mientras menos componentes dependan unos de otros, más reusable y flexible será el sistema. Nuestro objetivo fue un sistema muy poco acoplado.
- **Singularidad de Componentes.** La singularidad es el grado en que los componentes tienen un propósito muy específico. En CodeIgniter, cada clase y sus funciones son muy autónomas a fin de permitir la máxima utilidad.

CodeIgniter es un sistema instanciado dinámicamente, poco acoplado con alta singularidad de componentes. Se esfuerza por ser simple, flexible y tener alto desempeño en un paquete de tamaño reducido.



Introducción

Este tutorial está pensado para presentarle el framework CodeIgniter y los principios básicos de la arquitectura MVC. Le mostrará cómo se construye paso a paso una aplicación CodeIgniter básica.

En este tutorial Ud creará una **aplicación básica de noticias**. Comenzará escribiendo el código que carga páginas estáticas. Luego creará una sección de noticias que lee ítems de noticias desde una base de datos. Finalmente, agregará un formulario para crear ítems de noticias a la base de datos.

Este tutorial se enfocará principalmente en:

- Fundamentos de Modelo-Vista-Controlador
- Fundamentos del Ruteo
- Validación de formularios
- Ejecución de consultas básica de base de datos usando el "Active Record"

El tutorial completo está separado en varias páginas, cada una explicando una parte pequeña del framework CodeIgniter. Visitará las siguientes páginas:

- Introducción, esta página, que le da un vistazo general de lo que viene.
- Páginas Estáticas, que le enseña lo básico de los controladores, vistas y ruteo.
- Sección de Noticias, donde empezará a usar los modelos y hará algunas operaciones básicas de base de datos.
- Crear Ítems de Noticias, que le mostrará operaciones más avanzadas de base de datos y validación de formularios.
- Conclusión, que le dará algunas indicaciones de lecturas posteriores y otros recursos.

Disfrute la explicación del framework CodeIgniter.

Páginas Estáticas

Note: Este tutorial asume que ya descargó e instaló el framework CodeIgniter en su entorno de desarrollo.

La primera cosa que hará es configurar un controlador para manejar páginas estáticas. Un controlador es simplemente una clase que ayuda a delegar trabajo. Es el núcleo de su aplicación web.

Por ejemplo, cuando se hace una llamada a:

```
http://ejemplo.com/noticias/ultimas/10
```

Podemos imaginar que hay un controlador llamado **noticias**. El método que se llama en **noticias** sería **ultimas**. El trabajo del método de **noticias** podría ser tomar **10** ítems de noticias y mostrarlos en la página. Muy a menudo en MVC, verá patrones de URL que con coincidan con:

```
http://ejemplo.com/[clase-controlador]/[método-controlador]/[argumentos]
```

Cuando el esquema de URL se hace más complejo, esto puede cambiar. Pero por ahora, esto es todo lo que necesitamos saber.

Crear un archivo en **application/controllers/pages.php** con el siguiente código:

```
<?php  
  
class Pages extends CI_Controller {  
  
    public function view($page = 'home')  
    {  
    }  
}
```

Se creó una clase llamada **Pages**, con un **método view** que acepta un **solo argumento** llamado **\$page**. La **clase Pages** se **extiende** de la **clase CI_Controller**. Esto **significa** que la **nueva clase Pages** **puede acceder** a los **métodos y variables definidos** en la clase **CI_Controller (system/core/Controller.php)**.

El controlador es lo que se convertirá en el centro de cada solicitud de su aplicación web. En discusiones muy técnicas de CodeIgniter, esto puede **referirse** como el **super objeto**. Como **cualquier otra clase de PHP**, se **referirá a ella dentro** de sus **controladores** como **\$this**. Referirse a **\$this** es como cargará bibliotecas, vistas y como generalmente utilizará el framework.

Ahora que ha creado su primer método, es momento de hacer alguna plantilla básica de página. Crearemos dos "vistas" (plantillas de páginas) que actúan como su encabezado y pie de página.

Cree el encabezado en **application/views/templates/header.php** y agregarle este código:

```
<html>  
<head>  
    <title><?php echo $title ?> - Tutorial de CodeIgniter 2</title>  
</head>  
<body>  
    <h1>Tutorial de CodeIgniter 2</h1>
```

El encabezado contiene código HTML básico que mostrará antes de cargar la vista principal, junto al encabezado. También mostrará la variable **\$title**, que definiremos luego en el controlador. Ahora, cree el pie de página en **application/views/templates/footer.php** e incluya este código:

```
<strong>&copy; 2012</strong>
</body>
</html>
```

Agregar lógica al controlador

Con anterioridad, establecimos un controlador con el método **view()**. El método acepta un parámetro, que es el nombre de la página a cargar. Las plantillas de páginas estáticas se localizarán en el directorio **application/views/pages/**.

En ese directorio, cree dos archivos llamados **home.php** y **about.php**. Dentro de ellos escriba algún texto – lo que quiera – y guárdelos. Si no quiere ser particularmente original, escriba "Hola Mundo!".

Para cargar esas páginas, tiene que verificar si la página solicitada realmente existe:

```
public function view($page = 'home')
{
    function ruta
    if ( ! file_exists('application/views/pages/'.$page.'.php') )
    {
        // Oh, oh... no tenemos una pagina para esto!
        show_404();
    }

    $data['title'] = ucfirst($page); // Capitaliza la primera letra

    $this->load->view('templates/header', $data);
    $this->load->view('pages/'.$page, $data);
    $this->load->view('templates/footer', $data);
}
```

Ahora, cuando la página exista, se la cargará incluyendo el encabezado y el pie y se la mostrará al usuario. Si la página no existe, se mostrará el error "404 Page not found" (página no encontrada).

La primera línea de este método verifica si la página existe realmente. Se usa la función nativa de PHP **file_exists()** para verificar si el archivo está donde se espera que esté. **show_404()** es una función interna de CodeIgniter que presenta la página de error por defecto.

En la plantilla del encabezado, la variable **\$title** se usa para personalizar el título de la página. El valor de **\$title** se define en el método, pero en lugar de asignarle el valor a una variable, se asigna al elemento **title** en el array **\$data**.

La última cosa que tiene que hacer es cargar las vistas en el orden en que se mostrarán. El segundo parámetro en el método **view()** se usa para pasar valores a la vista. Cada valor en el array **\$data** se asigna a una variable con el nombre de su clave. Por lo tanto, el valor de **\$data['title']** en el controlador es equivalente a **\$title** en la vista.

Ruteo

El controlador ahora está funcionando! Apunte su navegador a **[url-de-su-sitio]index.php/pages/view** para ver la página. Al visitar **index.php/pages/view/about** verá la página acerca, incluyendo el encabezado y el pie de página.

Al usar **reglas de ruteo personalizadas**, Ud tiene el poder de **mapear cualquier URI a cualquier controlador y método**, liberándose de la convención normal:

```
http://ejemplo.com/[clase-controlador]/[método-controlador]/[argumentos]
```

Hagamos eso. Abra el archivo de ruteo localizado en **application/config/routes.php** y agregue las siguientes dos líneas. Quite el resto del código y establezca cualquier elemento en el array **\$route**.

```
$route['default_controller'] = 'pages/view';
$route['(:any)'] = 'pages/view/$1';
```

Este comodín sirve para cualquier cadena, con cualquier carácter.

CodeIgniter lee sus **reglas de ruteo de arriba hacia abajo** y rutea las solicitudes a la primer **regla** que **coincide**. Cada **regla** es una **expresión regular** (lado izquierdo) **mapeada al nombre de controlador y método separados por barras** (lado derecho). Cuando ingresa una solicitud, CodeIgniter busca la primer coincidencia, y llama al controlador y métodos adecuados, posiblemente con argumentos.

Puede encontrar más información acerca del ruteo en la documentación del Ruteo URI.

Aquí, la segunda regla en el array **\$routes** coincide con cualquier solicitud que use el comodín **(:any)** y pasa el parámetro al método **view()** de la clase **Pages**.

Ahora visite **index.php/about**. ¿Se ruteó correctamente al método **view()** en el controlador de páginas? ¡Maravilloso!

(:num)

Este comodín sirve para cualquier cadena que contenga solamente dígitos numéricos

Toda la potencia de las expresiones regulares la podemos aplicar a CodeIgniter para crear reglas de enrutamiento tan complejas como deseemos.

```
$route['manuales/([0-9]+)'] = "manuales/muestra/$1";
```

Si detectamos en la primera parte la palabra **manuales** y en el segundo segmento un conjunto de dígitos numéricos, se enrutaría a través del controlador **manuales**, el método **muestra**, con el parámetro indicado como dígitos numéricos.

```
$route['espana/([a-z_-]+)/([a-z_-]+)'] = "localidad/$1/$2";
```

Esta regla de enrutamiento, un poco más compleja, serviría para detectar la palabra "espana" en el primer segmento y luego dos segmentos adicionales, con sendas expresiones regulares que aceptarían letras de la "a" a la "z" con el guión bajo o medio, repetidas una o más veces. Las enrutaría a través del controlador "localidad" y luego, el método especificado en la primera expresión y el parámetro indicado en la segunda expresión regular

Sección de noticias

En la última sección, vimos algunos conceptos básicos del framework para escribir una clase que incluía páginas estáticas. También reformamos la URI al agregar reglas de ruteo personalizadas. Ahora es momento de introducir contenido dinámico y comenzar a usar la base de datos.

Configurar su modelo

En lugar de escribir operaciones de base de datos directamente en el controlador, las consultas deberían ubicarse en un modelo, para que se puedan reusar más tarde con facilidad. Los **modelos** son el lugar donde se **devuelve**, **inserta** y **actualiza** la **información** de la **base de datos** u **otros almacenamientos**. Ellos representan sus datos.

Abra el directorio **application/models**, cree un archivo nuevo llamado **news_model.php** y agregue el siguiente código. Asegúrese de haber configurado adecuadamente su base de datos como se describe en la sección Configuración de la Base de Datos.

```
<?php
class News_model extends CI_Model {
    public function __construct()
    {
        $this->load->database();
    }
}
```

Este código se ve similar al código del controlador que usamos antes. Crea un nuevo modelo al extender **CI_Model** y carga la biblioteca database. Esto **hace** que la **clase Database** esté disponible mediante el **puntero** **\$this->db**.

pseudo-variable objeto

Antes de consultar la base de datos, se tiene que crear un esquema de base de datos. Conecte a su base de datos y ejecute los comandos SQL siguientes. Agregue también algunos registros.

```
CREATE TABLE news (
    id int(11) NOT NULL AUTO_INCREMENT,
    title varchar(128) NOT NULL,
    slug varchar(128) NOT NULL,
    text text NOT NULL,
    PRIMARY KEY (id),
    KEY slug (slug)
);
```

Ahora que se configuraron la base de datos y el modelo, necesitará un **método** para **obtener** todos los **mensajes** **desde** la **base** de **datos**. Para hacer esto se usa la **capa de abstracción de base de datos** que se incluye con CodeIgniter — el **Active Record**. Éste hace posible escribir sus “consultas” una vez y usarlas luego en todos los sistemas soportados de bases de datos. Agregue este código a su modelo.

```
public function get_news($slug = FALSE)
{
    if ($slug === FALSE)
    {
        $query = $this->db->get('news');
        return $query->result_array();
    }

    $query = $this->db->get_where('news', array('slug' => $slug));
```

```

        return $query->row_array();
    }
}

```

Con este código puede realizar dos consultas diferentes. Puede obtener todos los registros de noticias, o bien obtener un ítem de noticias mediante su identificador. Tiene que advertir que la variable \$slug no se descontaminó antes de ejecutar la consulta; el Active Record hace esto por Ud.

Mostrar las noticias

Ahora que se escribieron las consultas, el modelo debería vincularse a las vistas que van a mostrar los ítems de noticias al usuario. Esto se puede hacer en su controlador de páginas creado anteriormente, pero en aras de la claridad, definimos un nuevo controlador news. Cree el nuevo controlador en application/controllers/news.php.

```

<?php
class News extends CI_Controller {

    public function __construct()
    {
        parent::__construct();
        $this->load->model('news_model');
    }

    public function index()
    {
        $data['news'] = $this->news_model->get_news();
    }

    public function view($slug)
    {
        $data['news'] = $this->news_model->get_news($slug);
    }
}

```

Mirando el código, puede verse alguna similitud con los archivos que creamos anteriormente. Primero, el método **__construct**: llama al constructor de su clase padre (CI_Controller) y carga el modelo, por lo que se lo puede usar en todos los demás métodos de este controlador.

Lo siguiente son los dos métodos para ver todos los ítems de noticias y uno para ver un ítem específico. Puede ver que la variable \$slug se pasa al método del modelo en el segundo método. El modelo está usando esta variable para identificar el ítem de noticias a devolver.

Ahora, el controlador devuelve los datos mediante el modelo, pero todavía no se muestra nada. La siguiente cosa a hacer es pasar estos datos a las vistas.

```

public function index()
{
    $data['news'] = $this->news_model->get_news();
    $data['title'] = 'Archivo de noticias';

    $this->load->view('templates/header', $data);
    $this->load->view('news/index', $data);
    $this->load->view('templates/footer');
}

```

El código anterior obtiene todos los registros de noticias desde el modelo y lo asigna a una variable. El valor para el

título también se asigna al elemento `$data['title']` y todos los datos se pasan a las vistas. Ahora necesita crear una vista para presentar los ítems de noticias. Cree `application/views/news/index.php` y agregue la siguiente porción de código.

```
<?php foreach ($news as $news_item): ?>

    <h2><?php echo $news_item['title'] ?></h2>
    <div id="main">
        <?php echo $news_item['text'] ?>
    </div>
    <p><a href="news/<?php echo $news_item['slug'] ?>">Ver artículo</a></p>

<?php endforeach ?>
```

Aquí, cada ítem de noticias se cicla y muestra al usuario. Puede ver que escribimos nuestra plantilla en una mezcla de PHP y HTML. Si prefiere usar un lenguaje de plantillas, puede usar la clase del Analizador de Plantillas de CodeIgniter o un analizador de terceras partes.

[Template Parser Class](#)

Ya está hecha la página que muestra las noticias en general, pero todavía falta una página para mostrar ítems de noticias individuales. El modelo creado anteriormente se hizo de tal forma que se puede usar fácilmente para esta funcionalidad. Solo tiene que agregar algo de código al controlador y crear una nueva vista. Regrese al controlador de noticias y agregue estas líneas de código al archivo.

```
public function view($slug)
{
    $data['news_item'] = $this->news_model->get_news($slug); objeto de la clase modelo

    if (empty($data['news_item']))
    {
        show_404();
    }
    esto es un array
    $data['title'] = $data['news_item']['title'];

    $this->load->view('templates/header', $data);
    $this->load->view('news/view', $data);
    $this->load->view('templates/footer');
}
```

En lugar de llamar al método `get_news()` sin parámetros, se pasa la variable `$slug` para que devuelva un ítem de noticias específico. La única cosa que queda por hacer es crear la vista correspondiente en `application/views/news/view.php`. Ponga el siguiente código en un archivo.

```
<?php
echo '<h2>'.$news_item['title'].'</h2>';
echo $news_item['text'];
```

Ruteo

Debido a la regla de ruteo con comodín creada anteriormente, se necesita una ruta adicional para ver el controlador que acabamos de hacer. Modifique el archivo de ruteo (`application/config/routes.php`) para que se vea como se muestra a continuación. Esto asegura que las solicitudes alcancen al controlador de noticias, en lugar de ir directamente al controlador de páginas. Las primera línea rutea las URI's con un slug al método `view()` en el controlador `news`.

[Slug -> título de la noticia](#)

```
$route['news/(:any)'] = 'news/view/$1';
$route['news'] = 'news';
$route['(:any)'] = 'pages/view/$1';
$route['default_controller'] = 'pages/view';
```

Apunte su navegador a la raíz de documentos seguido de **index.php/news** y vea sus páginas de noticias.

Crear ítems de noticias

Ahora Ud sabe como leer datos desde una base de datos usando CodeIgniter, pero todavía no escribió ninguna información en ella. En esta sección expandiremos nuestros controlador de noticias y modelo creados anteriormente para incluir esta funcionalidad.

Crear un formulario

Para ingresar datos en la base de datos, necesita **crear** un **formulario** donde pueda **ingresar** la **información** a **almacenarse**. Esto significa que necesitaremos un **formulario** con **dos** **campos**, **uno** para el **título** y **otro** para el **texto**. **Derivaremos** el **identificador** de **nuestro** **título** en el **modelo**. Cree la nueva vista en **application/views/news/create.php**.

```
<h2>Create a news item</h2>
  informa errores de validacion del formulario
<?php echo validation_errors(); ?>
  agrega funcionalidad campo oculto
<?php echo form_open('news/create') ?>

  <label for="title">Título</label>
  <input type="input" name="title" /><br />

  <label for="text">Texto</label>
  <textarea name="text"></textarea><br />

  <input type="submit" name="submit" value="Crear ítem de noticias" />

</form>
```

Los datos del formulario luego se envia al hacer submit al Controlador New función create

Las únicas dos cosas aquí que probablemente no se vean familiares son: las **funciones** **form_open()** y **validation_errors()**.

form_open()
La **primera función** la provee el **helper form** y presenta el **elemento form** y **agrega funcionalidad adicional**, como un **campo oculto** para **evitar** la **CSRF** (del inglés Cross-Site Request Forgery o falsificación de solicitud de sitios cruzados). La **última** se usa para **informar errores** relacionados a la **validación del formulario**.

Vuelva atrás a su controlador de noticias. Vamos a hacer dos cosas aquí, **verificar** si el **formulario** fue **enviado** y si los **datos enviados pasaron** las **reglas de validación**. Usaremos la **biblioteca de validación de formularios** para hacer esto.

```
public function create()
{
    $this->load->helper('form');
    $this->load->library('form_validation');

    $data['title'] = 'Crear un ítem de noticias';
    campos de entrada Mensaje de entrada
    $this->form_validation->set_rules('title', 'Título', 'required');
    $this->form_validation->set_rules('text', 'Texto', 'required');

    if ($this->form_validation->run() === FALSE)
    {
        $this->load->view('templates/header', $data);
        $this->load->view('news/create');
        $this->load->view('templates/footer');

    }
}
```

```

    else
    {
        $this->news_model->set_news();
        $this->load->view('news/success');
    }
}

```

El código anterior agrega un montón de funcionalidad. Las primeras líneas cargan el helper form y la biblioteca de validación de formularios. Después de eso, se establecen las reglas para la validación del formulario. El método **set_rules()** toma tres argumentos: el nombre del campo de entrada, el nombre a usarse en los mensajes de error y la regla. En este caso, los campos título y texto son obligatorios.

CodeIgniter tiene una poderosa biblioteca de validación de formularios como se demostró antes. Puede leer más acerca de esta biblioteca en la sección correspondiente a la clase **Form_validation**.

Más abajo, se puede ver una condición que verifica si la validación del formulario se ejecutó correctamente. Si no, se muestra el formulario, si se envió y se pasaron todas las reglas, se llama al modelo. Después de esto, se carga una vista para mostrar el mensaje de éxito. Cree una vista en **application/view/news/success.php** y escriba un mensaje de éxito.

Modelo

La única cosa que queda es escribir un método que escriba los datos en la base de datos. Usaremos la clase **Active Record** para insertar la información y usaremos la biblioteca **Input** para obtener los datos publicados. Abra el modelo creado anteriormente y agregue lo siguiente:

```

public function set_news()
{
    $this->load->helper('url');
    // desarma la cadena y la reemplaza por guiones y todos en minúscula
    $slug = url_title($this->input->post('title'), 'dash', TRUE);

    $data = array(
        'title' => $this->input->post('title'),
        'slug' => $slug,
        'text' => $this->input->post('text')
    );

    return $this->db->insert('news', $data);
}

```

Este nuevo método se encarga de insertar el ítem de noticias en la base de datos. La tercera línea contiene una nueva función, **url_title()**. Esta función – provista por el helper URL – desarma la cadena que se le pasa, reemplazando todos los espacios por guiones (-) y asegurando que todos los caracteres estén minúsculas. Esto le deja un bonito identificador, perfecto para crear URIs.

Continuemos con la preparación del registro que se insertará luego, dentro del array **\$data**. Cada elemento corresponde con una columna en la tabla de la base de datos creada anteriormente. Aquí puede advertir un nuevo método llamado **post()** de la biblioteca **Input**. Este método se asegura que los datos se descontaminen, protegiéndolo a Ud de desagradables ataques de terceros. La biblioteca **Input** se carga por defecto. Por último, insertamos nuestro array **\$data** en la base de datos.

Ruteo

Antes que pueda comenzar a agregar ítems de noticias en su aplicación CodeIgniter, Ud tiene que agregar una regla adicional en el archivo **config/routes.php**. Asegúrese que el archivo contenga lo siguiente. Esto asegura

que CodeIgniter vea a 'create' como un método, en lugar de un slug de ítem de noticias.

```
$route['news/create'] = 'news/_create';
$route['news/(:any)'] = 'news/view/$1';
$route['news'] = 'news';
$route['(:any)'] = 'pages/view/$1';
$route['default_controller'] = 'pages/view';
```

Ahora apunte su navegador a su entorno de desarrollo local donde instaló CodeIgniter y agregue **index.php/news/create** a la URL. ¡Felicitaciones, acaba de crear su primera aplicación CodeIgniter! Agregue alguna noticia y consulte las distintas páginas que creó.

Conclusión

Este tutorial no cubre todas las cosas que puede esperar de un sistema de administración de contenidos completamente desarrollado, pero le presenta los tópicos más importantes de ruteo, escritura de controladores y modelos. Esperamos que este tutorial le haya dado una idea de algunos de los patrones de diseño básicos de CodeIgniter, los cuales puede ampliar.

Ahora que completó este tutorial, le recomendamos que consulte el resto de la documentación. CodeIgniter es a menudo elogiado por su amplia documentación. Use esto como una ventaja y lea a fondo las secciones "Introducción" y "Temas Generales". Debería leer las referencias de clases y helpers cuando las necesite.

Cualquier programador PHP intermedio debería ser capaz de encontrarle la vuelta a CodeIgniter en pocos días.

Si todavía tiene preguntas acerca del framework o de su propio código CodeIgniter, puede:

- Consultar nuestro [foro](#)
- Visitar nuestra [sala de chat IRC](#)
- Explorar nuestra [Wiki](#)



Temas Generales

Las URLs de CodeIgniter

Por defecto, las URLs en CodeIgniter se diseñan para ser amigables con los motores de búsqueda y las personas. En lugar de usar el enfoque estándar de las "query string" para las URLs que es sinónimo de sistemas dinámicos, CodeIgniter usa el enfoque **basado en segmentos**:

ejemplo.com/noticias/articulo/mi_articulo

Nota: Como se verá más adelante, opcionalmente se pueden habilitar las query strings de URLs.

Segmentos URI

Siguiendo el enfoque Modelo-Vista-Controlador, los segmentos en la URL normalmente representan:

ejemplo.com/clase/función/ID

1. El primer segmento representa la **clase** del controlador que se debería invocar.
2. El segundo segmento representa la **función** de la clase, o método que se debería llamar.
3. El tercer y cualquier otro segmentos adicionales, representa el ID y cualquier variable que se pasará al controlador.

La Clase URI y el Helper de URL contienen funciones que hacen fácil trabajar con datos de URI. Además para mayor flexibilidad, sus URLs se pueden remapear usando la funcionalidad de Ruteo de URI.

Quitar el archivo index.php

Por defecto, el archivo **index.php** estará incluido en sus URLs:

ejemplo.com/index.php/noticias/articulo/mi_articulo

Se puede quitar fácilmente este archivo usando un archivo **.htaccess** con algunas reglas simples. Aquí hay un ejemplo de tal archivo, usando el método "negativo" donde todo se redirecciona excepto los ítems especificados:

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

En el ejemplo anterior, cualquier solicitud HTTP distinta de **index.php**, imágenes, y **robots.txt** se trata como una solicitud a su archivo **index.php**.

Agregar un Sufijo a una URL

En su archivo **application/config/config.php** puede especificar un sufijo que se agregará a todas las URLs generadas por CodeIgniter. Por ejemplo, si tiene esta URL:

ejemplo.com/index.php/productos/ver/zapatos

Puede agregar opcionalmente un **sufijo**, tal como **.html**, haciendo que la página parezca ser de un cierto tipo:

```
ejemplo.com/index.php/productos/ver/zapatos.html
```

Habilitar las Query Strings

En algunos casos puede preferir usar las URLs con query strings:

```
index.php?c=productos&m=ver&id=345
```

CodeIgniter soporta opcionalmente esta capacidad, que se puede habilitar en su archivo **application/config/config.php**. Si abre su archivo de configuración verá estos ítems:

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

Si cambia "enable_query_strings" a **TRUE** esta funcionalidad se activará. Entonces, sus controladores y funciones estarán accesibles usando la palabra "trigger" que Ud estableció para invocar a sus controladores y métodos:

```
index.php?c=controlador&m=metodo
```

Por favor advierta: Si está usando query strings tendrá que armar sus propias URLs, en lugar de utilizar los helpers de URL (y otros helpers que generan URLs, como algunos helpers de formulario) ya que están diseñados para trabajar con segmentos basados en URLs.

Controladores

Los controladores son el corazón de su aplicación, ya que determinan como se manejan las solicitudes HTTP.

¿Qué es un Controlador?

Un Controlador es simplemente un **archivo de clase** que se **nombre** de una **forma** en la que **se puede asociar** con una **URI**.

Considere esta URI:

```
example.com/index.php/blog/
```

En el ejemplo anterior, CodeIgniter intentaría encontrar un **controlador** llamado **blog.php** y cargarlo.

Cuando el nombre de un Controlador coincide con el primer segmento de una URI, se lo carga.

Probémoslo: Hola Mundo!

Vamos a crear un controlador simple para que pueda verlo en acción. Usando un editor de texto, cree un archivo llamado **blog.php** y escriba el siguiente código:

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

Luego guarde el archivo en su carpeta **application/controllers/**. Ahora visite su sitio usando una URL similar a esta:

```
example.com/index.php/blog/
```

Si hizo todo bien, debería ver **Hola Mundo!**.

Nota: Los nombres de clases tienen que comenzar 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 asegúrese que su controlador **extienda** a la clase del controlador padre para que pueda heredar todas sus funciones.

Funciones

En el ejemplo anterior, el nombre de la función es **index()**. La función "index" se carga siempre por defecto si el **segundo segmento** de la **URI** está vacío. Otra forma de mostrar su mensaje "Hola Mundo" sería este:

```
ejemplo.com/index.php/blog/index/
```

El segundo segmento de la URI determina qué función del controlador se llama.

Probémoslo. Agregue una nueva función a su controlador:

```
<?php
class Blog extends CI_Controller {

    public function index()
    {
        echo 'Hola Mundo!';
    }

    public function comentarios()
    {
        echo 'Mire esto!';
    }
}
?>
```

Ahora cargue la siguiente URL para ver la función **comentarios**:

```
ejemplo.com/index.php/blog/comentarios/
```

Debería ver su nuevo mensaje.

Pasar Segmentos URI a sus Funciones

Cadena de caracteres que identifica los recursos de una red de forma univoca y no varian en el tiempo , recursos son accesibles en una red o sistema

Si su **URI** contiene más de dos segmentos, ellos se pasarán a la función como parámetros.

Por ejemplo, digamos que tiene una URI como esta:

```
ejemplo.com/index.php/productos/zapatos/sandalias/123
```

Los **segmentos URI** 3 y 4 ("sandalias" y "123") se pasarán a su función:

```
<?php
class Productos extends CI_Controller {

    public function zapatos($sandalias, $id)
```

```

    {
        echo $sandalias;
        echo $id;
    }
?>
```

Importante: Si está usando Ruteo de URI, los segmentos pasados a su función serán redirigidos.

Definir un Controlador por Defecto

Se le puede decir a CodeIgniter que cargue un controlador por defecto cuando una URI no está presente, como serán los casos en los que se solicite solamente la URL raíz de su sitio. Para especificar el controlador por defecto, abra su archivo **application/config/routes.php** y establezca esta variable:

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

Donde **Blog** es el nombre de la clase controlador que quiere usar. Si ahora carga su archivo **index.php** principal sin especificar ningún segmento URI, verá por defecto el mensaje **Hola Mundo**.

Remapear las Llamadas de Función

Como se señaló anteriormente, el segundo parámetro de la URI normalmente determina qué función se llama en el controlador. CodeIgniter le permite anular este comportamiento mediante el uso de la función **_remap()**:

```
public function _remap()
{
    // Algun código aquí...
}
```

Importante: Si su controlador contiene una función llamada **_remap()**, se la llamará siempre independientemente de lo que la URI contenga. Se reemplaza el comportamiento normal en el que la URI determina qué función se llama, permitiéndole definir sus propias reglas de ruteo.

Para anular la llamada a la función (normalmente el segundo segmento de la URI) se pasará como parámetro a la función **_remap()**:

controlador/funcion/parametro

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

Cualquier segmento adicional después del nombre del método se pasa a **_remap()** como segundo parámetro opcional. Este array se puede usar en combinación con **call_user_func_array** de PHP para emular el comportamiento por defecto de CodeIgniter.

```
public function _remap($method, $params = array())
{
    $method = 'process_'. $method;
    if (method_exists($this, $method))
    {
        return call_user_func_array(array($this, $method), $params);
    }
    show_404();
}
```

Procesar la Salida

CodeIgniter tiene una clase para salidas que se encarga de enviar automáticamente sus datos finales al navegador web. Se puede encontrar más información sobre esto en las páginas Vistas y Clase **Output**. En algunos casos, sin embargo, puede desear pos-procesar los datos terminados en alguna forma y enviarlos Ud mismo al navegador. CodeIgniter le permite agregar una función llamada **_output()** a su controlador que recibirá los datos terminados de salida.

Importante: Si su controlador contiene una función llamada **_output()**, la clase **Output** siempre la llamará en lugar de imprimir los datos terminados directamente. El primer parámetro de la función contendrá la salida terminada.

Aquí hay un ejemplo:

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

Por favor, advierta que su función **_output()** recibirá los datos en su estado finalizado. Antes que se pasen a la función **_output()**, se presentarán los datos de evaluación de desempeño y uso de memoria, se escribirán los archivos de caché (si tiene el caché habilitado), y se enviarán los encabezados (si usa esta funcionalidad).

Para tener la salida de su controlador adecuadamente cacheada, su método **_output()** puede usar:

```
if ($this->output->cache_expiration > 0)
{
    $this->output->_write_cache($output);
}
```

Si está usando esta funcionalidad, las estadísticas del temporizador de ejecución de la página y uso de memoria pueden no ser perfectamente exactas, ya que no tendrán en cuenta cualquier proceso posterior que haga. Para conocer una forma alternativa de controlar la salida antes que se haga cualquier procesamiento final, por favor ver los métodos disponibles en la Clase **Output**.

Funciones Privadas

En algunos casos, puede desear que ciertas funciones estén ocultas del acceso público. Para hacer una función privada, simplemente agregue un guion de subrayado como prefijo del nombre y la función no será servida mediante una solicitud de URL. Por ejemplo, si fuera a tener una función como esta:

```
private function _hacer()
{
    // algo de código
}
```

No funcionará tratar de accederla mediante la URL:

```
ejemplo.com/index.php/blog/_hacer/
```

Organizar sus Controladores en Subcarpetas

Si está armando una **aplicación grande**, puede encontrar conveniente **organizar sus controladores** en **subcarpetas**. CodeIgniter le permite hacer esto.

Simplemente cree carpetas dentro de su directorio **application/controllers** y ubique sus clases controlador dentro de ellas.

Nota: Al usar esta funcionalidad, el **primer segmento** de la **URI** tiene que **especificar la carpeta**. Por ejemplo, digamos que tiene un controlador ubicado aquí:

```
application/controllers/productos/zapatos.php
```

Para llamar al controlador anterior, su URI lucirá como esto:

```
ejemplo.com/index.php/productos/zapatos/mostrar/123
```

Cada una de sus **subcarpetas** puede **contener** un **controlador** por **defecto** que **será llamado** si la **URL** **contiene solo la subcarpeta**. Simplemente **nombre** a su **controlador** por **defecto** **como se especifica** en su archivo **application/config/routes.php**.

CodeIgniter también le permite remapear sus URIs usando su funcionalidad Ruteo de URI.

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:

```
constructor - inicializa los objetos
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 Reservados de Función

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.

Nombres Reservados

CodeIgniter usa una serie de funciones y nombres en su operación. Debido a esto, el desarrollador no puede usar algunos nombres. La siguiente es la lista de nombres reservados que no se pueden usar.

Nombres de Controlador

Como sus clases de controlador extenderán al controlador principal de la aplicación, tiene que ser cuidadoso de no nombrar a sus funciones del mismo modo que las usadas por otras clases, sino sus clases las anularán. La siguiente es la lista de nombres reservados. No use ninguno de estos nombres para llamar a su controlador:

- Controller
- CI_Base
- _ci_initialize
- Default
- index

Funciones

- is_really_writable()
- load_class()
- get_config()
- config_item()
- show_error()
- show_404()
- log_message()
- _exception_handler()
- get_instance()

Variables

- \$config
- \$mimes
- \$lang

Constantes

- ENVIRONMENT
- EXT
- FCPATH
- SELF
- BASEPATH
- APPPATH
- CI_VERSION
- FILE_READ_MODE
- FILE_WRITE_MODE
- DIR_READ_MODE
- DIR_WRITE_MODE
- FOPEN_READ
- FOPEN_READ_WRITE
- FOPEN_WRITE_CREATE_DESTRUCTIVE
- FOPEN_READ_WRITE_CREATE_DESTRUCTIVE
- FOPEN_WRITE_CREATE
- FOPEN_READ_WRITE_CREATE
- FOPEN_WRITE_CREATE_STRICT
- FOPEN_READ_WRITE_CREATE_STRICT

Vistas

Una **vista** es simplemente una **página web** o un **fragmento de página**, tal como un **encabezado**, **pie de página**, una **barra lateral**, etc. De hecho, las vistas se pueden **embeber** flexiblemente **dentro de otras vistas** (dentro de otras vistas, etc., etc.) si necesita este tipo de jerarquía.

Nunca se puede llamar directamente a las **vistas**, las **tiene que cargar** un **controlador**. Recuerde que en un framework MVC, el Controlador actúa como el policía de tránsito, por lo que es responsable de traer una vista en particular. Si no leyó la página Controladores debería hacerlo antes de continuar.

Usando el ejemplo creado en la página Controladores, agregaremos una vista a él.

Crear una Vista

Usando su editor de texto, cree un archivo llamado **blog_view.php** y escriba esto en él:

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

Luego guarde el archivo en la carpeta **application/views/**.

Cargar 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 se necesitar especificar la extensión de archivo **.php**, a menos que use otra distinta a **.php**.

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

```
<?php
class Blog extends CI_Controller {

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

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

```
ejemplo.com/index.php/blog/
```

Cargar Varias Vistas

CodeIgniter manejará inteligentemente las llamadas a `$this->load->view` desde dentro de un controlador. Si ocurre más de una llamada, se agregarán juntas. Por ejemplo, puede que desee tener una vista de encabezado, una vista de menú, una vista de contenido, y una vista de pie de página. Eso podría ser algo como esto:

```
<?php

class Page extends CI_Controller {

    function index()
    {
        $data['page_title'] = 'Su título';
        $this->load->view('header');
        $this->load->view('menu');
        $this->load->view('content', $data);
        $this->load->view('footer');
    }

}
?>
```

En el ejemplo anterior, estamos usando "datos agregados dinámicamente", lo que verá más abajo.

Almacenar Vistas dentro de Subcarpetas

Sus archivos de vista también se pueden almacenar dentro de subcarpetas si prefiere ese tipo de organización. Al hacer eso, necesitará incluir el nombre de la carpeta que carga la vista. Ejemplo:

```
$this->load->view('nombre_de_carpeta/nombre_de_archivo');
```

Agregar Datos Dinámicos a la Vista

Los datos se pasan del controlador a la vista por medio de un **array** o un **objeto** en el **segundo parámetro** en la función de carga de la vista. Este es un ejemplo usando un array:

```
$data = array(
    'title' => 'Mi Titulo',
    'heading' => 'Mi Encabezado',
    'message' => 'Mi Mensaje'
);

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

Y este es un ejemplo usando un objeto:

```
$data = new Alguna_clase();
$this->load->view('blog_view', $data);
```

Nota: Si usa un objeto, las variables de clase se convertirán en elementos del array.

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

```
<?php
class Blog extends CI_Controller {

    function index()
    {
        $data['title'] = "Mi Titulo Real";
        $data['heading'] = "Mi Encabezado Real";

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

Ahora abra su archivo de vista y cambie el texto para variables que corresponden a las claves del array en sus datos:

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

Entonces cargue la página en el URL que usó y debería ver las variables reemplazadas.

Crear Bucles

El array de datos que pasa a su vista no se limita a variables simples. Puede pasar arrays multidimensionales, los que se pueden ciclar para generar varias filas. Por ejemplo, si extrae los datos de su base de datos, será en la forma de un array multidimensional.

Este es un ejemplo simple. Agregue esto a su controlador:

```
<?php
class Blog extends CI_Controller {

    function index()
    {
        $data['todo_list'] = array('Limpiar la casa', 'Llamar a mamá',
                                   'Hacer los mandados');

        $data['title'] = "Mi Titulo Real";
        $data['heading'] = "Mi Encabezado Real";

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

Ahora abra su archivo de vista y cree un bucle:

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

<h3>Mi Lista de Pendientes</h3>

<ul>
<?php foreach ($todo_list as $item) :?>
<li><?php echo $item;?></li>
<?php endforeach;?>
</ul>

</body>
</html>
```

Nota: Advertirá que en el ejemplo anterior estamos usando la sintaxis alternativa de PHP. Si no está familiarizado con ella, puede leer acerca suyo en "Sintaxis Alternativa de PHP para Archivos de Vistas".

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**, que la enviará al navegador. Recuerde asignarla a una variable si quiere que devuelva datos:

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

Modelos

Los modelos están disponibles **opcionalmente** para aquellos que quieren usar un enfoque MVC más tradicional.

¿Qué es un Modelo?

Los **modelos** son **clases de PHP** que se **diseñan para trabajar con información** en su **base de datos**. Por ejemplo, digamos que usa CodeIgniter para administrar un blog. Podría tener una clase de modelo que contenga **funciones para insertar, actualizar y devolver los datos de su blog**. Aquí hay un ejemplo de cómo luciría tal clase de modelo:

```
class Blog_model extends CI_Model {
    var $title = '';
    var $content = '';
    var $date = '';

    function __construct()
    {
        // Llamar al constructor de CI_Model
        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 las funciones de base de datos del **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')**

Anatomía de un Modelo

Las clases de modelos se almacenan en su carpeta **application/models/**. Pueden estar anidadas dentro de subcarpetas si quiere este tipo de organización.

El prototipo básico para la clase de un modelo es este:

```
class Nombre_modelo extends CI_Model {

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

Donde **Nombre_modelo** es el nombre de su clase. Los nombres de clases **TIENEN QUE TENER** la primera letra en **mayúscula** con el resto del nombre en minúsculas. Asegúrese que su clase extiende a la clase base **CI_Model**.

El nombre de archivo será la versión en minúsculas del 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
```

Cargar un Modelo

Sus modelos normalmente se cargarán y llamarán desde dentro de sus funciones controlador. Para cargar un modelo, usará la siguiente función:

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

Si su modelo está ubicado en una subcarpeta, incluir la ruta relativa de su carpeta de modelos. Por ejemplo, si tiene un modelo ubicado en **application/models/blog/consultas.php** lo llamará usando:

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

Una vez cargado, accederá a las funciones de su modelo usando un objeto con el mismo nombre que su clase:

```
$this->load->model('Nombre_modelo');
$this->Nombre_modelo->function();
```

Si quisiera que su modelo se asigne a un nombre de objeto diferente, puede especificarlo mediante el segundo parámetro de la función de carga:

```
$this->load->model('Nombre_modelo', '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'); La clase Blog se convierte en objeto e invoca la función
        $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**.

Conectar a su Base de Datos

Cuando se carga un modelo, **NO** conecta automáticamente a la base de datos. Están disponibles las siguientes opciones de conexión:

- Puede conectar usando los métodos estándar de base de datos descriptos aquí, tanto desde su clase controlador como desde cualquier clase modelo.
- Puede decirle a la **función de carga del modelo** que **conecte automáticamente** pasándole **TRUE** (booleano) mediante el **tercer parámetro**, y **valores de conectividad**, como se define en su **archivo de configuración de base de datos**:

```
$this->load->model('Nombre_modelo', '', TRUE);
```

- Puede pasar manualmente los valores de conectividad de base de datos mediante el tercer parámetro:

```
$config['hostname'] = "localhost";
$config['username'] = "mi_usuario";
$config['password'] = "mi_contraseña";
$config['database'] = "mi_base_de_datos";
$config['dbdriver'] = "mysql";
$config['dbprefix'] = "";
$config['pconnect'] = FALSE;
$config['db_debug'] = TRUE;

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

Funciones Helper

Los helpers, como su nombre sugiere, lo ayudan con las tareas. Cada archivo de helper es simplemente una colección de funciones en una categoría particular. Hay Helpers de URL que ayudan en la creación de enlaces, Helpers de Formulario que lo ayudan a crear elementos de formulario, Helpers de Texto que ejecutan varias rutinas de formateo de texto, Helpers de Cookie que escriben y leen cookies, Helpers de Archivo que ayudan a trabajar con archivos, etc.

A diferencia de la mayoría de otros sistemas de CodeIgniter, los Helpers no se escriben en formato Orientado a Objetos. Son simples funciones procedimentales. Cada función helper ejecuta una tarea específica, sin dependencia con otras funciones.

CodeIgniter no carga por defecto los Archivos Helper, por lo tanto el primer paso para usar un Helper es cargarlo. Una vez cargado, se hace disponible globalmente en su controlador y vista.

Los helpers se almacenan normalmente en su directorio system/helpers o application/helpers. CodeIgniter buscará primero en su directorio application/helpers. Si el directorio no existe o el helper indicado no está ubicado allí, en su lugar, CodeIgniter buscará en la carpeta global system/helpers.

Cargar un Helper

Cargar un archivo Helper es bastante sencillo usando la siguiente función:

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

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

Por ejemplo, para cargar el archivo Helper de URL que se llama url_helper.php, debería hacer esto:

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

Se puede cargar un helper en cualquier lugar dentro de las funciones controlador (o aún dentro de los archivos de vistas, aunque esto no es una buena práctica), siempre y cuando se lo cargue antes de usarlo. Puede cargar sus helpers en el constructor de un controlador, por lo que estarán disponibles automáticamente en cualquier función, o puede cargar un helper en una función específica que lo necesite.

Nota: La función de carga de Helpers anterior, no devuelve un valor. Por lo tanto no intente asignarla a una variable. Simplemente úsela como se muestra.

Cargar Varios Helpers

Si necesita cargar más de un helper, puede especificarlos en un array, de este modo:

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

Carga Automática de Helpers

Si tiene la necesidad de disponer globalmente de un helper a lo largo de su aplicación, puede decirle a CodeIgniter que lo cargue automáticamente durante la inicialización del sistema. Esto se hace agregando el helper al array de carga automática del archivo application/config/autoload.php.

Usar un Helper

Una vez que se **cargó** el archivo **Helper** contenido la función que tiene la **intención usar**, la **llamará** de la forma que lo haría **con una función estándar de PHP**.

Por ejemplo, para **crear** un **enlace** usando la **función anchor()** en uno de los archivos de vista, debería hacer esto:

```
<?php echo anchor('blog/comentarios', 'Clic aquí');?>
```

Donde "Clic aquí" es el nombre del enlace, y "blog/comentarios" es la **URI** al **controlador/funció**n que desea enlazar.

"Extender" Helpers

Para **"extender"** **Helpers**, cree un **archivo** en su **carpeta application/helpers/** con el **mismo nombre** que el **Helper existente, pero prefijado** con **MY_** (este ítem es configurable. Ver más abajo).

Si todo lo que **necesita** es **agregar** alguna **funcionalidad** a un **helper existente** - quizás **agregar** **una o dos funciones**, o **cambiar** **como** **una función opera** - entonces es exagerado reemplazar el helper entero con su versión. En este caso es mejor simplemente **"extender"** el Helper. El término **"extender"** se usa ligeramente, ya que las **Funciones Helpers** son **procedimentales** y **discretas** y **no se pueden extender** en el **sentido tradicional** de la programación. Internamente, le da la posibilidad de agregar funciones a las que provee el Helper o modificar cómo opera en forma nativa la Función Helper.

Por ejemplo, para extender el Helper Array nativo creará un archivo llamado **application/helpers/MY_array_helper.php**, y agregará o anulará funciones:

```
// any_in_array() no está en el Helper Array, por lo que define una nueva función
function any_in_array($needle, $haystack)
{
    $needle = (is_array($needle)) ? $needle : array($needle);

    foreach ($needle as $item)
    {
        if (in_array($item, $haystack))
        {
            return TRUE;
        }
    }

    return FALSE;
}

// random_element() está incluido en el Helper Array,
// por lo que anula la función nativa
function random_element($array)
{
    shuffle($array);
    return array_pop($array);
}
```

Establecer su Propio Prefijo

Los prefijos de archivo para "extender" los **Helpers** son los mismos usados para extender bibliotecas y clases del núcleo. Para establecer su propio prefijo, abra su archivo **application/config/config.php** y busque este ítem:

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

Por favor advierta que todas las bibliotecas nativas de CodeIgniter están prefijadas con **CI_**, por lo tanto NO use ese prefijo.

¿Y Ahora Qué?

En la tabla de contenido encontrará la lista de todos los archivos de helper disponibles. Inspeccione cada uno para ver que hacen.

Usar las Bibliotecas de CodeIgniter

Todas las bibliotecas disponibles se ubican en la carpeta **system/libraries**. En la mayoría de los casos, el **uso** de una 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 **desea** **invocar**. Por ejemplo, para **cargar** la **clase** de **validación** de **formularios** debería hacer esto:

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

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

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

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

Crear sus Propias Bibliotecas

Por favor lea la sección de la guía del usuario que discute cómo crear sus propias bibliotecas.

Crear sus Propias Bibliotecas

Cuando usamos el término "Bibliotecas" nos referimos normalmente a las **clases** que están localizadas en el directorio **/libraries** y se describen en la Referencia de Clases de esta guía del usuario. En este caso, sin embargo, describiremos en su lugar como puede crear sus propias bibliotecas dentro del directorio **application/libraries** para mantener la separación entre sus recursos locales y los recursos globales del framework.

Además, CodeIgniter permite que sus bibliotecas extiendan a las clases nativas si simplemente necesita agregar alguna funcionalidad a una biblioteca existente. O incluso puede reemplazar bibliotecas nativas con solamente colocar versiones con el mismo nombre en su carpeta **application/libraries**.

En resumen:

- Puede crear bibliotecas completamente nuevas.
- Puede extender bibliotecas nativas.
- Puede reemplazar bibliotecas nativas.

La siguiente página explica estos tres conceptos en detalle.

Nota: Las **clases** de **base de datos** no se **pueden** **extender** o **reemplazar** con **sus propias clases**. Todas las otras **clases** se pueden **reemplazar/extender**.

Almacenamiento

Sus clases de biblioteca se deberían ubicar dentro de la carpeta **application/libraries**, ya que este es el lugar donde CodeIgniter las buscará para inicializarlas.

Convenciones de Nombres

- Los **nombres** de **archivos** tienen que **comenzar** con **mayúscula**. Por ejemplo: **Mi_clase.php**
- Las **declaraciones de clase** tiene que **comenzar** con **mayúscula**. Por ejemplo: **class Mi_clase**
- Los **nombres de clase** y los **nombres de archivo** tiene que **coincidir**.

El Archivo de Clase

Las clases deberían tener este prototipo básico (**Nota:** Estamos usando el nombre **Alguna_clase** puramente como ejemplo):

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

class Alguna_clase {

    public function alguna_funcion()
    {
    }

}

/* Fin del archivo Alguna_clase.php */
```

Usar su Clase

Puede **inicializar** su **clase** desde **dentro** de cualquier función **controlador**, usando el estándar:

```
objecto      function
$this->load->library('alguna_clase');
```

Donde **alguna_clase** es el **nombre** del **archivo**, **sin** la extensión **".php"**. Puede enviar el nombre del archivo iniciando en mayúsculas o todo en minúsculas. A CodeIgniter no le importa.

Una vez cargada puede **acceder** a su **clase** usando la versión **en minúsculas**:

```
$this->alguna_clase->alguna_funcion(); // las instancias del objeto siempre
                                         // estarán en minúsculas
```

Pasar Parámetros al Inicializar su Clase

En la **función** de **carga** de **bibliotecas**, puede pasar dinámicamente datos como un **array** mediante el **segundo parámetro**, el cual **se pasará** al **constructor** de su **clase**:

```
$params = array('tipo' => 'grande', 'color' => 'rojo');
               función para cargar bibliotecas
$this->load->library('Alguna_clase', $params);
```

Si **utiliza** esta **funcionalidad**, tiene que **configurar** al **constructor** de su **clase** para que **acepte** **datos**:

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

class Alguna_clase {

    public function __construct($params)
    {
        // Hacer algo con $params
    }
}

?>
```

También puede pasar parámetros almacenados en un archivo de configuración. Simplemente cree un archivo de configuración llamado igual que **el nombre del archivo** de la clase y guárdelo en su carpeta **application/config/**. Advertia que si pasa dinámicamente un parámetro como se describe arriba, la opción del archivo de configuración no estará disponible.

Utilizar Recursos de CodeIgniter dentro de sus Bibliotecas

Para **acceder** a los **recursos** **nativos** de **CodeIgniter** **dentro** de sus **bibliotecas**, use la **función** **get_instance()**. Esta **función** **devuelve** el **super objeto** de **CodeIgniter**.

Normalmente, desde dentro de las funciones de controlador llamará a cualquier función de CodeIgniter disponible usando la construcción **\$this**:

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

Sin embargo, **\$this** solamente funciona directamente dentro de sus controladores, sus modelos o sus vistas. Si quisiera usar las clases de CodeIgniter desde dentro de sus propias clases, puede hacer lo siguiente:

Primero, asignar el objeto CodeIgniter a una variable:

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

Una vez que asignó el objeto a una variable, usará esa variable *en lugar* de **\$this**:

```
$CI =& get_instance();

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

Nota: Advertirá que la función anterior **get_instance()** se pasa por referencia:

\$CI =& get_instance();

Esto es muy importante. La asignación por referencia le permite usar el objeto CodeIgniter original en lugar de hacer una copia de él.

Reemplazar Bibliotecas Nativas con sus Versiones

Simplemente al nombrar su clase igual que a una biblioteca nativa, provocará que CodeIgniter la use en lugar de la nativa. Para usar esta funcionalidad, tiene que nombrar al archivo y la declaración de clase exactamente igual que la biblioteca nativa. Por ejemplo, para reemplazar la biblioteca nativa **Email**, creará un archivo llamado **application/libraries/Email.php** y declarará su clase con:

```
class CI_Email {  
}
```

Advierta que la mayoría de las clases nativas están prefijadas con **CI_**.

Para cargar su biblioteca verá la función de carga estándar:

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

Nota: Por el momento, las clases de base de datos no se pueden reemplazar con sus propias versiones.

Extender Bibliotecas Nativas

Si todo lo que necesita hacer es agregar alguna funcionalidad a una biblioteca existente - quizás agregar una o dos funciones - entonces es exagerado reemplazar toda la biblioteca con su versión. En este caso es mejor simplemente extender la clase. Extender una clase es casi lo mismo que reemplazar una clase, con un par de excepciones:

- La declaración de clase tiene que extender la clase padre.
- El nuevo nombre de clase y nombre de archivo se tiene que prefijar con **MY_** (este ítem es configurable. Ver más abajo).

Por ejemplo, para extender la clase nativa **Email**, creará un archivo llamado **application/libraries/MY_Email.php**, y declarará su clase con:

```
class MY_Email extends CI_Email {  
}
```

Nota: Si necesita usar un constructor en su clase, asegúrese que extiende el constructor padre:

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

Cargar sus Subclases

Para cargar sus subclases, usará la sintaxis estándar usada normalmente. **NO** incluya su prefijo. Por ejemplo, para cargar el ejemplo anterior que extiende la clase **Email**, usará:

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

Una vez cargada, usará la variable de clase como lo haría normalmente para la clase que está extendiendo. En el caso de la clase **Email**, todas las llamadas usarán:

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

Establecer su Propio Prefijo

Para establecer su propio prefijo de subclase, abra su archivo **application/config/config.php** y busque este ítem:

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

Por favor advierta que todas las bibliotecas nativas de CodeIgniter están prefijadas con **CI_**, por lo tanto, **NO** use ese 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->algun_padre->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 sus Propios Drivers

Por favor lea la sección de la guía del usuario que discute cómo crear sus propios drivers.

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  
            Nombre_de_driver_subclase_1.php  
            Nombre_de_driver_subclase_2.php  
            Nombre_de_driver_subclase_3.php
```

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

Crear Clases del Núcleo

Cada vez que CodeIgniter se ejecuta hay varias clases base que se inicializan automáticamente como parte del núcleo del framework. Sin embargo, es posible cambiar cualquiera de las clases del núcleo del sistema con sus propias versiones o aún extender las versiones del núcleo.

La mayoría de los usuarios nunca tendrán la necesidad de hacer esto, pero existe la posibilidad de reemplazarlas o extenderlas, para aquellos que quisieran alterar significativamente el núcleo de CodeIgniter.

Nota: Jugar con una clase del núcleo del sistema tiene muchas implicaciones, así que asegúrese que sabe lo que va a hacer antes de intentarlo.

Lista de Clases del Sistema

La siguiente es la lista de los archivos del núcleo del sistema que se invocan cada vez que se ejecuta CodeIgniter:

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

Reemplazar Clases del Núcleo

Para usar una de sus propias clases del sistema en lugar de una por defecto, simplemente ubique su versión dentro de su directorio **application/core** local:

```
application/core/alguna-clase.php
```

Si este directorio no existe, puede crearlo.

Cualquier archivo nombrado del mismo modo que uno de la lista anterior, se usará en lugar del original.

Por favor advierta que su clase tiene que usar el prefijo **CI**. Por ejemplo, si su archivo se llama **Input.php** la clase se llamará:

```
class CI_Input {  
}
```

Extender las Clases del Núcleo

Si todo lo que necesita hacer es agregar alguna funcionalidad a una biblioteca existente - quizás agregar una o dos funciones - entonces es exagerado reemplazar la biblioteca entera con su propia versión. En este caso es mejor simplemente extender la clase. Extender una clase es casi lo mismo que reemplazar una clase, solo que con un par de excepciones:

- La declaración de la clase tiene que extender a la clase padre.
- Su nuevo nombre de clase y nombre de archivo tienen que estar prefijados con **MY_** (este ítem es configurable. Ver más abajo).

Por ejemplo, para extender la clase nativa **Input**, creará un archivo llamado **application/core/MY_Input.php**, y declarará su clase con:

```
class MY_Input extends CI_Input {  
}
```

Nota: Si necesita usar un constructor en su clase, asegúrese que extiende el constructor padre:

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

Tip: Cualquier función en su clase que se llame del mismo modo que alguna función en la clase padre se usará en lugar de la nativa (esto se conoce como "anulación 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 constructores de sus controladores de la aplicación.

```
class Welcome extends MY_Controller {  
  
    function __construct()  
    {  
        parent::__construct();  
    }  
  
    function index()  
    {  
        $this->load->view('welcome_message');  
    }  
}
```

Establecer su Propio Prefijo

Para establecer su propio prefijo de subclase, abra su archivo **application/config/config.php** y busque este ítem:

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

Por favor advierta que todas las bibliotecas nativas de CodeIgniter están prefijadas con **CI_**, por lo tanto, **NO** use ese prefijo.

Hooks - Extender el Núcleo del Framework

La funcionalidad de Hooks de CodeIgniter provee un medio para aprovechar y modificar el funcionamiento interno del framework sin alterar los archivos del núcleo. Cuando CodeIgniter corre, sigue un proceso de ejecución específico, diagramado en la página Flujo de la Aplicación. Puede haber casos, sin embargo, en que le gustaría hacer algún tipo de acción que tenga lugar en una etapa particular del proceso de ejecución. Por ejemplo, es posible que desee ejecutar un script antes que se carguen sus controladores, o exactamente después, o es posible que desee activar uno de sus propios scripts en algún otro lugar.

Habilitar Hooks

La funcionalidad de hooks se puede habilitar/deshabilitar globalmente al establecer el siguiente ítem en el archivo **application/config/config.php**:

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

Definir un Hook

Los hooks están definidos en el archivo **application/config/hooks.php**. Cada hook está especificado como un array con este prototipo:

```
$hook['pre_controller'] = array(
    'class'      => 'Mi_clase',
    'function'   => 'Mi_funcion',
    'filename'   => 'Mi_clase.php',
    'filepath'   => 'hooks',
    'params'     => array('cerveza', 'vino', 'soda')
);
```

Notas:

El índice del array se corresponde con el nombre del punto de enganche en particular que desea usar. En el ejemplo anterior, el punto de enganche es **pre_controller**. Abajo hay una lista de puntos de enganche. Los siguientes ítems se deberían definir en su array asociativo de hooks:

- **class:** Nombre de la clase que desea invocar. Si prefiere usar una función procedimental en lugar de una clase, deje este ítem en blanco.
- **function:** Nombre de la función que desea llamar.
- **filename:** Nombre del archivo que contiene su clase o función.
- **filepath:** Nombre del directorio que contiene su script. **Nota:** Su script tiene que estar localizado en un directorio DENTRO de su carpeta **application**, por lo que su ruta de archivo es relativa a esa carpeta. Por ejemplo, si su script está localizado en **application/hooks**, simplemente usará **hooks** como su ruta de archivo. Si su script está localizado en **application/hooks/utilities** usará **hooks/utilities** como su ruta de archivo. Sin barra al final.
- **params:** Cualquier parámetro que deseé pasarle al script. Este ítem es opcional.

Varias Llamadas al Mismo Hook

Si quiere usar el mismo punto de enganche con más de un script, simplemente haga multidimensional su declaración de array, así:

```
$hook['pre_controller'][] = array(
    'class'      => 'Mi_clase',
    'function'   => 'Mi_funcion',
    'filename'   => 'Mi_clase.php',
```

```
        'filepath' => 'hooks',
        'params'    => array('cerveza', 'vino', 'soda')
    );

$hook['pre_controller'][] = array(
    'class'      => 'Mi_otra_clase',
    'function'   => 'Mi_otra_funcion',
    'filename'   => 'Mi_otra_clase.php',
    'filepath'   => 'hooks',
    'params'     => array('rojo', 'amarillo', 'azul')
);
```

Advierta los corchetes después de cada índice de array:

```
$hook['pre_controller'][]
```

Esto le permite tener el mismo punto de enganche con varios scripts. El orden que define su array será el orden de ejecución.

Puntos de Enganche

La siguiente es la lista de puntos de enganche disponibles.

- **pre_system**
Llamado muy pronto durante la ejecución del sistema. En este punto, solamente se cargaron las clases de hooks y benchmark. No ocurrió ningún ruteo u otro proceso.
- **pre_controller**
Llamado inmediatamente antes de que cualquiera de sus controladores haya sido llamado. Se hicieron todas las clases base, ruteo y verificaciones de seguridad.
- **post_controller_constructor**
Llamado inmediatamente después que su controlador se instanció, pero antes que haya ocurrido cualquier llamada a un método.
- **post_controller**
Llamado inmediatamente después que su controlador se ejecutó completamente.
- **display_override**
Anula la función **_display()**, usada para enviar la página finalizada al navegador web al final de la ejecución del sistema. Esto le permite usar su propia metodología de impresión. Advierta que necesitará referenciar al superobjeto **CI** con **\$this->CI =& get_instance()** y luego los datos finalizados estarán disponibles llamando a **\$this->CI->output->get_output()**
- **cache_override**
Le permite llamar a su propia función en lugar de la función **_display_cache()** en la clase **Output**. Esto le permite usar su propio mecanismo de visualización del caché.
- **post_system**
Llamado después que la página final renderizada se envíe al navegador, en el final de la ejecución del sistema y después que los datos finalizados se envían al navegador.

Carga Automática de Recursos

CodeIgniter viene con una funcionalidad de "carga automática" que permite cargar bibliotecas, helpers, y modelos para inicializarlos automáticamente cada vez que se ejecuta el sistema. Si necesita globalmente ciertos recursos a través de su aplicación, por comodidad debería considerar cargarlos automáticamente.

Los siguientes elementos se cargan automáticamente:

- Clases del Núcleo encontradas en la carpeta "libraries"
- Archivos Helper encontrados en la carpeta "helpers"
- Archivos personalizados de configuración encontrados en la carpeta "config"
- Archivos de idioma encontrados en la carpeta "system/language"
- Modelos encontrados en la carpeta "models"

Para cargar automáticamente los recursos, abra el archivo **application/config/autoload.php** y agregue el ítem que quiere que se cargue al array **\$autoload**. Encontrará instrucciones en ese archivo correspondientes a cada tipo de ítem.

Nota: No incluya la extensión del archivo (.php) al agregar elementos al array **\$autoload**.

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 Clase **Config**. Sin embargo, **config_item()** se puede usar para recuperar claves simples. Para más información, lea la documentación de la Clase **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 **acceso directo** a la **función htmlspecialchars()**. **Acepta cadenas y arrays**. Es muy útil para **evitar** Cross Site Scripting (XSS).

Ruteo URI

Normalmente hay una relación de uno a uno entre una cadena de URL y su correspondiente clase/método controlador. Los segmentos en una URI usualmente siguen este patrón:

```
ejemplo.com/clase/funcion/id/
```

Sin embargo en algunos casos, se puede remapear esta relación para que, en su lugar, se llame a una clase/función diferente de la correspondiente a la URL.

Por ejemplo, digamos que quiere que sus URLs tengan este prototipo:

```
ejemplo.com/producto/1/  
ejemplo.com/producto/2/  
ejemplo.com/producto/3/  
ejemplo.com/producto/4/
```

Usualmente el segundo segmento de la URL se reserva para el nombre de la función, pero en el ejemplo anterior está el ID del producto. Para superar esto, CodeIgniter te permite remapear el gestor de URI.

Establecer sus Propias Reglas de Ruteo

Las reglas de ruteo están definidas en su archivo **application/config/routes.php**. En él verá un array llamado **\$route** que le permite especificar sus propios criterios de ruteo. Las rutas se pueden especificar sea usando **Comodines** como **Expresiones Regulares**.

Comodines

Un comodín típico puede lucir así:

```
$route['producto/:num'] = "catalogo/producto_lookup";
```

En una ruta, la clave del array contiene la URI que debe coincidir, mientras que el valor del array contiene el destino al que se debe redirigir. En el ejemplo anterior, si se encuentra la palabra literal "producto" en el primer segmento de la URL, y un número en el segundo segmento, la clase "catalogo" y el método "producto_lookup" se usan en su lugar.

Puede buscar coincidir los valores literales o puede usar dos tipos de comodines:

(:num) buscará la coincidencia en un segmento que contiene números solamente.
(:any) buscará la coincidencia en un segmento que contiene cualquier carácter.

Nota: Las rutas se ejecutarán en el orden que están definidas. Las rutas más altas siempre tendrán precedencia sobre las más bajas.

Ejemplos

Aquí hay algunos ejemplos de ruteo:

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

Una URL que contiene la palabra "diarios" en el primer segmento, se remapeará a la clase "blogs".

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

Una URL que contiene los segmentos blog/jose se remapeará a la clase "blogs" y al método "usuarios". El ID se establecerá como "34".

```
$route['producto/(:any)'] = "catalogo/producto_buscar";
```

Una URL con "producto" como primer segmento y cualquier cosa en el segundo, se remapeará a la clase "catalogo" y al método "producto_buscar".

```
$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 o al final.

Expresiones Regulares

Si lo prefiere, puede usar expresiones regulares para definir sus reglas de ruteo. Se permite cualquier expresión regular válida, como son las back-references.

Nota: Si usa back-references tiene que usar la sintaxis de dólar en lugar de la sintaxis de barras invertidas dobles.

Una regla RegEx típica luciría así:

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

En el ejemplo anterior, una URI similar a **productos/camisetas/123** llamaría en su lugar a la clase controlador camisetas y a la función id_123.

También puede mezclar comodines con expresiones regulares.

Rutas Reservadas

Hay dos rutas reservadas:

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

Esta ruta indica que la clase controlador se debería cargar si la URI no contiene datos, lo que será el caso de cuando la gente carga la URL raíz. En el ejemplo anterior, se cargaría la clase "bienvenida". Se le recomienda que siempre tenga una ruta por defecto, ya que de lo contrario, aparecerá una página 404 por defecto.

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

Esta ruta indica la clase de controlador se debería cargar si no se encuentra al controlador solicitado. Esto anulará a la página por defecto del error 404. Esto no afectará a la función **show_404()**, que continuará cargando el

archivo **error_404.php** por defecto en **application/errors/error_404.php**.

Importante: Las rutas reservadas tienen que estar antes que cualquier ruta con comodines o expresiones regulares.

Manejo de Errores

CodeIgniter le permite armar un reporte de errores en sus aplicaciones usando las [funciones](#) descriptas debajo. Además, tiene una clase de registro de errores que permite que los mensajes de error y depuración se guarden como archivos de texto.

Nota: Por defecto, CodeIgniter muestra todos los errores de PHP. Podría desear cambiar este comportamiento una vez que el desarrollo se complete. Encontrará la función [error_reporting\(\)](#) ubicada en la parte superior del archivo [index.php](#) principal. Deshabilitar el reporte de errores NO evitara que los archivos de registro se escriban si hay errores.

A diferencia de la mayoría de los sistemas en CodeIgniter, las funciones de error son interfaces simples de procedimientos que están disponibles globalmente a lo largo de la aplicación. Este enfoque le permite a los mensajes de error dispararse sin tener que preocuparse acerca del ámbito de clases/funciones.

Las siguientes funciones le permiten generar errores:

show_error('mensaje' [, int \$codigo_estado= 500])

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

application/errors/error_general.php

El parámetro opcional **\$codigo_estado** determina que código de estado de HTTP se debería enviar con el error.

show_404('página' [, 'log_error'])

Esta función mostrará el mensaje del error 404 proporcionado a ella usando la siguiente plantilla de error:

application/errors/error_404.php

La función espera que la cadena pasada a ella sea la ruta de archivo a la página que no se encuentra. Advierta que CodeIgniter automáticamente muestra mensajes 404 si no se encuentran los controladores.

CodeIgniter registra automáticamente cualquier llamada a [show_404\(\)](#). El registro se detiene cuando se establece el segundo parámetro opcional a **FALSE**.

log_message('level' , 'message')

Esta función le permite escribir mensajes en sus archivos de registro. Tiene que suministrar uno de los tres "niveles" en el primer parámetro, indicando que tipo de mensaje es (debug, error, info), y el mensaje en el segundo parámetro. Ejemplo:

```
if ($alguna_variable == "")  
{  
    log_message('error', 'alguna_variable no contenía valor.');//  
}  
else  
{  
    log_message('debug', 'alguna_variable está bien establecida');//  
}  
  
log_message('info', 'El propósito de alguna_variable es proveer algún valor.');//
```

Hay tres tipos de mensajes:

1. **Mensajes de Error.** Estos son errores reales, tales como errores de PHP o errores del usuario.
2. **Mensajes de Depuración.** Estos son mensajes que lo asisten en la depuración. Por ejemplo, si una clase fue inicializada, podría registrar esto como información de depuración.
3. **Mensajes Informativos.** Estos son los mensajes de menor prioridad, que simplemente dan información acerca de algún proceso. CodeIgniter no genera nativamente ningún mensaje de información, pero puede querer hacerlo en su aplicación.

Nota: Para que realmente se escriba el archivo de registro, la carpeta "logs" tiene que ser escribible. Además, tiene que establecerse "log_threshold" en **application/config/config.php**. Por ejemplo, podría desear que se registren solamente los mensajes de error, y no los otros dos tipos. Si lo establece a cero, el registro estará deshabilitado.

Almacenamiento en Caché de Páginas Web

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

Aunque CodeIgniter es bastante rápido, la cantidad de información dinámica que muestra en sus páginas se correlacionarán directamente con los recursos del servidor, memoria y ciclos de procesamiento utilizados, que afectan la velocidad de carga de sus páginas. Al cachear sus páginas, como se guardan en su estado completamente renderizadas, se puede alcanzar un rendimiento que se aproxima al de las páginas web estáticas.

¿Cómo Funciona el Almacenamiento en Caché?

El almacenamiento en caché se puede habilitar por página, y se puede establecer la cantidad de tiempo que una página debería permanecer cacheada antes que sea refrescada. Cuando se carga una página por primera vez, se escribirá el archivo de caché a su carpeta **application/cache**. En las siguientes cargas de la página, se recuperará el archivo de caché y se lo enviará al navegador del usuario que la solicite. Si caducó, se borrará y refrescará antes de ser enviada al navegador.

Nota: La etiqueta Benchmark no se cachea, por lo que todavía puede ver la velocidad de carga de su página cuando el almacenamiento en caché está habilitado.

Habilitar el Almacenamiento en Caché

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

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

Donde **n** es la cantidad de minutos que desea que la página permanezca cacheada entre refrescos.

La anterior etiqueta puede ir en cualquier parte dentro de una función. No se ve afectada por el orden en que aparece, por lo que ubíquela donde sea más lógico para Ud. Una vez que la etiqueta está en su lugar, las páginas comienzan a cachearse.

Atención: Debido a la forma en que CodeIgniter almacena el contenido para la impresión, el almacenamiento en caché funcionará solamente si se está generando salida para su controlador con una vista.

Nota: Antes que se puedan escribir los archivos de caché, se tienen que establecer los permisos de archivo en su carpeta **application/cache** para que se pueda escribir.

Borrar Cachés

Si no desea más cachear un archivo, puede quitar la etiqueta de almacenamiento en caché y no se refrescará más cuando caduque. **Nota:** El caché no se borrará inmediatamente porque se quite la etiqueta. Tendrá que caducar normalmente. Si necesita quitarlo antes, deberá borrarlo manualmente de la carpeta de caché.

Perilar su Aplicación

La Clase Profiler mostrará los resultados de la evaluación de desempeño, consultas que ejecute, y los datos \$_POST al final de sus páginas. Esta información puede ser útil durante el desarrollo para ayudar con la depuración y la optimización.

Inicializar la Clase

Importante: Esta clase **NO** necesita inicializarse. La Clase Output la carga automáticamente si el perfilado está habilitado como se muestra más abajo.

Activar el Perfilador

Para habilitar el perfilador ubicar la siguiente función en cualquier parte dentro de sus funciones del controlador:

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

Al habilitarse, se genera un reporte y se lo inserta al final de sus páginas.

Para deshabilitar el perfilador, usará:

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

Establecer Puntos de Evaluación de Desempeño

Para que el Perfilador compile y muestre sus datos de evaluación de desempeño, tiene que marcar puntos usando una sintaxis específica.

Por favor lea la información sobre cómo establecer puntos de evaluación de desempeño en la página de la Clase Benchmark.

Habilitando y Deshabilitando Secciones del Perfilador

Cada sección de los datos del Perfilador se puede habilitar o deshabilitar estableciendo la correspondiente variable de configuración a **TRUE** o **FALSE**. Esto se puede hacer de una de dos formas. Primero, puede establecer los valores más amplios por defecto de la aplicación con el archivo de configuración application/config/profiler.php.

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

En sus controladores, puede anular los valores por defecto y valores del archivo de configuración llamando al método **set_profiler_sections()** de la Clase Output:

```
$sections = array(
    'config'  => TRUE,
    'queries' => TRUE
);

$this->output->set_profiler_sections($sections);
```

En la siguiente tabla se describen las secciones disponibles y el array usado para accederlas.

Clave	Descripción	Por Defecto
benchmarks	Tiempo transcurrido de los puntos de Evaluación de Desempeño y tiempo total de ejecución	TRUE
config	Variables de Configuración de CodeIgniter	TRUE
controller_info	Clase Controller y métodos requeridos	TRUE
get	Cualquier dato GET pasado en la solicitud	TRUE
http_headers	Encabezados HTTP para la solicitud actual	TRUE
memory_usage	Cantidad de memoria consumida por la solicitud actual, en bytes	TRUE
post	Cualquier dato POST pasado en la solicitud	TRUE
queries	Listado de todas las consultas de base de datos ejecutadas, incluyendo el tiempo de ejecución	TRUE
uri_string	La URI de la solicitud actual	TRUE

Administrar Aplicaciones

Por defecto se asume que tiene la intención de usar CodeIgniter para administrar una sola aplicación, la que armará en su directorio **application**. Sin embargo es posible tener varios conjuntos de aplicaciones que comparten una sola instalación de CodeIgniter, o incluso renombrar o relocatear su carpeta **application**.

Renombrar la Carpeta de la Aplicación

Si quisiera **renombrar** su **carpeta application**, puede hacerlo siempre y cuando abra su **archivo index.php principal** y establezca su nombre usando la variable **\$application_folder**:

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

Reubicar la Carpeta de la Aplicación

Es posible mover su carpeta **application** a una ubicación diferente en su servidor que la carpeta **system**. Para hacer esto, abra su archivo **index.php** principal y establezca una ruta completa de servidor en la variable **\$application_folder**.

```
$application_folder = "/ruta/a/su/aplicacion";
```

Ejecutar Varias Aplicaciones con una Instalación de CodeIgniter

Si quisiera compartir una instalación común de CodeIgniter para **administrar varias aplicaciones diferentes**, simplemente ponga todos los **directorios** localizados dentro de su **carpeta application** dentro de su propia **subcarpeta**.

Por ejemplo, digamos que quiere **crear dos aplicaciones**, "foo" y "bar". Podría estructurar sus carpetas de aplicación de este modo:

```
application/foo/
application/foo/config/
application/foo/controllers/
application/foo/errors/
application/foo/libraries/
application/foo/models/
application/foo/views/
application/bar/
application/bar/config/
application/bar/controllers/
application/bar/errors/
application/bar/libraries/
application/bar/models/
application/bar/views/
```

Para seleccionar una aplicación en particular para su uso, se necesita que abra su archivo **index.php** principal y establezca variable **\$application_folder**. Por ejemplo, para seleccionar la aplicación "foo" para su uso, debería hacer esto:

```
$application_folder = "application/foo";
```

Nota: Cada una de sus aplicaciones necesitará su propio archivo **index.php** que llama la aplicación deseada. El archivo **index.php** puede llamarse de la manera que desee.

Sintaxis Alternativa de PHP para Archivos de Vistas

Si no utiliza el motor de plantillas de CodeIgniter, estará usando PHP puro en sus archivos de vistas. Para minimizar el código PHP en esos archivos, y para hacerlo más fácil para identificar bloques de código, se recomienda que use la sintaxis alternativa de PHP para estructuras de control y sentencias de eco con etiquetas cortas. Si no está familiarizado con esta sintaxis, ésta le permite eliminar las llaves de su código, y eliminar las sentencias "echo".

Soporte Automático para Etiquetas Cortas

Nota: Si encuentra que la sintaxis descripta en esta página no funciona en su servidor esto puede deberse a que "short tags" está deshabilitado en su archivo ini de PHP. CodeIgniter escribirá opcionalmente las etiquetas cortas al vuelo, permitiéndole usar esa sintaxis aún si su servidor no la soporta. Esta funcionalidad se puede habilitar en su archivo **application/config/config.php**.

Por favor advierta que si usa esta funcionalidad, si se encuentran errores de PHP en sus archivos de vistas, el número de línea y mensaje de error no se mostrarán con precisión. En cambio, todos los errores se mostrarán como errores de **eval()**.

Ecos Alternativos

Normalmente para hacer eco o imprimir una variable haría esto:

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

Con la sintaxis alternativa podría hacer eso de esta otra forma:

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

Estructuras de Control Alternativas

Las estructuras de control, como **if**, **for**, **foreach**, y **while** se pueden escribir también de forma simplificada. Aquí hay un ejemplo usando **foreach**:

```
<ul>

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

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

<?php endforeach; ?>

</ul>
```

Advierta que no hay llaves. En cambio, la llave final se reemplazó con **endforeach**. Cada una de las estructuras de control listadas anteriormente tiene una sintaxis similar de cierre: **endif**, **endfor**, **endforeach**, y **endwhile**.

También advierta que en lugar de usar un punto y coma después de cada estructura (excepto en la última), hay dos puntos. ¡Esto es importante!

Aquí hay otro ejemplo usando **if/elseif/else**. Advierta los dos puntos:

```
<?php if ($usuario == 'sara'): ?>  
    <h3>Hola Sara</h3>  
<?php elseif ($usuario == 'jose'): ?>  
    <h3>Hola Jose</h3>  
<?php else: ?>  
    <h3>Hola usuario desconocido</h3>  
<?php endif; ?>
```

Seguridad

Esta página describe algunas "buenas prácticas" en materia de seguridad web y detalles de características de la seguridad interna de CodeIgniter.

Seguridad URI

CodeIgniter es bastante restrictivo en cuanto a los caracteres que permite en las cadenas URI para ayudar a minimizar las posibilidades que datos maliciosos se puedan pasar a la aplicación. Las URIs solamente pueden contener lo siguiente:

- Texto alfanumérico
- Tilde: ~
- Punto: .
- Dos puntos: :
- Guión de subrayado: _
- Guión: -

Register_globals

Durante la **inicialización** del sistema todas las **variables globales se destruyen**, excepto aquellas que se encuentran en los arrays **`$_POST`** y **`$_COOKIE`**. La rutina de destrucción es lo mismo que hacer **`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 Manejar Varios Entornos.

magic_quotes_runtime

La directiva **`magic_quotes_runtime`** se desactiva durante la inicialización del sistema, por lo que no tiene que quitar las barras cuando recupere datos desde la base de datos.

Buenas Prácticas

Antes de aceptar cualquier datos en su aplicación, ya sean datos **`POST`** desde el envío de un formulario, datos de **`COOKIE`**, datos de URI, datos XML-RPC data, o aún datos desde el array **`SERVER`**, se le recomienda practicar este enfoque de tres pasos:

- Filtrar los datos como si estuvieran contaminados.
- Validar los Datos para asegurar que cumplen con el tipo correcto, longitud, tamaño, etc. (a veces este paso puede reemplazar al anterior).
- Escapar los datos antes de meterlos en la base de datos.

CodeIgniter provee las siguientes funciones para asistirlo en este proceso:

- **Filtrado XSS**
CodeIgniter viene con un filtro contra XSS (Cross Site Scripting). Este filtro busca técnicas comúnmente

usadas para embeber código Javascript malicioso en sus datos u otro tipo de código que intente secuestrar cookies o hacer otras cosas maliciosas. El Filtro de XSS se describe aquí.

- **Validar los Datos**
CodeIgniter tiene una Clase para Validación de Formularios que lo ayuda en la validación, filtrado y preparación de sus datos.
- **Escapar Todos los Datos Antes de Insertarlos en la Base de Datos**
Nunca inserte información en su base de datos sin escaparla. Para más información, por favor, lea la sección que discute las consultas.

Estilo y Sintaxis Generales

La siguiente página 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 `?>` es opcional para el analizador de PHP. Sin embargo, si se usa, cualquier espacio 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 las 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 **contenido**. 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 Clase
 *
 * @package Nombre del paquete
 * @subpackage Subpaquete
 * @category Categoría
 * @author Nombre del autor
 * @link http://ejemplo.com
 */
class Super_clase {
```

```
/**
 * Codifica una cadena para usarla en XML
 *
 * @access public
 * @param string
 * @return string
 */
function xml_encode($str)
```

Usar 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 // falta el guion de subrayado y no est&acute;a completamente en may&usculas
N           // no hay constantes de una sola letra
S_C_VER     // no es descriptivo
$str = str_replace('{foo}', 'bar', $str); // deberia usar las constantes LD y RD
```

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 $bar) // ok, pero no se recomienda para las aplicaciones comunes de
                    // resaltado de sintaxis
if (!$foo)
if (! is_array($foo))
```

CORRECTO:

```
if ($foo OR $bar)
if ($foo && $bar) // recomendado
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 5.1 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 carácter. 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 carácter 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.

INCORRECTO:

```
$arr[ $foo ] = 'foo';
```

CORRECTO:

```
$arr[$foo] = 'foo'; // no hay espacios alrededor de la clave del array
```

INCORRECTO:

```
function foo ( $bar )
{
}
```

CORRECTO:

```
function foo($bar) // no hay espacios alrededor de los paréntesis en declaraciones
                  // de funciones
{
```

INCORRECTO:

```
foreach( $query->result() as $row )
```

CORRECTO:

```
foreach ($query->result() as $row) // un solo espacio siguiendo las estructuras de
                                    // control de PHP, pero no en paréntesis
```

```
// interiores
```

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 "Selección inválida";
```

CORRECTO:

```
return $this->lang->line('seleccion_invalida');
```

Métodos Privados y Variables

Se deberían prefijar con un guion 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 "Habilitado";  
}
```

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:

```
"Mi cadena"                                // no hay análisis de variables, por
                                            // lo tanto no use comillas dobles
"Mi cadena $foo"                            // se necesitan llaves
'SELECT foo FROM bar WHERE baz = \'bag\''  // repugnante
```

CORRECTO:

```
'Mi cadena'
"Mi cadena {$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 consultas 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)
```

Escribir Documentación

Para ayudar a escribir un estilo de documentación fácil de leer y consistente para proyectos CodeIgniter, EllisLab le da libremente a la Comunidad el código de la Guía del Usuario de CodeIgniter para su uso. Para su comodidad se creó un archivo de plantilla que incluye bloques de marcado usados con ejemplos breves.

Archivos

- **Hoja de Estilos**

Vea el archivo [/user_guide/userguide.css](#) de su instalación.

- **Plantilla de Página**

Vea el archivo [/user_guide/doc_style/template.html](#) de su instalación.



Referencia de Clases

Clase Benchmark

CodeIgniter tiene una clase para evaluar el desempeño que está siempre activa, permitiendo que se calcule la diferencia de tiempo entre dos puntos cualesquiera marcados.

Nota: El sistema inicializa automáticamente esta clase, por lo que no hay necesidad de hacerlo manualmente.

Además, la evaluación de desempeño siempre arranca en el momento que se invoca al framework, y la termina la clase **Output**, justo antes de enviar la vista final al navegador, permitiendo mostrar una medición de tiempo muy exacta de la ejecución del sistema al completo.

Usar la Clase Benchmark

La Clase **Benchmark** se puede usar dentro de sus controladores, vistas, o modelos. El proceso de uso es:

1. Marcar un punto de inicio
2. Marcar un punto de fin
3. Ejecutar la función "elapsed time" para ver el resultado

Este es un ejemplo que usa código real:

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

// Algún código aquí

$this->benchmark->mark('codigo_fin');

echo $this->benchmark->elapsed_time('codigo_inicio', 'codigo_fin');
```

Nota: Las palabras "codigo_inicio" y "codigo_fin" son arbitrarias. Se usan simplemente para establecer dos marcadores. Se puede usar cualquier palabra y se pueden establecer varios conjuntos de marcadores. Consideré este ejemplo:

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

// Algún código aquí

$this->benchmark->mark('gato');

// Más código aquí

$this->benchmark->mark('ave');

echo $this->benchmark->elapsed_time('perro', 'gato');
echo $this->benchmark->elapsed_time('gato', 'ave');
echo $this->benchmark->elapsed_time('perro', 'ave');
```

Perfilar sus Puntos de Evaluación

Si quiere que sus datos de evaluación de desempeño estén disponibles para el Perfilador, todos sus puntos marcados se tienen que configurar de a pares, y cada nombre de punto de marca tiene que terminar con **_start** y **_end**. Por otra parte, cada par de puntos se tiene que llamar igual. Ejemplo:

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

// Algún código aquí...

$this->benchmark->mark('mi_marca_end');

$this->benchmark->mark('otra_marca_start');

// Más código aquí...

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

Para mayor información, lea la página del Perfilador.

Mostrar el Tiempo Total de Ejecución

Si quisiera mostrar el tiempo total transcurrido desde el momento en que CodeIgniter arranca al momento final en que la salida se envía al navegador, simplemente ubique esto en una de las líneas de la plantilla de vista:

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

Advertirá que es la misma función usada en el ejemplo anterior para calcular el tiempo entre dos puntos, salvo que **no** se usa ningún parámetro. Cuando los parámetros están ausentes, CodeIgniter no detiene la evaluación de desempeño sino justo antes cuando la salida final se envía al navegador. No importa donde ponga la llamada a la función, el temporizador continuará corriendo hasta el final.

Una forma alternativa para mostrar el tiempo transcurrido en sus archivos de vista, es usar esta seudo-variable, si prefiere no usar PHP puro:

```
{elapsed_time}
```

Nota: Si quiere evaluar cualquier cosa dentro de su controlador, tiene que establecer sus propios puntos de inicio/fín.

Mostrar el Consumo de Memoria

Si su instalación de PHP está configurada con **--enable-memory-limit**, puede mostrar la cantidad de memoria consumida por el sistema entero usando el siguiente código en uno de los archivos de vista:

```
<?php echo $this->benchmark->memory_usage();?>
```

Nota: Esta función solamente se puede usar en sus archivos de vista. El consumo reflejará el total de memoria usada por la aplicación completa.

Una forma alternativa para mostrar el uso de la memoria en sus archivos de vista, es usar esta seudo-variable, si prefiere no usar PHP puro:

```
{memory_usage}
```

Clase Calendar

La Clase **Calendar** le permite **crear calendarios dinámicamente**. Sus calendarios se puede **formatear** a través del uso de una **plantilla** de calendario, permitiendo control al 100% todos los aspectos de su diseño. Además, puede pasarle **datos** a las **celdas del calendario**.

Iniciar la Clase

Como la mayoría de las clases en CodeIgniter, la Clase **Calendar** se **inicializa** en su **controlador** usando la **función \$this->load->library**:

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

la clase/libreria

Una vez cargada, el objeto Calendar estará disponible usando: **\$this->calendar**.

Mostrar un Calendario

Aquí hay un **ejemplo** simple que muestra **como** puede **mostrar** un **calendario**:

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

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

El código anterior generará un calendario para el mes/año actuales basado en la fecha del servidor. Para mostrar un calendario para un mes y año específicos, le pasará esa información a la función de generación de calendarios:

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

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

El código anterior generará un calendario que muestra el mes de junio de 2006. El primer parámetro especifica el año y el segundo parámetro especifica el mes.

Pasar Datos a las Celdas del Calendario

Agregar **datos** a las **celdas** de su **calendario** implica **crear** un **array asociativo** en el que las **claves** corresponden a los **días** que **desea llenar** y los **valores** del **array** **contienen** los **datos**. El **array** se **pasa como tercer parámetro** de la **función de generación de calendarios**. Considere este ejemplo:

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

$data = array(
    3 => 'http://ejemplo.com/noticias/articulo/2006/03/',
    7 => 'http://ejemplo.com/noticias/articulo/2006/07/',
    13 => 'http://ejemplo.com/noticias/articulo/2006/13/',
    26 => 'http://ejemplo.com/noticias/articulo/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 asume que su array contendrá enlaces. En la sección que explica las plantillas de calendario, verá cómo puede personalizar la forma en que se pasan los datos a las celdas, por lo que puede pasar distinto tipo de información.

Establecer las Preferencias de Visualización

Hay siete preferencias que puede establecer para controlar varios aspectos del calendario. Las preferencias se establecen al pasar un array de preferencias en el segundo parámetro de la función de carga. Aquí hay un ejemplo:

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

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

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

El código anterior comenzaría el calendario en sábado, usando el encabezado de mes "largo" y los nombres de los días "cortos". Más información acerca de las preferencias, más abajo.

Preferencia	Por Defecto	Opciones	Descripción
template	Ninguno	Ninguno	Una cadena conteniendo la plantilla de calendario. Vea la sección de plantillas más abajo.
local_time	time()	Ninguno	Una marca de tiempo de Unix correspondiente a la hora actual.
start_day	sunday	Cualquier día de la semana (sunday, monday, tuesday, etc.)	Establece el día de la semana con el que el calendario debería comenzar.
month_type	long	long, short	Determina qué versión del nombre del mes usa el encabezado. long = January, short = Jan.
day_type	abr	long, short, abr	Determina qué versión del nombre del día de la semana usan los encabezados de columna. long = Sunday, short = Sun, abr = Su.
show_next_prev	FALSE	TRUE/FALSE (booleano)	Determina si mostrar enlaces que le permiten cambiar a meses siguiente/anterior. Ver información acerca de esta funcionalidad más abajo.
next_prev_url	Ninguno	Una URL	Establece la ruta básica usada en los enlaces de calendario siguiente/anterior.

Mostrar Enlaces de Mes Siguiente/Anterior

Para permitirle a su calendario incrementar/decrementar dinámicamente mediante los enlaces siguiente/anterior, se necesita que configure el código de su calendario similar a este ejemplo:

```
$prefs = array (
    'show_next_prev' => TRUE,
    'next_prev_url'   => 'http://ejemplo.com/index.php/calendar/show/'
);

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

echo $this->calendar->generate($this->uri->segment(3), $this->uri->segment(4));
```

Advertirá unas pocas cosas acerca del ejemplo anterior:

- Tiene que establecer "show_next_prev" a **TRUE**.
- Tiene que proporcionar la URL al controlador que contiene su calendario en la preferencia "next_prev_url".
- Tiene que proporcionar el "año" y "mes" a la función que genera el calendario mediante segmentos URI donde aparecen (**Nota:** La Clase Calendar automáticamente agrega el año/mes a la URL base que Ud proveyó).

Crear una Plantilla de Calendario

Al crear una plantilla de calendario, Ud tiene el control total sobre el diseño del calendario. Cada componente de su calendario se ubicará dentro de un par de seudo-variables como se muestra aquí:

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


```

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

Clase Cart

X Nulo

La Clase **Cart** permite que los ítems se agreguen a la sesión que permanece activa mientras un usuario está navegando su sitio. Estos ítems se puede recuperar y mostrar en un formato estándar de "carrito de compras", permitiéndole al usuario actualizar o quitar ítems del carrito.

Por favor advierta que la Clase **Cart** SOLAMENTE provee funcionalidad de "carrito". No provee funcionalidades de envío, autorización de tarjeta de crédito u otro procesamiento adicional.

Inicializar la Clase Cart

Importante: La Clase **Cart** usa la Clase **Session** de CodeIgniter para guardar la información del carrito en la base de datos, por lo tanto, antes de usar la Clase **Cart** tiene que establecer una tabla de base de datos como se indica en la documentación de la Clase **Session** y establecer las preferencias de la sesión en su archivo **application/config/config.php** para utilizar una base de datos.

Para inicializar la Clase **Cart** en su controlador, use la función **\$this->load->library**:

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

Una vez cargada, el objeto Cart estará disponible usando: **\$this->cart**.

Nota: La Clase **Cart** cargará e inicializará a la Clase **Session** automáticamente, por lo que a menos que esté usando sesiones en otra parte de su aplicación, no necesita cargar la Clase **Session**.

Agregar un Ítem al Carrito

Para agregar un ítem al carrito, simplemente pase un array con la información del producto a la función **\$this->cart->insert()**, según se muestra aquí:

```
$data = array(
    'id'      => 'sku_123ABC',
    'qty'     => 1,
    'price'   => 39.95,
    'name'    => 'T-Shirt',
    'options' => array('Size' => 'L', 'Color' => 'Red')
);

$this->cart->insert($data);
```

Importante: Los primeros cuatro índices del array anterior (id, qty, price, and name) son obligatorios. Si omite alguno de ellos, los datos no se guardarán en el carrito. El quinto índice (options) es opcional. Está pensado para usarse en casos donde su producto tiene opciones asociadas con él. Use un array para opciones, como se muestra arriba.

Los cinco índices reservados son:

- **id** - Cada producto en el almacén tiene un identificador único. Normalmente será un "sku" u otra identificación.
- **qty** - Es la cantidad que se está comprando.
- **price** - Es el precio del ítem.
- **name** - Es el nombre del ítem.
- **options** - Cualquier atributo adicional que es necesario para identificar al producto. Tienen que pasarse mediante un array.

Además de los cinco índices de arriba, hay dos palabras reservadas: **rowid** y **subtotal**. Se usan internamente en la Clase Cart, por lo que **NO** debe usar esas palabras como índices al insertar datos en el carrito.

Un array puede contener datos adicionales. Cualquier cosa que incluya en el array, se guardará en la sesión. Sin embargo, lo mejor es la estandarización de los datos entre todos sus productos con el fin de mostrar la información en una tabla más fácil.

Agregar Varios Ítems al Carrito

Es posible agregar en una sola acción varios productos al carrito al usar un array multidimensional. Esto es útil en casos donde desea permitirle a la gente seleccionar entre varios ítems de la misma página.

```
$data = array(
    array(
        'id'      => 'sku_123ABC',
        'qty'     => 1,
        'price'   => 39.95,
        'name'    => 'T-Shirt',
        'options' => array('Size' => 'L', 'Color' => 'Red')
    ),
    array(
        'id'      => 'sku_567ZYX',
        'qty'     => 1,
        'price'   => 9.95,
        'name'    => 'Coffee Mug'
    ),
    array(
        'id'      => 'sku_965QRS',
        'qty'     => 1,
        'price'   => 29.95,
        'name'    => 'Shot Glass'
    )
);

$this->cart->insert($data);
```

Mostrar el Carrito

Para mostrar el carrito creará un archivo de vista con código similar al mostrado a continuación.

Por favor advierta que este ejemplo usa el helper form.

```
<?php echo form_open('ruta/al/controller/update/function'); ?>
<table cellpadding="6" cellspacing="1" style="width:100%" border="0">
<tr>
```

```
<th>Cantidad</th>
<th>Descripción</th>
<th style="text-align:right">Precio</th>
<th style="text-align:right">Sub-Total</th>
</tr>

<?php $i = 1; ?>

<?php foreach ($this->cart->contents() as $items): ?>

<?php echo form_hidden($i.'[rowid]', $items['rowid']); ?>

<tr>
    <td><?php echo form_input(array('name' => $i.'[qty]',
                                    'value' => $items['qty'],
                                    'maxlength' => '3',
                                    'size' => '5')); ?>
    </td>
    <td>
        <?php echo $items['name']; ?>
        <?php if ($this->cart->has_options($items['rowid'])) == TRUE): ?>
        <p>
            <?php foreach ($this->cart->product_options($items['rowid']) as
                $option_name => $option_value): ?>
            <strong><?php echo $option_name; ?></strong>
            <?php echo $option_value; ?><br />
        <?php endforeach; ?>
        </p>
        <?php endif; ?>
    </td>
    <td style="text-align:right">
        <?php echo $this->cart->format_number($items['price']); ?>
    </td>
    <td style="text-align:right">
        $<?php echo $this->cart->format_number($items['subtotal']); ?>
    </td>
</tr>

<?php $i++; ?>

<?php endforeach; ?>

<tr>
    <td colspan="2"></td>
    <td class="right"><strong>Total</strong></td>
    <td class="right">
        $<?php echo $this->cart->format_number($this->cart->total()); ?>
    </td>
</tr>

</table>

<p><?php echo form_submit('', 'Update your Cart'); ?></p>
```

Actualizar el Carrito

Para actualizar la información del carrito, tiene que pasar un array conteniendo el **Row ID** y la cantidad a la función **\$this->cart->update()**:

Nota: Si la cantidad se establece a cero, se eliminará el ítem del carrito.

```
$data = array(
    'rowid' => 'b99ccdf16028f015540f341130b6d8ec',
    'qty'    => 3
);

$this->cart->update($data);

// O un array multidimensional

$data = array(
    array(
        'rowid' => 'b99ccdf16028f015540f341130b6d8ec',
        'qty'    => 3
    ),
    array(
        'rowid' => 'xw82g9q3r495893iajdh473990rikw23',
        'qty'    => 4
    ),
    array(
        'rowid' => 'fh4kdkkkaoe30njgoe92rkdkkobec333',
        'qty'    => 2
    )
);

$this->cart->update($data);
```

¿Qué es un Row ID? El **row ID** es un identificador único que lo genera el código del carrito cuando se agrega un ítem al carrito. La razón por la que se crea un identificador único es para que los mismos productos con diferentes opciones puedan ser administrados por el carrito.

Por ejemplo, digamos que alguien compra dos camisetas iguales (mismo ID de producto), pero diferentes tamaños. El ID de producto (y otros atributos) serán idénticos para ambos tamaños porque es la misma camiseta. La única diferencia será el tamaño. El carrito por lo tanto, tiene un medio para identificar esta diferencia para que los dos tamaños de camisetas se puedan manejar independientemente. Lo hace mediante la creación de un único "row ID" basado en el ID de producto y algunas opciones asociadas a él.

En casi todos los casos, la actualización del carrito será algo que el usuario hace mediante la página "ver carrito", por lo tanto como desarrollador, es poco probable que tenga que preocuparse por el "row ID", aparte de asegurarse que su página "ver carrito" contenga esta información en un campo oculto de formulario y asegurarse que se pasa a la función de actualización cuando se envía el formulario de actualización. Para mayor información, examine más abajo la construcción de la página "ver carrito".

Referencia de Funciones

\$this->cart->insert();

Le permite agregar ítems al carrito de compras, como se indica más arriba.

\$this->cart->update();

Le permite actualizar ítems en el carrito de compras, como se indica más arriba.

\$this->cart->total();

Muestra el total del carrito.

\$this->cart->total_items();

Muestra la cantidad de total de ítems en el carrito.

\$this->cart->contents();

Devuelve un array conteniendo todo lo del carrito.

\$this->cart->has_options(rowid);

Devuelve **TRUE** (booleano) si una fila en particular del carrito contiene opciones. Se diseñó esta función para usarla dentro de un bucle con **\$this->cart->contents()**, ya que tiene que pasar el **rowid** a esta función, como se muestra en el ejemplo de Mostrar el Carrito más arriba.

\$this->cart->product_options(rowid);

Devuelve un array de opciones para un producto en particular. Se diseñó esta función para usarla dentro de un bucle con **\$this->cart->contents()**, ya que tiene que pasar el rowid a esta función, como se muestra en el ejemplo de Mostrar el Carrito más arriba.

\$this->cart->destroy();

Le permite destruir el carrito. Esta función probablemente sea llamada cuando esté terminando el procesamiento del pedido del cliente.

Clase Config

La Clase **Config** provee un medio para recuperar preferencias de configuración. Estas preferencias pueden venir de un archivo de configuración por defecto (**application/config/config.php**) o de sus propios archivos de configuración personalizados.

Nota: El sistema inicializa automáticamente esta clase, por lo que no hay necesidad de hacerlo manualmente.

Anatomía de un Archivo de Configuración

Por defecto, CodeIgniter tiene un archivo primario de configuración, localizado en **application/config/config.php**. Si abre el archivo usando su editor de texto verá que los ítems de configuración se almacenan en un array llamado **\$config**.

Puede agregar sus propios ítems de configuración a este archivo, o si prefiere mantener sus ítems de configuración separados (asumiendo que necesita ítems de configuración), simplemente cree su propio archivo y guárdelo en la carpeta **config**.

Nota: Si crea sus propios archivos de configuración, use el mismo formato que el del primario, almacenando sus ítems en un array llamado **\$config**. CodeIgniter administrará inteligentemente estos archivos, por lo que no habrá conflicto aún cuando el array tenga el mismo nombre (asumiendo que no se repiten los índices del array).

Cargar un Archivo de Configuración

Nota: CodeIgniter carga automáticamente el archivo primario de configuración (**application/config/config.php**), por lo que solamente necesitará cargar un archivo que haya creado Ud mismo.

Hay dos formas de cargar un archivo de configuración:

- **Carga Manual**

Para cargar uno de sus archivos de configuración, usará la siguiente función dentro del controlador que lo necesite:

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

Donde **nombre_de_archivo** es el nombre de su archivo de configuración sin la extensión .php.

Si necesita cargar varios archivos de configuración, normalmente se fusionarán en un array de configuración maestro. Sin embargo, pueden ocurrir colisiones de nombres si nombra del mismo modo índices de array en archivos de configuración diferentes. Para evitar colisiones puede establecer el segundo parámetro a **TRUE** y cada archivo de configuración se almacenará en un índice de array correspondiente al nombre del archivo de configuración. Ejemplo:

```
// Almacenado en un array con este prototipo:  
// $this->config['blog_settings'] = $config  
$this->config->load('blog_settings', TRUE);
```

Por favor vea más abajo la sección llamada Recuperar Items de Configuración para aprender como recuperar los ítems de configuración establecidos de esta forma.

El tercer parámetro le permite suprimir los errores en caso que no exista el archivo de configuración:

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

- **Carga Automática**

Si necesita tener globalmente un archivo de configuración en particular, puede hacérselo cargar automáticamente al sistema. Para hacer esto, abra el archivo **autoload.php** ubicado en **application/config/autoload.php** y agregue su archivo de configuración como se indica en el archivo.

Recuperar Items de Configuración

Para recuperar un ítem desde su archivo de configuración, use la siguiente función:

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

Donde **nombre_item** es el índice del array **\$config** que quiere recuperar. Por ejemplo, para recuperar su elección de idioma usará esto:

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

La función devuelve **FALSE** (booleano) si el ítem que está intentado recuperar no existe.

Si está usando el segundo parámetro de la función **\$this->config->load** a fin de asignar sus ítems de configuración a un índice específico, puede recuperarlo al especificar el nombre del índice en el segundo parámetro de la función **\$this->config->item()**. Ejemplo:

```
// Carga un archivo de configuración llamado blog_settings.php y lo asigna a un
// índice llamado "blog_settings"
$this->config->load('blog_settings', TRUE);

// Devuelve un ítem de configuración llamado site_name contenido dentro del array
// blog_settings
$site_name = $this->config->item('site_name', 'blog_settings');

// Una manera alternativa de especificar el mismo ítem:
$blog_config = $this->config->item('blog_settings');
$site_name = $blog_config['site_name'];
```

Establecer un Item de Configuración

Si quisiera establecer dinámicamente un ítem de configuración o cambiar uno existente, puede hacerlo usando:

```
$this->config->set_item('nombre_item', 'valor_item');
```

Donde **nombre_item** es el índice del array **\$config** que quiere cambiar, y **valor_item** su valor.

Entornos

Puede cargar distintos archivos de configuración en el entorno actual. La constante **ENVIRONMENT** está definida en **index.php**, y se describe en detalle en la sección Manejar Varios Entornos.

Para crear un archivo de configuración específico de un entorno, crear una copia de un archivo de configuración en **application/config/{ENVIRONMENT}/{FILENAME}.php**

Por ejemplo, para crear **config.php** de producción únicamente, haría:

1. Crear el directorio **application/config/producción/**
2. Copiar su archivo **config.php** existente en el directorio anterior
3. Editar **application/config/producción/config.php** para que contenga los parámetros de producción

Cuando establezca la constante **ENVIRONMENT** 'producción', se cargarán los valores para su nuevo archivo **config.php** solamente de producción.

Puede colocar los siguientes archivos de configuración en carpetas específicas del entorno:

- Archivos de configuración de CodeIgniter por defecto
- Sus propios archivos de configuración personalizados

Nota: CodeIgniter siempre intenta cargar primero los archivos de configuración para el entorno actual. Si el archivo no existe, el archivo de configuración global (es decir, se carga el de **application/config/**). Esto significa que no está obligado a ubicar todos sus archivos de configuración en una carpeta de entorno – solamente los archivos que cambian por entorno.

Funciones Helper

La Clase **Config** tiene las siguiente funciones helper:

\$this->config->site_url()

Esta función recupera la URL para su sitio, junto con el valor "index" que especificó en el archivo de configuración.

\$this->config->base_url()

Esta función devuelve la URL de su sitio, junto con una ruta opcional tal como la de una hoja de estilo o imagen.

Las dos funciones anteriores se acceden normalmente mediante las funciones correspondientes en el Helper URL.

\$this->config->system_url()

Esta función recupera la URL para su carpeta **system**.

Clase Database

CodeIgniter viene con una clase de **base de datos abstracta** muy rápida y completa que soporta tanto las estructuras tradicionales como los patrones **Active Record**. Las funciones de base de datos ofrecen una sintaxis clara y sencilla.

Inicio Rápido: Código de Ejemplo

La siguiente página contiene código de ejemplo mostrando como se usa la clase **database**. Para obtener detalles completos, por favor lea las páginas individuales que describen cada función.

Iniciar la Clase Database

El siguiente código **carga** e **inicializa** la clase **database** basado en los valores de su configuración:

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

Una vez cargada la clase, está lista para usarse como se describe a continuación.

Nota: Si **todas las páginas necesitan acceso**, se puede conectar automáticamente. Vea la página de conexión para más detalles.

Consulta Estándar con Resultados Múltiples (Versión Objetos)

```
$query = $this->db->query('SELECT nombre, titulo, email FROM mi_tabla');

foreach ($query->result() as $row)
{
    echo $row->titulo;
    echo $row->nombre;
    echo $row->email;
}

echo 'Resultados totales: ' . $query->num_rows();
```

La función **result()** anterior **devuelve** un **array** de **objetos**. Ejemplo: **\$row->titulo**.

Consulta Estándar con Resultados Múltiples (Versión Array)

```
$query = $this->db->query('SELECT nombre, titulo, email FROM mi_tabla');

foreach ($query->result_array() as $row)
{
    echo $row['titulo'];
    echo $row['nombre'];
    echo $row['email'];
}
```

La función **result_array()** anterior **devuelve** un **array** de **índices** de **array** estándar. Ejemplo: **\$row['titulo']**.

Comprobar Resultados

Si ejecuta consultas que pueden no producir resultados, le aconsejamos probar primero de usar la función `num_rows()`:

```
$query = $this->db->query("SU CONSULTA");

if ($query->num_rows() > 0)
{
    foreach ($query->result() as $row)
    {
        echo $row->titulo;
        echo $row->nombre;
        echo $row->cuerpo;
    }
}
```

Consulta Estándar con Resultado Simple

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

$row = $query->row();
echo $row->nombre;
```

La función `row()` anterior devuelve un objeto. Ejemplo: `$row->nombre`.

Consulta Estándar con Resultado Simple (Versión Array)

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

$row = $query->row_array();
echo $row['nombre'];
```

La función `row_array()` anterior devuelve un array. Ejemplo: `$row['nombre']`.

Inserción Estándar

```
$sql = "INSERT INTO mi_tabla (titulo, nombre)
       VALUES ('".$this->db->escape($titulo)."', '".$this->db->escape($nombre)."')";

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

echo $this->db->affected_rows();
```

Consulta Active Record

El Patrón **Active Record** le da una forma simplificada de devolver los datos:

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

foreach ($query->result() as $row)
{
```

```
    echo $row->titulo;  
}
```

La función **get()** anterior devuelve todos los resultados desde la tabla suministrada. La clase **Active Record** contiene un complemento completo de funciones para trabajar con datos.

Inserción Active Record

```
$data = array(  
    'titulo' => $titulo,  
    'nombre' => $nombre,  
    'fecha' => $fecha  
);  
  
$this->db->insert('mi_tabla', $data);  
  
// Produce: INSERT INTO mi_tabla (titulo, nombre, fecha) VALUES ('{$titulo}',  
// '{$nombre}', '{$fecha}')
```

Configuración de la Base de Datos

CodeIgniter tiene un archivo de configuración que le permite almacenar los valores de conexión de la base de datos (usuario, contraseña, nombre de la base de datos, etc.). El archivo de configuración está ubicado en **application/config/database.php**. También puede establecer valores de conexión de base de datos para entornos específicos al ubicar **database.php** en la carpeta de configuración del entorno respectivo.

Los parámetros de configuración se almacenan en un array multidimensional con este prototipo:

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "root";
$db['default']['password'] = "";
$db['default']['database'] = "nombre_de_base_de_datos";
$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 razón por la que usamos un array multidimensional en lugar de uno más simple, es para permitirle almacenar opcionalmente varios conjuntos de valores de conexión. Si, por ejemplo, ejecuta varios entornos (desarrollo, producción, prueba, etc.) bajo una instalación simple, puede configurar un grupo de conexión para cada uno y luego cambiar entre grupos según se necesite. Por ejemplo, para configurar un entorno "prueba" podría hacer esto:

```
$db['prueba']['hostname'] = "localhost";
$db['prueba']['username'] = "root";
$db['prueba']['password'] = "";
$db['prueba']['database'] = "nombre_de_base_de_datos";
$db['prueba']['dbdriver'] = "mysql";
$db['prueba']['dbprefix'] = "";
$db['prueba']['pconnect'] = TRUE;
$db['prueba']['db_debug'] = FALSE;
$db['prueba']['cache_on'] = FALSE;
$db['prueba']['cachedir'] = "";
$db['prueba']['char_set'] = "utf8";
$db['prueba']['dbcollat'] = "utf8_general_ci";
$db['prueba']['swap_pre'] = "";
$db['prueba']['autoinit'] = TRUE;
$db['prueba']['stricton'] = FALSE;
```

Entonces, para decirle globalmente al sistema que use ese grupo, podría establecer esta variable localizada en el archivo de configuración:

```
$active_group = "prueba";
```

Nota: El nombre "prueba" es arbitrario. Puede ser cualquiera que desee. Por defecto usamos la palabra "default" para la conexión primaria, pero también se la puede renombrar a algo más relevante en el proyecto.

Active Record

La Clase **Active Record** se habilita o deshabilita globalmente estableciendo la variable **\$active_record** en el archivo de configuración de la base de datos a **TRUE/FALSE** (booleano). Si no está usando la Clase **Active Record**, establecer el valor a **FALSE** usará menos recursos cuando la Clase **Database** esté inicializada.

```
$active_record = TRUE;
```

Nota: Algunas **clases** de CodeIgniter, tales como **Sessions**, necesitan que Active Records esté habilitado para acceder a ciertas funcionalidades.

Explicación de valores:

- **hostname** - El nombre del host de su servidor de base de datos. Frecuentemente es "localhost".
- **username** - El usuario utilizado para conectar con la base de datos.
- **password** - La contraseña utilizada para conectar con la base de datos.
- **database** - El nombre de la base de datos con la que se quiere conectar.
- **dbdriver** - El **tipo** de **base de datos**. Por ejemplo: mysql, postgres, odbc, etc. Tiene que especificarse en minúsculas.
- **dbprefix** - Un prefijo opcional para tablas que se agregará al nombre de las tablas al ejecutar consultas del Active Record. Esto permite que varias instalaciones de CodeIgniter comparten una sola base de datos.
- **pconnect** - **TRUE/FALSE** (booleano) - Si desea usar una conexión persistente.
- **db_debug** - **TRUE/FALSE** (booleano) - Si se tienen que mostrar los errores de la base de datos.
- **cache_on** - **TRUE/FALSE** (booleano) - Si está habilitado el cacheo de consultas, ver también el Driver Caché.
- **cachedir** - La ruta absoluta en el servidor del directorio de cacheo de consultas de la base de datos.
- **char_set** - El conjunto de caracteres usado en la comunicación con la base de datos.
- **dbcollat** - La codificación de caracteres usada en la comunicación con la base de datos.

Nota: Para bases de datos MySQL y MySQLi, este parámetro se usa solamente como una copia de respaldo si el servidor está corriendo PHP < 5.2.3 o MySQL < 5.0.7 (y las consultas de creación de tablas se hicieron con DB Forge). Hay una incompatibilidad en PHP con **mysql_real_escape_string()** que puede hacer su sitio vulnerable a inyecciones de SQL si está usando un conjunto de caracteres multi-byte y se están corriendo versiones menores a esas. Los sitios que usan conjuntos de caracteres y ordenamiento Latin-1 o UTF-8 no están afectados.

- **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 número de puerto de base de datos. Para usar este valor tiene que agregar una línea al array de configuración de la base de datos.

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

Nota: Dependiendo de que plataforma esté usando (MySQL, Postgres, etc.) no se necesitarán todos los valores. Por ejemplo, al usar SQLite no necesitará suministrar un usuario o contraseña, y la base de datos será la ruta a su archivo de base de datos. La información anterior asume que está usando MySQL.

Conectarse a una Base de Datos

Hay dos formas de conectar a una base de datos:

Conectarse Automáticamente

La función "auto conectar" cargará e instanciará la clase **database** con cada carga de página. Para habilitar la "auto conexión", agregar la palabra **database** al array de la **biblioteca**, como se indica en el siguiente archivo:

```
application/config/autoload.php
```

Conectarse Manualmente

Si solamente algunas páginas requieren de conectividad de base de datos, puede conectar manualmente a la base de datos agregando esta línea de código en cualquier función donde se la necesite, o en el constructor de la clase para hacer que la base de datos sea global a esa clase.

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

Si la función anterior **no** contiene información alguna en el primer parámetro, conectará con el grupo especificado en su archivo de configuración de base de datos. Para la mayoría de la gente, este es el método preferido.

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 Conectarse a Varias Bases de Datos).
3. **TRUE/FALSE** (booleano). Si habilitar la Clase **Active Record**. Establecido a **TRUE** por defecto.

Conectarse Manualmente a una Base de Datos

El primer parámetro de esta función se puede usar **opcionalmente** para especificar un grupo de base de datos en particular en el archivo de configuración, o incluso puede presentar valores de conexión para una base de datos que no se especifica en su archivo de configuración. Ejemplos:

Para elegir un grupo específico del archivo de configuración puede hacer esto:

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

Donde **nombre_de_grupo** es el nombre del grupo de conexión en su archivo de configuración.

Para conectar manualmente a la base de datos deseada, puede pasar un array de valores:

```
$config['hostname'] = "localhost";
$config['username'] = "mi_usuario";
$config['password'] = "mi_contraseña";
$config['database'] = "mi_DB";
$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 mayor información sobre cada uno de estos valores, por favor vea la página de configuración.

O puede presentar los valores de base de datos como un Nombre de Fuente de Datos (DSN). Los DSNs tienen que tener este prototipo:

```
$dsn = 'dbdriver://usuario:contraseña@nombre_host/base_de_datos';

$this->load->database($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://usuario:contraseña@nombre_host/base_de_datos?
char_set=utf8&dbcollat=utf8_general_ci&cache_on=true&cachedir=/ruta/al/cache';

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

Conectarse a Varias Bases de Datos

Si necesita conectar a más de una base de datos simultáneamente, puede hacer lo siguiente:

```
$DB1 = $this->load->database('grupo_uno', TRUE);
$DB2 = $this->load->database('grupo_dos', TRUE);
```

Nota: Cambiar las palabras "grupo_uno" y "grupo_dos" a los nombres de grupo específicos para los que está conectando (o pase los valores de conexión como se indicó antes).

Estableciendo el segundo parámetro a **TRUE** (booleano) la función devolverá el objeto database.

Al conectar de este modo, usará su nombre de objeto para ejecutar comandos en lugar de la sintaxis usada en esta guía. En otras palabras, en lugar de ejecutar comandos con:

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

En su lugar usará:

```
$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();
```

Consultas

\$this->db->query()

Para realizar una consulta, usar la siguiente función:

```
$this->db->query('PONER AQUI LA CONSULTA');
```

La función **query()** devuelve un objeto de resultado de base de datos cuando se ejecutan consultas tipo "leer", las cuales puede usar para mostrar sus resultados. Cuando se ejecutan consultas tipo "escribir" la función simplemente devuelve **TRUE** o **FALSE** dependiendo del éxito o fracaso. Al devolver datos, normalmente asignará la consulta a una variable, así:

```
$query = $this->db->query('PONER AQUI LA CONSULTA');
```

\$this->db->simple_query()

Esta es una versión simplificada de la función **\$this->db->query()**. **SOLAMENTE** devuelve **TRUE/FALSE** en caso de éxito o fracaso. **NO** devuelve un conjunto de resultados de base de datos, ni establece el temporizador de consultas, ni compila datos enlazados, o almacena consultas para depuración. Simplemente le permite realizar una consulta. La mayoría de los usuarios rara vez usan esta función.

Agregar Manualmente Prefijos de Base de Datos

Si configuró un prefijo de base de datos y quisiera agregarlo manualmente, puede usar lo siguiente.

```
$this->db->dbprefix('nombre_de_tabla');  
// imprime: prefijo_nombre_de_tabla
```

Proteger Identificadores

En muchas bases de datos es recomendable proteger las tablas y nombres de campos - por ejemplo con backticks en MySQL. Las consultas del **Active Record** están protegidas automáticamente, sin embargo, si necesita proteger manualmente un identificador, puede usar:

```
$this->db->protect_identifiers('nombre_de_tabla');
```

Esta función también agregará un prefijo de tabla a su tabla asumiendo que tiene un prefijo establecido en su archivo de configuración de la base de datos. Para habilitar el prefijado, establecer a **TRUE** (booleano) el segundo parámetro:

```
$this->db->protect_identifiers('nombre_de_tabla', TRUE);
```

Escapar Consultas

Es una muy buena práctica de seguridad escapar los datos antes de enviarlos a la base de datos. CodeIgniter tiene tres métodos para ayudarle a hacer esto:

1. **\$this->db->escape():** Esta función determina el tipo de datos por lo que solamente puede escapar datos de cadena. También automáticamente agrega comillas simples alrededor de los datos, por lo que Ud no tiene que hacerlo:

```
$sql = "INSERT INTO tabla (titulo) VALUES(\".$this->db->escape($titulo).\")";
```

2. **\$this->db->escape_str():** Esta función escapa los datos pasados a ella, independientemente del tipo. La mayoría de las veces usará la función anterior en lugar de esta. Usar esta función así:

```
$sql = "INSERT INTO tabla (titulo) VALUES('".
    $this->db->escape_str($titulo).')";
```

3. **\$this->db->escape_like_str():** Se debería usar este método cuando las cadenas se usan en condiciones LIKE por lo que los comodines de LIKE ('%', '_') en cadenas también se escapan adecuadamente.

```
$search = '20% raise';
$sql = "SELECT id FROM tabla WHERE column LIKE '%".
    $this->db->escape_like_str($search).%"';
```

Enlazado de Consultas

El enlazado le permiten simplificar la sintaxis de sus consultas, al permitir que el sistema junte las consultas por Ud. Considere el siguiente ejemplo:

```
$sql = "SELECT * FROM alguna_tabla WHERE id = ? AND estado = ? AND autor = ?";
$this->db->query($sql, array(3, 'vivo', 'Ricardo'));
```

Los signos de pregunta en la consulta se reemplazan automáticamente con los valores en el array en el segundo parámetro de la función **query()**.

El beneficio secundario de usar enlazado es que los **valores** se **escapan automáticamente**, produciendo **consultas más seguras**. No tiene que recordar escapar manualmente los datos; el motor lo hace automáticamente por Ud.

Generar Resultados de Consultas

Hay varias formas de generar resultados de consultas:

→ **result()**

Esta función **devuelve un array de objetos**, o un **array vacío** en caso de **falla**. Normalmente, Ud **usará** esta función en un bucle **foreach**, como este:

```
$query = $this->db->query("SU CONSULTA");

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

La función anterior es un alias de **result_object()**.

Si ejecuta consultas que pueden **no** producir un resultado, lo animamos a probar primero el resultado:

```
$query = $this->db->query("SU CONSULTA");

if ($query->num_rows() > 0) PROBAR SI HAY RESULTADOS
{
    foreach ($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; // llamar atributos
    echo $row->reverse_name(); // o métodos definidos en la clase 'User'
}
```

result_array()

Esta función **devuelve** un **resultado** de **consulta** como un **array puro**, o un **array vacío** cuando no se produce resultado. Normalmente, usará esta función dentro de un bucle **foreach**, así:

```
$query = $this->db->query("SU CONSULTA");

foreach ($query->result_array() as $row)
{
```

```

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

```

row()

Esta función devuelve una fila simple de resultado. Si su consulta tiene más de una fila, devolverá solamente la primera. El resultado se devuelve como un objeto. Aquí hay un ejemplo de 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 quiere que se devuelva una fila específica, tiene que enviar el número de 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; // llamar atributos
echo $row->reverse_name(); // o métodos definidos en la clase 'User'

```

row_array()

Idéntica a la función **row()** anterior, salvo que devuelve un array. 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 quiere que se devuelva una fila específica, puede enviar el número de fila como un dígito en el primer parámetro:

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

Además, puede ir hacia adelante/atrás/primera/última en sus resultados, usando estas variaciones:

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

Por defecto, devuelven un objeto, a menos que ponga la palabra "array" en el parámetro:

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

Funciones Helper de Resultados

\$query->num_rows()

Cantidad de filas devueltas por la consulta. **Nota:** En este ejemplo, **\$query** es la variable a la que se le asigna el objeto de resultado de la consulta:

```
$query = $this->db->query('SELECT * FROM mi_tabla');
echo $query->num_rows();
```

\$query->num_fields()

Cantidad de **CAMPOS** (columnas) devueltos por la consulta. Asegurarse de llamar la función usando su objeto de resultado de consulta:

```
$query = $this->db->query('SELECT * FROM mi_tabla');
echo $query->num_fields();
```

\$query->free_result()

Libera la memoria asociada con el resultado y borra el ID del recurso de resultado. Normalmente PHP libera su memoria automáticamente al final de la ejecución del script. Sin embargo, si está ejecutando un montón de consultas en un script en particular, podría querer liberar el resultado después de que cada resultado de consulta se haya generado de forma de reducir el consumo de memoria. Ejemplo:

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

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

```
$query->free_result(); // El objeto de resultado $query no estará más disponible  
$query2 = $this->db->query('SELECT name FROM alguna_tabla');  
$row = $query2->row();  
echo $row->name;  
$query2->free_result(); // El objeto de resultado $query2 no estará más disponible
```

Funciones Helper de Consultas

\$this->db->insert_id()

El número ID de la inserción al ejecutar inserciones en la base de datos.

\$this->db->affected_rows()

Muestra la cantidad de filas afectadas, al hacer una consulta tipo "escribir" (inserción, actualización, etc.).

Nota: En MySQL "DELETE FROM TABLE" devuelve 0 filas afectadas. La clase **Database** tiene un pequeño truco que permite devolver la cantidad correcta de filas afectadas. Por defecto, este truco está habilitado, pero se puede apagar en el archivo del driver de la base de datos.

\$this->db->count_all()

Le permite determinar la cantidad de filas en una tabla en particular. Envíe el nombre de la tabla en el primer parámetro. Ejemplo:

```
echo $this->db->count_all('mi_tabla');  
// Produce un entero, como 25
```

\$this->db->platform()

Imprime la plataforma de base de datos que está ejecutando (MySQL, MS SQL, Postgres, etc...):

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

\$this->db->version()

Imprime la versión de base de datos que está ejecutando:

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

\$this->db->last_query()

Devuelve la última consulta que se ejecutó (la cadena de consulta, no su resultado). Ejemplo:

```
$str = $this->db->last_query();  
// Produce: SELECT * FROM alguna_tabla....
```

Las siguientes dos funciones ayudan a simplificar el proceso de escribir INSERTs y UPDATEs.

\$this->db->insert_string()

Esta función simplifica el proceso de escribir inserciones en la base de datos. Devuelve una cadena de inserción de SQL correctamente formateada. Ejemplo:

```
$data = array('name' => $name, 'email' => $email, 'url' => $url);  
$str = $this->db->insert_string('nombre_de_tabla', $data);
```

El **primer parámetro** es el **nombre de la tabla**, el **segundo** es un **array asociativo** con los datos **a insertarse**. El ejemplo anterior produce:

```
INSERT INTO nombre_de_tabla (name, email, url) VALUES ('Ricardo',  
'ricardo@ejemplo.com', 'ejemplo.com')
```

Nota: Los valores se escapan automáticamente, produciendo consultas más seguras.

\$this->db->update_string()

Esta función **simplifica** el **proceso** de **escribir actualizaciones** de **base de datos**. **Devuelve** un **cadena** de **actualización de SQL** correctamente **formateada**. Ejemplo:

```
$data = array('name' => $name, 'email' => $email, 'url' => $url);  
$where = "autor_id = 1 AND estado = 'activo"';  
$str = $this->db->update_string('nombre_de_tabla', $data, $where);
```

El **primer parámetro** es el **nombre de la tabla**, el **segundo** es un **array asociativo** con los **datos a actualizarse**, y el **tercer parámetro** es la cláusula **"where"**. El ejemplo anterior produce:

```
UPDATE nombre_de_tabla SET name = 'Ricardo', email = 'ricardo@ejemplo.com',  
url = 'ejemplo.com' WHERE autor_id = 1 AND estado = 'activo'
```

Nota: Los valores se escapan automáticamente, produciendo **consultas más seguras**.

La Clase Active Record

CodeIgniter usa una versión modificada del Patrón de Base de Datos **Active Record**. Este patrón permite que la información sea obtenida, insertada y actualizada en la base de datos con mínimo código. En algunos casos solamente son necesarias una o dos líneas de código para ejecutar una acción en la base de datos. CodeIgniter no necesita que cada tabla de base de datos tenga su propio archivo de clase. En lugar de eso, provee una interfaz más simplificada.

Más allá de la simplicidad, el mayor beneficio de usar las funcionalidades del **Active Record** es que le permite crear aplicaciones independientes de las bases de datos, ya que cada adaptador de base de datos genera la sintaxis de la consulta. Esto también permite consultas más seguras, ya que el sistema escapa automáticamente los valores.

Nota: Si tiene intención de escribir sus propias consultas, puede deshabilitar esta clase en su archivo de configuración de la base de datos, permitiéndole al núcleo de la Base de Datos y al adaptador usar pocos recursos.

Seleccionar Datos

Las siguientes funciones le permiten construir sentencias SELECT de SQL.

Nota: Si está usando PHP 5, puede usar el Método de Encadenamiento para obtener una sintaxis más compacta. Esto se describe al final de la página.

\$this->db->get()

Ejecuta la consulta de selección y devuelve el resultado. Puede utilizarse por sí mismo para recuperar todos los registros de una tabla:

```
$query = $this->db->get('mi_tabla');  
// Produce: SELECT * FROM mi_tabla
```

El segundo y tercer parámetros le permiten establecer una cláusula "limit" y "offset": Dentro de un Array Indica la distancia (desplazamiento) desde el inicio del objeto hasta un punto o elemento dado

```
$query = $this->db->get('mi_tabla', [10, 20]);  
// Produce: SELECT * FROM mi_tabla LIMIT 20, 10 (en MySQL. Otras bases de datos  
// pueden tener sintaxis ligeramente diferentes)
```

Advertirá que la función anterior se asigna a una variable llamada **\$query**, la cual se usa para mostrar el resultado:

```
$query = $this->db->get('mi_tabla');  
  
foreach ($query->result() as $row)  
{  
    echo $row->título;  
}
```

Por favor consulte la página funciones de resultado para ver la discusión completa independientemente de la generación del resultado.

\$this->db->get_where()

Idéntica a la función anterior, excepto que le permite agregar una cláusula "where" en el segundo parámetro, en lugar de usar la función **db->where()**:

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

Por favor, leer acerca de la función **where()** abajo para más información.

Nota: **get_where()** era conocida antes como **getwhere()**, la cual fue eliminada.

\$this->db->select()

Le permite escribir la porción **SELECT** de una consulta:

```
$this->db->select('titulo, contenido, $fecha');

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

// Produce: SELECT titulo, contenido, $fecha FROM mi_tabla
```

Nota: Si está seleccionando todo (*) de una tabla no necesita usar esta función. Al omitirse, CodeIgniter asume que desea seleccionar todo (SELECT *)

\$this->db->select() acepta un segundo parámetro opcional. Si lo establecer como **FALSE**, CodeIgniter no intentará proteger sus nombres de campo o tabla con backticks. Esto es útil si necesita una sentencia compuesta de selección.

```
$this->db->select('(SELECT SUM(pagos.cantidad) FROM pagos
    WHERE pagos.factura_id=4') AS cantidad_pagada', FALSE);
$query = $this->db->get('mi_tabla');
```

\$this->db->select_max()

Escribe una porción "SELECT MAX(campo)" en su consulta. Opcionalmente puede incluir un segundo parámetro para renombrar el campo de resultado.

```
$this->db->select_max('edad');
$query = $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()

Escribe una porción "SELECT MIN(campo)" en su consulta. Como con **select_max()**, opcionalmente puede incluir un segundo parámetro para renombrar el campo de resultado.

```
$this->db->select_min('edad');
$query = $this->db->get('miembros');
// Produce: SELECT MIN(edad) AS edad FROM miembros
```

\$this->db->select_avg()

Escribe una porción "SELECT AVG(campo)" en su consulta. Como con **select_max()**, opcionalmente puede incluir un segundo parámetro para renombrar el campo de resultado.

```
$this->db->select_avg('edad');
$query = $this->db->get('miembros');
// Produce: SELECT AVG(edad) AS edad FROM miembros
```

\$this->db->select_sum()

Escribe una porción "SELECT SUM(campo)" en su consulta. Como con **select_max()**, opcionalmente puede incluir un segundo parámetro para renombrar el campo de resultado.

```
$this->db->select_sum('edad');
$query = $this->db->get('miembros');
// Produce: SELECT SUM(edad) AS edad FROM miembros
```

\$this->db->from()

Le permite escribir la porción FROM de su consulta:

```
$this->db->select('titulo, contenido, $fecha');
$this->db->from('mi_tabla');

$query = $this->db->get();

// Produce: SELECT titulo, contenido, $fecha FROM mi_tabla
```

Nota: Como se mostró antes, la porción FROM de su consulta se puede especificar en la función **\$this->db->get()**, por lo que puede usar el método que prefiera.

\$this->db->join()

Le permite escribir la porción JOIN de su 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
```

Se pueden hacer varias llamadas de función si necesita varios joins en una consulta.

Si necesita un tipo específico de JOIN puede especificarlo mediante el tercer parámetro de la función. Las opciones son: left, right, outer, inner, left outer, y right outer.

```
$this->db->join('comentarios', 'comentarios.id = blogs.id', 'left');
// Produce: LEFT JOIN comentarios ON comentarios.id = blogs.id
```

\$this->db->where()

Esta función le permite establecer cláusulas **WHERE** usando uno de los estos cuatro métodos:

1. Método simple de clave/valor:

```
$this->db->where('nombre', $nombre);
// Produce: WHERE nombre = 'Jose'
```

Advierta que se agrega el signo igual por Ud.

Si usa varias llamadas de función, se encadenarán todas juntas con **AND** entre ellas:

```
$this->db->where('nombre', $nombre);
$this->db->where('titulo', $titulo);
$this->db->where('estado', $status);

// WHERE nombre = 'Jose' AND titulo = 'jefe' AND estado = 'activo'
```

2. Método personalizado de clave/valor:

Puede incluir un operador en el primer parámetro para controlar la comparación:

```
$this->db->where('nombre !=', $nombre);
$this->db->where('id <', $id);

// Produce: WHERE nombre != 'Jose' AND id < 45
```

3. Método del array asociativo:

```
$array = array('nombre' => $nombre, 'titulo' => $titulo, 'estado' =>
$status);

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

// Produce: WHERE nombre = 'Jose' AND titulo = 'jefe' AND estado = 'activo'
```

También puede incluir sus propios operadores usando este método:

```
$array = array('nombre !=' => $nombre, 'id <' => $id, '$fecha >' => $fecha);

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

4. Cadena personalizada:

Puede escribir sus propias cláusulas manualmente:

```
$where = "nombre='Jose' AND estado='jefe' OR estado='activo'";
$this->db->where($where);
```

\$this->db->where() acepta un tercer parámetro opcional. Si se lo establece a **FALSE**, CodeIgniter no intentará proteger sus nombres de campos o tabla con backticks.

```
$this->db->where('MATCH (campo) AGAINST ("valor")', NULL, FALSE);
```

\$this->db->or_where()

Esta función es idéntica a la anterior, excepto que las instancias múltiples se unen con OR:

```
$this->db->where('nombre !=', $nombre);
$this->db->or_where('id >', $id);

// Produce: WHERE nombre != 'Jose' OR id > 50
```

Nota: **or_where()** antes era conocida como **orwhere()**, la que fue eliminada.

\$this->db->where_in()

Genera una consulta SQL WHERE campo IN ('item', 'item') unida mediante AND si corresponde.

```
$nombres = array('Federico', 'Tomas', 'Juan');
$this->db->where_in('usuario', $nombres);
// Produce: WHERE usuario IN ('Federico', 'Tomas', 'Juan')
```

\$this->db->or_where_in()

Genera una consulta SQL WHERE campo IN ('item', 'item') unida mediante OR si corresponde.

```
$nombres = array('Federico', 'Tomas', 'Juan');
$this->db->or_where_in('usuario', $nombres);
// Produce: OR usuario IN ('Federico', 'Tomas', 'Juan')
```

\$this->db->where_not_in()

Genera una consulta SQL WHERE campo NOT IN ('item', 'item') unida mediante AND si corresponde.

```
$nombres = array('Federico', 'Tomas', 'Juan');
$this->db->where_not_in('usuario', $nombres);
// Produce: WHERE usuario NOT IN ('Federico', 'Tomas', 'Juan')
```

\$this->db->or_where_not_in()

Genera una consulta SQL WHERE campo NOT IN ('item', 'item') unida mediante OR si corresponde.

```
$nombres = array('Federico', 'Tomas', 'Juan');
$this->db->or_where_not_in('usuario', $nombres);
// Produce: OR usuario NOT IN ('Federico', 'Tomas', 'Juan')
```

\$this->db->like()

Esta función le permite generar cláusulas **LIKE**, útiles para hacer búsquedas.

Nota: Todos los valores pasados a esta función se escapan automáticamente.

1. Método simple de clave/valor:

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

// Produce: WHERE titulo LIKE '%match%'
```

Si usa varias llamadas a la función, se encadenarán juntas con **AND** entre ellas:

```
$this->db->like('titulo', 'match');
$this->db->like('cuerpo', 'match');

// WHERE titulo LIKE '%match%' AND cuerpo LIKE '%match%'
```

Si quiere controlar donde se ubica el comodín (%), puede usar un tercer parámetro opcional. Las opciones son 'before', 'after' y 'both' (que es el valor por defecto).

```
$this->db->like('titulo', 'match', 'before');
// Produce: WHERE titulo LIKE '%match'

$this->db->like('titulo', 'match', 'after');
// Produce: WHERE titulo LIKE 'match%'

$this->db->like('titulo', 'match', 'both');
// Produce: WHERE titulo LIKE '%match%'
```

2. Método del array 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()

Esta función es idéntica a la anterior, excepto que las instancias múltiples se unen mediante OR:

```
$this->db->like('titulo', 'match');
$this->db->or_like('cuerpo', $match);

// WHERE titulo LIKE '%match%' OR cuerpo LIKE '%match%'
```

Nota: **or_like()** antes era conocida como **orlike()**, la cual fue eliminada.

\$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 las instancias múltiples se unen mediante OR:

```
$this->db->like('titulo', 'match');
$this->db->or_not_like('cuerpo', 'match');

// WHERE titulo LIKE '%match%' OR cuerpo NOT LIKE '%match%'
```

\$this->db->group_by()

Le permite escribir la porción GROUP BY de su consulta:

```
$this->db->group_by("titulo");

// Produce: GROUP BY titulo
```

También puede pasarle un array de múltiples valores:

```
$this->db->group_by(array("titulo", "$fecha"));

// Produce: GROUP BY titulo, $fecha
```

Nota: **group_by()** antes era conocida como **groupby()**, la cual fue eliminada.

\$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()

Le permite escribir la porción HAVING de su consulta. Hay dos sintaxis posibles, uno o dos argumentos:

```
$this->db->having('usuario_id = 45');
// Produce: HAVING usuario_id = 45

$this->db->having('usuario_id', 45);
// Produce: HAVING usuario_id = 45
```

También puede pasarle un array de múltiples valores:

```
$this->db->having(array('titulo => 'Mi Titulo', 'id < => $id'));

// Produce: HAVING titulo = 'Mi Titulo', id < 45
```

Si está usando una base de datos para la que CodeIgniter escapa las consultas, puede evitar de escapar el contenido pasando un tercer parámetro opcional, y estableciéndolo a **FALSE**.

```
$this->db->having('usuario_id', 45);
// Produce: HAVING `usuario_id` = 45 en algunas bases de datos como MySQL

$this->db->having('user_id', 45, FALSE);
// Produce: HAVING user_id = 45
```

\$this->db->or_having()

Idéntica a **having()**, salvo que separa varias cláusulas mediante "OR".

\$this->db->order_by()

Le permite establecer una cláusula ORDER BY. El primer parámetro contiene el nombre de la columna por la que querría ordenar. El segundo parámetro le permite establecer la dirección del resultado. Las opciones son **asc**, **desc**, o **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 se pueden hacer varias llamadas a la función si necesita varios campos.

```
$this->db->order_by("titulo", "desc");  
$this->db->order_by("nombre", "asc");  
  
// Produce: ORDER BY titulo DESC, nombre ASC
```

Nota: **order_by()** antes conocida como **orderby()**, la cual fue eliminada.

Nota: el ordenamiento aleatorio no es actualmente soportado por los adaptadores de Oracle o MSSQL. Estos tendrán por defecto a 'ASC'.

\$this->db->limit()

Le permite limitar la cantidad de filas que desea que la consulta devuelva:

```
$this->db->limit(10);  
// Produce: LIMIT 10
```

El segundo parámetro le permite establecer un desplazamiento del resultado.

```
$this->db->limit(10, 20);  
  
// Produce: LIMIT 20, 10 (en MySQL. Otras bases de datos pueden tener una sintaxis  
// ligeramente diferente)
```

\$this->db->count_all_results()

Le permite determinar la cantidad de filas en una consulta del **Active Record**. Las consultas aceptarán restrictores 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()

Le permite determinar la cantidad de filas en una tabla en particular. Presente el nombre de la tabla como primer parámetro. Ejemplo:

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

// Produce un entero, como 25
```

Insertar Datos

\$this->db->insert()

Genera una **cadena insert** de SQL basada en los **datos que suministra**, y **ejecuta la consulta**. Tanto puede **pasar un array** como un **objeto** a la **función**. Aquí hay un ejemplo usando un array:

```
$data = array(
    'titulo' => 'Mi titulo' ,
    'nombre' => 'Mi nombre' ,
    '$fecha' => 'Mi $fecha'
);

$this->db->insert('mi_tabla', $data);

// Produce: INSERT INTO mi_tabla (titulo, nombre, $fecha) VALUES ('Mi titulo',
// 'Mi nombre', 'Mi $fecha')
```

El primer parámetro contendrá el nombre de la tabla y el segundo un array asociativo de valores.

Aquí hay un ejemplo **usando un objeto**:

```
/*
class MiClase {
    var $titulo = 'Mi Titulo';
    var $contenido = 'Mi Contenido';
    var $$fecha = 'Mi $fecha';
}
*/
$object = new MiClase;

$this->db->insert('mi_tabla', $object);

// Produce: INSERT INTO mi_tabla (titulo, contenido, $fecha) VALUES ('Mi Titulo',
// 'Mi Contenido', 'Mi $fecha')
```

El **primer parámetro contendrá el nombre de la tabla** y el **segundo es un objeto**.

Nota: Todos los valores se escapan automáticamente para producir 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(
        'titulo' => 'Mi titulo' ,
        'nombre' => 'Mi nombre' ,
        'fecha' => 'Mi fecha'
    ),
    array(
        'titulo' => 'Otro titulo' ,
        'nombre' => 'Otro nombre' ,
        'fecha' => 'Otra fecha'
    )
);

$this->db->insert_batch('mi_tabla', $data);

// Produce: 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 le `permite establecer` `valores` para `inserciones` o `actualizaciones`.

Se puede usar en lugar de pasar datos directamente a un array para las funciones de inserción o actualización:

```
$this->db->set('nombre', $nombre);
$this->db->insert('mi_tabla');

// Produce: INSERT INTO mi_tabla (name) VALUES ('{$nombre}')
```

Si usa varias llamadas de función, se ensamblarán adecuadamente en función de si usted `está haciendo una inserción o una actualización`:

```
$this->db->set('nombre', $nombre);
$this->db->set('titulo', $titulo);
$this->db->set('estado', $status);
$this->db->insert('mi_tabla');
```

`set()` también aceptará un tercer parámetro opcional (`$escape`), que evitará que los datos se escapen si se lo establece a `FALSE`. Para ilustrar la diferencia, aquí se usa `set()` con y sin el parámetro de escape.

```
$this->db->set('campo', 'campo+1', FALSE);
$this->db->insert('mi_tabla');
// Devuelve: INSERT INTO mi_tabla (campo) VALUES ('campo+1')

$this->db->set('campo', 'campo+1');
$this->db->insert('mi_tabla');
// Devuelve: INSERT INTO mi_tabla (campo) VALUES ('campo+1')
```

También puede pasar un array asociativo a esta función:

```
$array = array('nombre' => $nombre, 'titulo' => $titulo, 'estado' => $status);

$this->db->set($array);
$this->db->insert('mi_tabla');
```

O un objeto:

```
/*
  class Mi_clase {
    var $titulo = 'Mi titulo';
    var $contenido = 'Mi contenido';
    var $fecha = 'Mi fecha';
  }
*/

$object = new Mi_clase;

$this->db->set($object);
$this->db->insert('mi_tabla');
```

Actualizar Datos

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

Genera una cadena update de SQL y ejecuta la consulta basada en los datos provistos. A la función puede pasarle un `array` o un `objeto`. Aquí hay un ejemplo usando un array:

```
$data = array(
  'titulo' => $titulo,
  'nombre' => $nombre,
  'fecha' => $fecha
);

$this->db->where('id', $id);
$this->db->update('mi_tabla', $data);

// Produce:
// UPDATE mi_tabla
// SET title = '{$titulo}', name = '{$nombre}', date = '{$fecha}'
// WHERE id = $id
```

O puede proporcionar un objeto:

```
/*
class Mi_clase {
    var $titulo = 'Mi titulo';
    var $nombre = 'Mi nombre';
    var $fecha = 'Mi fecha';
}
*/
$object = new Mi_clase;

$this->db->where('id', $id);
$this->db->update('mi_tabla', $object);

// Produce:
// UPDATE mi_tabla
// SET title = '{$titulo}', name = '{$nombre}', date = '{$fecha}'
// WHERE id = $id
```

Nota: Todos los valores pasados a esta función se escapan, produciendo consultas más seguras.

Advertirá el uso de la función **\$this->db->where()**, permitiéndole establecer la cláusula WHERE. Opcionalmente puede pasar esta información directamente a la función **update()** como una cadena:

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

O como un array:

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

También puede usar la función **\$this->db->set()** descripta antes cuando se realicen actualizaciones.

Borrar Datos

\$this->db->delete()

Genera una cadena delete de SQL y ejecuta la consulta.

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

// Produce:
// DELETE FROM mi_tabla
// WHERE id = $id
```

El primer parámetro es el nombre de la tabla y el segundo la cláusula where. También puede usar las funciones **where()** u **or_where()** en lugar de pasarle los datos al segundo parámetro de la función:

```
$this->db->where('id', $id);
$this->db->delete('mi_tabla');

// Produce:
// DELETE FROM mi_tabla
// WHERE id = $id
```

Si quiere borrar más de una tabla, se puede pasar un array de nombres de tablas a **delete()**.

```
$tables = array('table1', 'table2', 'table3');
$this->db->where('id', '5');
$this->db->delete($tables);
```

Si quiere borrar todos los datos de una tabla, puede usar las funciones **truncate()** o **empty_table()**.

\$this->db->empty_table()

Genera una cadena delete de SQL y ejecuta la consulta.

```
$this->db->empty_table('mi_tabla');

// Produce
// DELETE FROM mi_tabla
```

\$this->db->truncate()

Genera una cadena truncate de SQL y ejecuta la consulta.

```
$this->db->from('mi_tabla');
$this->db->truncate();
// o
$this->db->truncate('mi_tabla');

// Produce:
// TRUNCATE mi_tabla
```

Nota: Si el comando TRUNCATE no está disponible, **truncate()** se ejecutará como "DELETE FROM table".

Método de Encadenamiento

El Método de Encadenamiento le permite simplificar la sintaxis conectando varias funciones. Considere este ejemplo:

```
$this->db->select('titulo')->from('mi_tabla')->where('id', $id)->limit(10, 20);

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

Nota: El método de encadenamiento solamente funciona con PHP 5.

Almacenamiento en Caché del Active Record

Si bien no es un caché "verdadero", el **Active Record** le permite guardar (o "cachear") ciertas partes de sus consultas para reusarlas más adelante en la ejecución de su script. Normalmente, cuando se completa una llamada del **Active Record**, toda la información almacenada se borra para la siguiente llamada. Con el almacenamiento en caché, puede evitar este borrado, y reusar la información fácilmente.

Las llamadas cacheadas son acumulativas. Si hace dos llamadas **select()** cacheadas, y luego dos llamadas **select()** sin cachear, esto resultará en cuatro llamadas **select()**. Hay tres funciones de Caché disponibles:

\$this->db->start_cache()

Se tiene que llamar a esta función para comenzar a cachear. Todas las consultas del **Active Record** de tipo correcto (ver más abajo para conocer las consultas soportadas) se almacenan para uso posterior.

\$this->db->stop_cache()

Se puede llamar a esta función para detener el caché.

\$this->db->flush_cache()

Esta función borra todos los elementos del caché del **Active Record**.

Aquí hay un ejemplo de uso:

```
$this->db->start_cache();
$this->db->select('campo1');
$this->db->stop_cache();

$this->db->get('nombre_de_tabla');

//Genera: SELECT `campo1` FROM (`nombre_de_tabla`)

$this->db->select('campo2');
$this->db->get('nombre_de_tabla');

//Genera: SELECT `campo1`, `campo2` FROM (`nombre_de_tabla`)

$this->db->flush_cache();

$this->db->select('campo2');
$this->db->get('nombre_de_tabla');

//Genera: SELECT `campo2` FROM (`nombre_de_tabla`)
```

Nota: Se pueden cachear las siguientes sentencias: select, from, join, where, like, group_by, having, order_by, set.

Transacciones

La abstracción de base de datos de CodeIgniter le permite usar **transacciones** con bases de datos que soporten tipos de tablas seguras en transacciones. In MySQL, necesitará ejecutar tablas de tipo InnoDB o BDB, en lugar del más común MyISAM. La mayoría de las otras plataformas de bases de datos soportan transacciones nativamente.

Si no está familiarizado con las transacciones, le recomendamos que busque en línea algún buen recurso para aprender acerca de ellas para su base de datos en particular. La información siguiente asume que Ud tiene un conocimiento básico de transacciones.

El Enfoque de CodeIgniter para las Transacciones

CodeIgniter utiliza un enfoque para transacciones que es muy similar al proceso usado por la popular clase de base de datos ADODB. Elegimos ese enfoque porque simplifica enormemente el proceso de ejecutar transacciones. En la mayoría de los casos se necesitan dos líneas de código.

Tradicionalmente, las transacciones han requerido una buena cantidad de trabajo para implementar, ya que le demandan hacer el seguimiento de sus consultas y determinar si **hacer** o **deshacer** las consultas basado en su éxito o fracaso. Esto es particularmente engorroso con consultas anidadas. Por el contrario, hemos implementado un sistema de transacciones inteligente que hace todo esto automáticamente (puede administrar sus transacciones manualmente si así lo desea, pero no hay un beneficio realmente).

Ejecutar Transacciones

Para ejecutar consultas usando transacciones, se usarán las funciones **\$this->db->trans_start()** y **\$this->db->trans_complete()** del siguiente modo:

```
$this->db->trans_start();
$this->db->query('UNA CONSULTA SQL...');
$this->db->query('OTRA CONSULTA SQL...');
$this->db->query('Y OTRA CONSULTA SQL MAS...');
$this->db->trans_complete();
```

Puede ejecutar tantas consultas como quiera entre las funciones de "start"/"complete" y se harán o desharán según resulten exitosas o fracasen para cualquier consulta dada.

Modo Estricto

Por defecto, CodeIgniter ejecuta todas las transacciones en **Modo Estricto**. Cuando el modo estricto está habilitado y se ejecutan varios grupos de transacciones, si un grupo falla todos los grupos se desharán. Si el modo estricto está deshabilitado, cada grupo es tratado independientemente, lo que significa que la falla de un grupo no afectará a los otros.

El Modo Estricto se puede deshabilitar de la siguiente manera:

```
$this->db->trans_strict(FALSE);
```

Administrar Errores

Si tiene el reporte de errores habilitado en su archivo **config/database.php** verá un mensaje de error estándar si la ejecución fue un fracaso. Si la depuración está desconectada puede administrar sus propios errores así:

```
$this->db->trans_start();
$this->db->query('UNA CONSULTA SQL...');
$this->db->query('OTRA CONSULTA SQL...');

if ($this->db->trans_status() === FALSE)
{
    // genera un error... o usa la función log_message() para registrar su error
}
```

Habilitar Transacciones

Las transacciones se habilitan automáticamente en el momento que se usa **\$this->db->trans_start()**. Si quisiera deshabilitar las transacciones, podría hacerlo usando **\$this->db->trans_off()**:

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

$this->db->trans_start();
$this->db->query('UNA CONSULTA SQL...');
$this->db->trans_complete();
```

Cuando se deshabilitan las transacciones, sus consultas se ejecutarán automáticamente, tal y como son cuando se ejecutan sin transacciones.

Modo de Prueba

Opcionalmente puede poner el sistema de transacciones en "modo prueba", lo que causará que sus consultas se deshagan -- aún cuando produzcan un resultado válido. Para usar el modo de prueba, simplemente establecer el primer parámetro en la función **\$this->db->trans_start()** a **TRUE**:

```
$this->db->trans_start(TRUE); // Se deshará la consulta
$this->db->query('UNA CONSULTA SQL...');
$this->db->trans_complete();
```

Ejecutar Transacciones Manualmente

Si quisiera ejecutar transacciones manualmente, debería hacer lo siguiente:

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

$this->db->query('UNA CONSULTA SQL...');
$this->db->query('OTRA CONSULTA SQL...');
$this->db->query('Y OTRA CONSULTA SQL MAS...');

if ($this->db->trans_status() === FALSE)
{
    $this->db->trans_rollback();
}
else
{
    $this->db->trans_commit();
}
```

Nota: Asegurarse de usar **\$this->db->trans_begin()** al ejecutar transacciones manuales, y **NO \$this->db->trans_start()**.

Metadatos de Tabla

Estas funciones le permiten obtener información acerca de la tabla.

\$this->db->list_tables()

Devuelve un array conteniendo los nombres de todas las tablas en la base de datos a la que está conectado actualmente. Ejemplo:

```
$tables = $this->db->list_tables();  
  
foreach ($tables as $table)  
{  
    echo $table;  
}
```

\$this->db->table_exists()

A veces es útil saber si existe una tabla en particular antes de ejecutar una operación sobre ella. Devuelve un booleano TRUE/FALSE. Ejemplo de uso:

```
if ($this->db->table_exists('nombre_de_tabla'))  
{  
    // algún código...  
}
```

Nota: Reemplace *nombre_de_tabla* con el nombre de la tabla por el que está buscando.

Metadatos de Campo

\$this->db->list_fields()

Devuelve un array conteniendo los nombres de campos. Se puede llamar a esta consulta de dos formas:

1. Puede suministrar el nombre de la tabla y llamarla desde el objeto \$this->db->:

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

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

2. Puede juntar los nombres de campos asociados con cualquier consulta que ejecute llamando a la función desde el objeto resultado de la consulta:

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

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

\$this->db->field_exists()

A veces es útil saber si existe un campo en particular antes de ejecutar una acción. Devuelve un booleano TRUE/FALSE. Ejemplo de uso:

```
if ($this->db->field_exists('nombre_de_campo', 'nombre_de_tabla'))
{
    // algún código...
}
```

Nota: Reemplace *nombre_de_campo* con el nombre de la columna por la que desea preguntar, y reemplace *nombre_de_tabla* con el nombre de la tabla por la que está buscando.

\$this->db->field_data()

Devuelve un array de objetos conteniendo información de los campos.

A veces es útil obtener los nombres de los campos u otro metadato, como el tipo de columna, longitud máxima, etc.

Nota: No todas las bases de datos proveen metadatos.

Ejemplo de Uso:

```
$fields = $this->db->field_data('nombre_de_tabla');

foreach ($fields as $field)
{
    echo $field->name;
    echo $field->type;
    echo $field->max_length;
    echo $field->primary_key;
}
```

Si ya ejecutó una consulta, puede usar el objeto de resultado en lugar de suministrar el nombre de la tabla:

```
$query = $this->db->query("SU CONSULTA");
$fields = $query->field_data();
```

Los siguientes datos están disponibles de esta función si los soporta su base de datos:

- **name** - nombre de la columna
- **max_length** - longitud máxima de la columna
- **primary_key** - 1 si la columna es clave primaria
- **type** - tipo de la columna

Llamadas a Funciones Personalizadas

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

Esta función le permite llamar a funciones de base de datos de PHP que no están nativamente incluidas en CodeIgniter, de forma independiente de la plataforma. Por ejemplo, digamos que desea llamar a la función `mysql_get_client_info()`, que no está nativamente soportada por CodeIgniter. Podría hacer algo como esto:

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

Tiene que suministrar el nombre de la función, sin el prefijo `mysql_` en el primer parámetro. El prefijo se agrega automáticamente basado en el adaptador de base de datos que se está utilizando actualmente. Esto le permite ejecutar la misma función en diferentes plataformas de base de datos. Obviamente no todas las llamadas a función son idénticas entre las plataformas, por lo que hay limitaciones respecto a la utilidad de esta función en términos de portabilidad.

Cualquier parámetro necesario para la función será agregado a partir del segundo argumento.

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

A veces necesitará suministrar un ID de conexión de base de datos o un ID de resultado de base de datos. El ID de conexión se puede acceder usando:

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

El ID de resultado se puede acceder desde dentro de su objeto de resultado así:

```
$query = $this->db->query("ALGUNA CONSULTA");  
$query->result_id;
```

Almacenamiento en Caché de Bases de Datos

La Clase para Almacenamiento en Caché de Base de Datos le permite cachear sus consultas como archivos de texto para reducir la carga de la base de datos.

Importante: El adaptador de base de datos inicializa automáticamente esta clase cuando el caché está habilitado. **NO** cargar manualmente esta clase.

Advierta también: No todas las funciones de resultado de consultas están disponibles cuando se usa el caché. Lea esta página con cuidado.

Habilitar el Caché

El almacenamiento en caché se habilita en tres pasos:

- Crear un directorio escribible en el servidor donde se puedan almacenar los archivos de caché.
- Establecer la ruta a la carpeta de caché en el archivo **application/config/database.php**.
- Habilitar la función de caché, ya sea globalmente mediante el establecimiento de la preferencia en el archivo **application/config/database.php**, o manualmente como se describe más abajo.

Una vez habilitado, el almacenamiento en caché ocurrirá automáticamente cuando se cargue una página que contenga consultas de base de datos.

¿Cómo Funciona el Caché?

El sistema de caché de consultas de CodeIgniter funciona dinámicamente cuando se visitan las páginas. Cuando el caché está habilitado, la primera vez que se carga una página web, el objeto de resultado de consultas se serializa y almacena en un archivo de texto en el servidor. La próxima vez que se carga la página, se usará el archivo de caché en lugar de acceder a su base de datos. El uso de su base de datos se puede reducir a cero en forma efectiva para cualquier página que se haya cacheado.

Se pueden cachear solamente consultas de **tipo leer** (SELECT), ya que solamente ese tipo de consultas producen resultado. El sistema no cacheará las consultas de **tipo escribir** (INSERT, UPDATE, etc.), ya que no generan un resultado.

Los archivos de caché **NO** expiran. Cualquier consulta que se haya cacheado permanecerá hasta que se la borre. El sistema de caché le permite borrar cualquier página individual asociada, o puede borrar la colección entera de archivos de caché. Normalmente querrá usar las funciones de limpieza descriptas debajo para borrar los archivos de caché después de que ciertos eventos hayan ocurrido, como cuando se agrega nueva información a la base de datos.

¿Mejorará el Caché el Desempeño del Sitio?

Obtener una ganancia de rendimiento como consecuencia del almacenamiento en caché depende de muchos factores. Si tiene una base de datos altamente optimizada bajo poca carga, probablemente no vea un aumento en el rendimiento. Si la base de datos está bajo uso pesado, probablemente verá una respuesta mejorada, asumiendo que el sistema de archivos no esté sobrecargado. Recuerde que el almacenamiento en caché simplemente cambia la forma en que se recupera su información, pasando de ser una operación de base de datos a una operación del sistema de archivos.

En algunos entornos de servidores en clúster por ejemplo, el almacenamiento en caché puede ser perjudicial, ya que las operaciones del sistema de archivos son demasiado intensas. En servidores simple de entornos compartidos, el almacenamiento en caché probablemente sea benéfico. Desafortunadamente no hay una respuesta simple a la pregunta de si se debería cachear la base de datos. Realmente depende de la situación.

¿Cómo se Almacenan los Archivos de Caché?

CodeIgniter ubica el resultado de CADA consulta en su propio archivo de caché. Los conjuntos de archivos en caché se organizan en subcarpetas que corresponden a las funciones controlador. Para ser precisos, las subcarpetas se llaman del mismo modo que los primeros dos segmentos de la URI (nombre de la clase del controlador y nombre de función).

Por ejemplo, digamos que tiene un controlador llamado **blog** con una función llamada **comentarios** que contiene tres consultas. El sistema de caché creará una carpeta de caché llamada **blog+comentarios**, dentro de la que escribirá tres archivos de caché.

Si usa consultas dinámicas que cambian basadas en la información en la URI (por ejemplo, al usar paginación), cada instancia de la consulta producirá su propio archivo de caché. Es posible, por lo tanto, terminar con muchos más archivos en caché que los que tiene de consultas.

Administrar los Archivos de Caché

Como los archivos en caché no expiran, necesitará armar rutinas de borrado para su aplicación. Por ejemplo, digamos que tiene un blog que permite comentarios de usuarios. Si se envía un nuevo comentario, querrá borrar los archivos cacheados asociados con la función controlador que sirve sus comentarios. Encontrará dos funciones que lo ayudan a borrar los datos, y que de describen más abajo.

No Todas las Funciones de Bases de Datos Funcionan con Caché

Finalmente debemos señalar que el objeto de resultado que se cachea es una versión simplificada del objeto completo. Por esta razón, algunas de las funciones de resultado de la consulta no están disponibles para usarse.

Las siguientes funciones **NO ESTAN** disponibles al usar un objeto de resultados cacheado:

- **num_fields()**
- **field_names()**
- **field_data()**
- **free_result()**

También, dos recursos de base de datos (`result_id` y `conn_id`) no están disponibles al cacheear, ya que los recursos de resultado solamente refieren a operaciones en tiempo de ejecución.

Referencia de Funciones

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

Habilita/deshabilita manualmente el caché. Esto puede ser útil si quiere impedir que ciertas consultas sean cacheadas. Ejemplo:

```
// Habilitar el caché
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM mi_tabla");

// Deshabilitar el caché solo para esta consulta
$this->db->cache_off();
$query = $this->db->query("SELECT * FROM miembros WHERE miembro_id = '$usuario_actual'");

// Volver a habilitar el caché
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM otra_tabla");
```

\$this->db->cache_delete()

Borra los archivos cacheados asociados con una página en particular. Esto es útil si necesita borrar archivos cacheados después de actualizar la base de datos.

El sistema de caché guarda sus archivos de caché en carpetas que corresponden a la URI de la página que está viendo. Por ejemplo, si está viendo una página en **example.com/index.php/blog/comentarios**, el sistema de caché pondrá todos los archivos de caché asociados con ella en una carpeta llamada **blog+comentarios**. Para borrar esos archivos de caché en particular se usará:

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

Si no usa los parámetros, la URI actual se usará para determinar lo que se debería borrar.

\$this->db->cache_delete_all()

Borra todos los archivos de caché existentes. Ejemplo:

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

Clase Forge de Base de Datos

La Clase **Forge** de Base de Datos contiene funciones que lo ayudan a administrar su base de datos.

Inicializar la Clase Forge

Importante: Para inicializar la clase **Forge**, su adaptador de base de datos tiene que estar ejecutándose, ya que la clase **Forge** se basa en él.

Cargar la Clase **Forge** así:

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

Una vez inicializada, accederá a las funciones usando el objeto **\$this->dbforge**:

```
$this->dbforge->alguna_funcion();
```

\$this->dbforge->create_database('db_nombre')

Le permite crear la base de datos especificada en el primer parámetro. Devuelve **TRUE/FALSE** dependiendo del éxito o fracaso:

```
if ($this->dbforge->create_database('mi_db'))  
{  
    echo 'Base de Datos creada!';  
}
```

\$this->dbforge->drop_database('db_nombre')

Le permite eliminar la base de datos especificada en el primer parámetro. Devuelve **TRUE/FALSE** dependiendo del éxito o fracaso:

```
if ($this->dbforge->drop_database('mi_db'))  
{  
    echo 'Base de Datos eliminada!';  
}
```

Crear y Eliminar Tablas

Hay varias cosas que puede querer hacer al crear tablas. Agregar campos o claves a la tabla, modificar columnas. CodeIgniter provee un mecanismo para esto.

Agregar Campos

Los campos se crean mediante un array asociativo. Dentro del array tiene que incluirse una clave 'type' que refiere al tipo de dato del campo. Por ejemplo, INT, VARCHAR, TEXT, etc. Muchos tipos de datos (por ejemplo VARCHAR) también necesitan la clave 'constraint'.

```
$campos = array(
    'usuarios' => array(
        'type' => 'VARCHAR',
        'constraint' => '100',
    ),
);

// Traducirá a "usuarios VARCHAR(100)" cuando se agregue el campo.
```

Además se pueden usar los siguientes pares clave/valor:

- **unsigned/true**: para generar "UNSIGNED" en la definición del campo.
- **default/valor**: para generar un valor por defecto en la definición del campo.
- **null/true**: para generar "NULL" en la definición del campo. Sin esto, el campo tendrá por defecto a "NOT NULL".
- **auto_increment/true**: generar un modificador auto_increment en el campo. Advierta que el tipo de campo tiene que ser un tipo lo soporte, tal como un entero.

```
$campos = array(
    'blog_id' => array(
        'type' => 'INT',
        'constraint' => 5,
        'unsigned' => TRUE,
        'auto_increment' => TRUE
    ),
    'blog_titulo' => array(
        'type' => 'VARCHAR',
        'constraint' => '100',
    ),
    'blog_autor' => array(
        'type' => 'VARCHAR',
        'constraint' => '100',
        'default' => 'Batman',
    ),
    'blog_descripcion' => array(
        'type' => 'TEXT',
        'null' => TRUE,
    ),
);
)
```

Después que los campos hayan sido definidos, se pueden agregar usando **\$this->dbforge->add_field(\$campos)** seguida por una llamada a la función **create_table()**.

\$this->dbforge->add_field()

La función **add_field()** aceptará el array anterior.

Pasar cadenas como campos

Si conoce exactamente cómo desea que se cree un campo, puede pasar la cadena en la definición del campo con **add_field()**.

```
$this->dbforge->add_field("label varchar(100) NOT NULL DEFAULT 'default label'");
```

Nota: Varias llamadas a **add_field()** son acumulativas.

Crear un campo id

Hay una excepción especial para crear campos id. Un campo con tipo id se asignará automáticamente a una Clave Primaria INT(9) con autoincremento.

```
$this->dbforge->add_field('id');  
// Devuelve: id INT(9) NOT NULL AUTO_INCREMENT
```

Agregar Claves

Generalmente hablando, Ud querrá que su tabla tenga claves. Esto se logra con **\$this->dbforge->add_key('campo')**. Un segundo parámetro opcional establecido a **TRUE** la convertirá en clave primaria. Advierta que **add_key()** tiene que estar seguida por una llamada a **create_table()**.

Si hay varias claves no primarias, se tienen que enviar como un array. El ejemplo impreso debajo es para MySQL.

```
$this->dbforge->add_key('blog_id', TRUE);  
// Devuelve: PRIMARY KEY `blog_id` (`blog_id`)  
  
$this->dbforge->add_key('blog_id', TRUE);  
$this->dbforge->add_key('sitio_id', TRUE);  
// Devuelve: PRIMARY KEY `blog_id_sitio_id`(`blog_id`, `sitio_id`)  
  
$this->dbforge->add_key('blog_nombre');  
// Devuelve: KEY `blog_nombre`(`blog_nombre`)  
  
$this->dbforge->add_key(array('blog_nombre', 'blog_rotulo'));  
// Devuelve: KEY `blog_nombre_blog_rotulo`(`blog_nombre`, `blog_rotulo`)
```

Crear una Tabla

Después de haber creado los campos y las claves, puede crear una nueva tabla con:

```
$this->dbforge->create_table('nombre_de_tabla');  
// Devuelve: CREATE TABLE nombre_de_tabla
```

Un segundo parámetro opcional establecido a **TRUE** agrega una cláusula "IF NOT EXISTS" en la definición:

```
$this->dbforge->create_table('nombre_de_tabla', TRUE);  
// Devuelve: CREATE TABLE IF NOT EXISTS nombre_de_tabla
```

Eliminar una Tabla

Ejecuta una instrucción SQL DROP TABLE.

```
$this->dbforge->drop_table('nombre_de_tabla');  
// Devuelve: DROP TABLE IF EXISTS nombre_de_tabla
```

Renombrar una Tabla

Renombra una tabla.

```
$this->dbforge->rename_table('nombre_viejo_de_tabla', 'nombre_nuevo_de_tabla');
// Devuelve: ALTER TABLE nombre_viejo_de_tabla RENAME TO nombre_nuevo_de_tabla
```

Modificar Tablas

\$this->dbforge->add_column()

La función **add_column()** se usa para modificar una tabla existente. Acepta el mismo array de campos que la anterior y se puede usar para una cantidad ilimitada de campos adicionales.

```
$campos = array(
    'preferences' => array('type' => 'TEXT')
);

$this->dbforge->add_column('nombre_de_tabla', $campos);
// Devuelve: ALTER TABLE nombre_de_tabla ADD preferences TEXT
```

\$this->dbforge->drop_column()

Se usa para eliminar una columna de la tabla.

```
$this->dbforge->drop_column('nombre_de_tabla', 'columna_a_eliminar');
```

\$this->dbforge->modify_column()

El uso de esta función es idéntica al de **add_column()**, excepto que altera una columna existente en lugar de agregar una nueva. Para cambiar el nombre puede agregar una clave "name" en el array que define el campo.

```
$campos = array(
    'nombre_viejo' => array(
        'name' => 'nombre_nuevo',
        'type' => 'TEXT',
    ),
);

$this->dbforge->modify_column('nombre_de_tabla', $campos);
// Devuelve: ALTER TABLE nombre_de_tabla CHANGE nombre_viejo nombre_nuevo TEXT
```

Clase de Utilidades de Base de Datos

La Clase de **utilidades** de **base** de **datos** contiene **funciones** que **ayudan** a **administrar la base de datos**.

Inicializar la Clase de Utilidades

Importante: Para inicializar la clase de utilidades, el driver de la base de datos tiene que estar ejecutándose, ya que la clase de utilidades se basa en él.

Cargar la clase de utilidades como sigue:

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

Una vez inicializada, se accederá a la funciones usando el objeto **\$this->dbutil**:

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

\$this->dbutil->list_databases()

Devuelve un array con nombres de bases de datos:

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

foreach ($dbs as $db)
{
    echo $db;
}
```

\$this->dbutil->database_exists()

A veces es útil saber si existe una base de datos en particular. Devuelve un booleano **TRUE/FALSE**. Ejemplo de uso:

```
if ($this->dbutil->database_exists('nombre_base_de_datos'))
{
    // algún código...
}
```

Nota: Reemplace *nombre_base_de_datos* con el nombre de la tabla que está buscando. Esta función distingue entre mayúsculas y minúsculas.

\$this->dbutil->optimize_table('nombre_de_tabla')

Nota: Esta funcionalidad solo está disponible en base de datos MySQL/MySQLi.

Le permite optimizar una tabla usando el nombre de tabla indicado en el primer parámetro. Devuelve **TRUE/FALSE** basado en el éxito o fracaso:

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

Nota: No todas las plataformas de base de datos soportan optimización de tablas.

\$this->dbutil->repair_table('nombre_de_tabla')

Nota: Esta funcionalidad solo está disponible en base de datos MySQL/MySQLi.

Le permite reparar una tabla usando el nombre de tabla indicado en el primer parámetro. Devuelve **TRUE/FALSE** basado en el éxito o fracaso:

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

Nota: No todas las plataformas de base de datos soportan optimización de tablas.

\$this->dbutil->optimize_database()

Nota: Esta funcionalidad solo está disponible en base de datos MySQL/MySQLi.

Le permite **optimizar la base de datos** a la que la **clase database** está **conectada**. Devuelve un **array** que **contiene** el **mensaje de estado** de la **base de datos** o **FALSE** en caso de falla.

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

Nota: No todas las plataformas de base de datos soportan optimización de tablas.

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

Le permite **generar** un **archivo CSV** con el **resultado** de la **consulta**. El primer parámetro de la función tiene que contener el objeto resultado de su consulta. Ejemplo:

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

El segundo y tercer parámetros le permiten establecer el delimitador y el carácter de nueva línea. Por defecto los tabuladores se usan como delimitadores y "\n" como nueva línea. Ejemplo:

```
$delimiter = ",";
$newline = "\r\n";

echo $this->dbutil->csv_from_result($query, $delimiter, $newline)
```

Importante: Esta función **NO** escribirá el archivo CSV por Ud. Simplemente crea el CSV. Necesita escribir el archivo usando el Helper File.

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

Le permite generar un **archivo XML** con el **resultado de la consulta**. El primer parámetro espera un objeto resultado de consulta y el segundo puede contener un array opcional de parámetros de configuración. Ejemplo:

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

$query = $this->db->query("SELECT * FROM mi_tabla");

$config = array (
    'root'      => 'root',
    'element'   => 'element',
    'newline'   => "\n",
    'tab'        => "\t"
);

echo $this->dbutil->xml_from_result($query, $config);
```

Importante: Esta función **NO** escribirá el archivo XML por Ud. Simplemente crea el XML. Si necesita escribir el archivo, use el Helper File.

\$this->dbutil->backup()

Le permite hacer una **copia de respaldo** de **toda la base de datos** o de **tablas individuales**. Los datos de la copia de respaldo se pueden comprimir ya sea en formato Zip como Gzip.

Nota: Esta funcionalidad está solamente disponible para bases de datos MySQL.

Nota: Debido al limitado tiempo de ejecución y memoria disponible de PHP, la realización de la copia de respaldo de bases de datos muy grandes puede no ser posible. Si su base de datos es muy grande, puede necesitar hacer la copia de respaldo directamente desde el servidor SQL mediante la línea de comandos o que la haga el administrador del servidor si Ud no tiene privilegios de root.

Ejemplo de Uso

```
// Cargar la clase de utilidades de BD
$this->load->dbutil();

// Hacer copia de respaldo para la BD entera y asignarla a una variable
$backup =& $this->dbutil->backup();

// Cargar el helper file y escribir el archivo en el servidor
$this->load->helper('file');
write_file('/path/to/mybackup.gz', $backup);
```

```
// Cargar el helper download y enviar el archivo a su escritorio
$this->load->helper('download');
force_download('mybackup.gz', $backup);
```

Establecer las Preferencias de la Copia de Respaldo

Las preferencias de la copia de respaldo se establecen al enviar un array de valores al primer parámetro de la función **backup()**. Ejemplo:

```
$prefs = array(
    // Array de tablas para hacer copia de respaldo
    'tables'      => array('table1', 'table2'),
    // Lista de tablas para omitir en copia de respaldo
    'ignore'      => array(),
    // gzip, zip, txt
    'format'      => 'txt',
    // Nombre de archivo - NECESARIO SOLAMENTE CON ARCHIVOS ZIP
    'filename'    => 'mybackup.sql',
    // Si agrega sentencias DROP TABLE al archivo de copia de respaldo
    'add_drop'    => TRUE,
    // Si agrega datos INSERT al archivo de copia de respaldo
    'add_insert'  => TRUE,
    // Carácter de Nueva Línea usado en el archivo de copia de respaldo
    'newline'     => "\n"
);

$this->dbutil->backup($prefs);
```

Descripción de las Preferencias de la Copia de Respaldo

Preferencia	Valor por Defecto	Opciones	Descripción
tables	array vacío	Ninguna	Array de tablas que se quieren respaldar. Si queda en blanco, se exportan todas las tablas.
ignore	array vacío	Ninguna	Array con las tablas que quiere que la rutina de respaldo ignore.
format	gzip	gzip, zip, txt	Formato del archivo de exportación.
filename	fecha/hora actual	Ninguna	Nombre del archivo de copia de respaldo. El nombre se necesita solamente si se está usando compresión zip.
add_drop	TRUE	TRUE/FALSE	Si incluir sentencias DROP TABLE en el archivo de exportación SQL.
add_insert	TRUE	TRUE/FALSE	Si incluir sentencias INSERT en el archivo de exportación SQL.
newline	"\n"	"\n", "\r", "\r\n"	Tipo de nueva línea a usar en el archivo de exportación SQL

Clase Email

La robusta Clase **Email** de CodeIgniter soporta las siguientes características:

- Varios Protocolos: Mail, Sendmail y SMTP
- Varios receptores
- CC y BCCs
- Correo HTML o de texto plano
- Adjuntos
- Salto de Línea Automático
- Prioridades
- Modo por Lotes BCC, que permite que listas grandes de email se corten en pequeños lotes BCC.
- Herramientas de Depuración de Email

Enviar Email

Enviar un email no es solamente sencillo, sino que se puede configurar al vuelo o establecer las preferencias en un archivo de configuración.

Aquí hay un ejemplo básico que muestra cómo se puede enviar un email. **Nota:** Este ejemplo asume que se está enviando el email desde uno de sus controladores.

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

$this->email->from('tu@ejemplo.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('Email de Prueba');
$this->email->message('Probando la Clase Email.');

$this->email->send();

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

Establecer Preferencias de Email

Hay 17 preferencias distintas disponibles para configurar los mensajes de email que se envían. Tanto puede establecerlas manualmente como se describe aquí, como automáticamente mediante preferencias almacenadas en archivos de configuración, tal como se describe más abajo:

Las preferencias se establecen al pasar un array de valores de preferencias a la función que inicializa al email. Este es un ejemplo de como 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 valores por defecto que se usarán si Ud no las establece.

Establecer Preferencias de Email en un Archivo de Configuración

Si prefiere no establecer las preferencias usando el método anterior, en su lugar puede ponerlas en un archivo de configuración. Simplemente cree un nuevo archivo llamado **email.php** y agréguele el array **\$config**. Luego guarde el archivo en **config/email.php** y se usará automáticamente. NO tiene necesidad de usar la función **\$this->email->initialize()** si guarda las preferencias en un archivo de configuración.

Preferencias de Email

La siguiente es la lista de todas las preferencias que se pueden establecer al enviar un email.

Preferencia	Valor por Defecto	Opciones	Descripción
useragent	CodeIgniter	Ninguno	El "agente de usuario"
protocol	mail	mail, sendmail o smtp	Protocolo de envío correo
mailpath	/usr/sbin/sendmail	Ninguno	Ruta del servidor a Sendmail
smtp_host	Sin Valor por Defecto	Ninguno	Dirección SMTP del Servidor
smtp_user	Sin Valor por Defecto	Ninguno	Usuario SMTP
smtp_pass	Sin Valor por Defecto	Ninguno	Contraseña SMTP
smtp_port	25	Ninguno	Puerto SMTP
smtp_timeout	5	Ninguno	Tiempo de Espera SMTP (en segundos)
wordwrap	TRUE	TRUE o FALSE (booleano)	Habilitar salto de línea automático
wrapchars	76		Cantidad de caracteres para el salto de línea automático
mailtype	text		Tipo de correo. Si envía correo HTML, tiene que enviarlo como una página web completa. Asegúrese que no tiene enlaces relativos o rutas relativas de imágenes, porque en caso contrario no funcionarán
charset	utf-8		Conjunto de caracteres (utf-8, iso-8859-1, etc.)
validate	FALSE	TRUE o FALSE (booleano)	Si validar la dirección de email
priority	3	1, 2, 3, 4, 5	Prioridad del Email. 1 = la más alta. 5 = la más baja. 3 = normal
crlf	\n	"\r\n", "\n" o "\r"	Carácter de Nueva Línea. (Use "\r\n" para cumplir con RFC 822)
newline	\n	"\r\n", "\n" o "\r"	Carácter de Nueva Línea. (Use "\r\n" para cumplir con RFC 822)
bcc_batch_mode	FALSE	TRUE o FALSE (booleano)	Habilitar Modo por Lotes BCC
bcc_batch_size	200	Ninguno	Cantidad de emails en cada lote BCC

Referencia de Funciones de Email

\$this->email->from()

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

```
$this->email->from('jose@ejemplo.com', 'Jose Perez');
```

\$this->email->reply_to()

Establece la dirección "responder a". Si no se provee, se usa la información de la función **from()**. Ejemplo:

```
$this->email->from('jose@ejemplo.com', 'Jose Perez');
```

\$this->email->to()

Establece la dirección de email del receptor. Puede ser un email simple, una lista delimitada por comas o un array:

```
$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()

Establece la dirección de email CC. Igual que con la función **to()**, puede ser un email simple, una lista delimitada por comas o un array.

\$this->email->bcc()

Establece la dirección de email BCC. Igual que con la función **to()**, puede ser un email simple, una lista delimitada por comas o un array.

\$this->email->subject()

Establece el asunto del email:

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

\$this->email->message()

Establecer el cuerpo del mensaje de email:

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

\$this->email->set_alt_message()

Establecer el cuerpo del mensaje alternativo de email:

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

Esta es una cadena de mensaje opcional que se puede usar si envía un correo formateado en HTML. Le permite especificar un mensaje alternativo sin formato HTML que se agrega a la cadena del encabezado para gente que no acepta email de HTML. Si no establecer su propio mensaje, CodeIgniter extraerá el mensaje de su email de HTML y le quitará las etiquetas.

\$this->email->clear()

Inicializa todas las variables a un estado vacío. Esta función está pensada para usarse si ejecuta la función de envío de email en un bucle, permitiendo que los datos se restablezcan entre ciclos.

```
foreach ($list as $name => $address)
{
    $this->email->clear();

    $this->email->to($address);
    $this->email->from('jose@ejemplo.com');
    $this->email->subject('Aqui está su información '.$name);
    $this->email->message('Hola '.$name.' Esta es la información solicitada.');
    $this->email->send();
}
```

Si establece el parámetro a **TRUE**, cualquier adjunto también será eliminado:

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

\$this->email->send()

Función de envío de Email. Devuelve el booleano **TRUE** o **FALSE** basada en el éxito o fracaso, permitiendo que sea usada condicionalmente:

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

\$this->email->attach()

Le **permite enviar** un **adjunto**. Poner ruta/archivo en el primer parámetro. **Nota:** Use una ruta de archivo, **no** una URL. Para varios adjuntos use la función varias veces. Por ejemplo:

```
$this->email->attach('/ruta/a/foto1.jpg');
$this->email->attach('/ruta/a/foto2.jpg');
$this->email->attach('/ruta/a/foto3.jpg');

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

\$this->email->print_debugger()

Devuelve una cadena conteniendo cualquier mensaje de servidor, encabezados de email y el mensaje de email. Útil para depuración.

Anular el Salto de Línea Automático

Si tiene el salto de línea automático habilitado (recomendado para cumplir con RFC 822) y tiene un enlace muy largo en su email, puede ocurrir que el salto de línea automático lo corte, causando que se vuelva imposible de cliquear por la persona que lo recibe. CodeIgniter le permite anular manualmente el salto de línea automático dentro de parte de su mensaje, así:

```
El texto de su email que  
normalmente salta de linea.
```

```
{unwrap}http://ejemplo.com/un_enlace_largo_que_no_deberia_cortarse.html{/unwrap}
```

```
Más texto que normalmente  
tiene su salto de linea.
```

Ubique el ítem que no quiere que se corte al final de la línea entre: **{unwrap} {/unwrap}**.

Clase Encrypt

La Clase **Encrypt** provee encriptación de datos de dos vías. Usa un esquema que, o bien compila el mensaje usando un esquema de codificación XOR a nivel de bits con un algoritmo hash aleatorio, o encripta usando la biblioteca Mcrypt. Si Mcrypt no está disponible en su servidor, el mensaje codificado todavía proveerá un grado razonable de seguridad para sesiones encriptadas u otros fines "ligeros". Si Mcrypt está disponible, proveerá un alto grado de seguridad adecuado para almacenamiento.

Establecer su Clave

Una *clave* es una pieza de información que controla el proceso criptográfico y permite que una cadena codificada se pueda decodificar. De hecho, la clave que elija **solamente** proveerá medios para decodificar datos que se hayan encriptado con esa clave, por lo tanto, no solo tiene que elegir cuidadosamente la clave, sino que nunca tiene que cambiarla si tiene pensado usarla con datos persistentes.

No hace falta decir que tiene que proteger cuidadosamente su clave. Si alguien accediera a su clave, los datos serían fáciles de decodificar. Si su servidor no está totalmente bajo su control es imposible garantizar la seguridad de su clave, por lo que tiene que pensar cuidadosamente antes de usarla para cualquier cosa que necesite alta seguridad, como almacenar números de tarjetas de crédito.

Para sacar la máxima ventaja del algoritmo de encriptación, su clave debería tener 32 caracteres de longitud (128 bits). La clave debería ser tan aleatoria como pueda, con números y letras en mayúsculas y minúsculas. Su clave **no** debería ser una cadena de texto simple. A fin de ser criptográficamente segura necesita ser tan aleatoria como sea posible.

Su clave se puede almacenar tanto en su archivo **application/config/config.php**, como puede diseñar su propio mecanismo de almacenamiento y pasar la clave dinámicamente al codificar o decodificar.

Para guardar su clave en su **application/config/config.php**, abra el archivo y establezca:

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

Longitud del Mensaje

Es importante que sepa que los mensajes codificados que la función de encriptación genera serán aproximadamente 2,6 veces más largos que el mensaje original. Por ejemplo, si encripta la cadena "mis datos super secretos", que tiene 24 caracteres de longitud, va a terminar con una cadena codificada de casi 63 caracteres (decimos "casi" porque la longitud de la cadena codificada se incrementa en bloques de 64 bits, por lo que no es exactamente lineal). Tenga presente esta información al seleccionar su mecanismo de almacenamiento de datos. Las cookies, por ejemplo, solo pueden mantener 4K de información.

Iniciar la Clase

Como la mayoría de las clases en CodeIgniter, la Clase **Encrypt** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez cargada, el objeto de la biblioteca Encrypt estará disponible usando: **\$this->encrypt**.

\$this->encrypt->encode()

Realiza la encriptación de datos y los devuelve como una cadena. Ejemplo:

```
$msg = 'Mi mensaje secreto';  
  
$encrypted_string = $this->encrypt->encode($msg);
```

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

```
$msg = 'Mi mensaje secreto';  
$key = 'clave-super-secreta';  
  
$encrypted_string = $this->encrypt->decode($msg, $key);
```

\$this->encrypt->set_cipher()

Le permite establecer un cifrador Mcrypt. Por defecto se usa **MCRYPT_RIJNDAEL_256**. Ejemplo:

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

Por favor visite php.net para el listado completo de [cifradores disponibles](#).

Si desea probar manualmente si su servidor soporta Mcrypt, puede usar:

```
echo ( ! function_exists('mcrypt_encrypt')) ? 'No' : 'Si';
```

\$this->encrypt->set_mode()

Le permite establecer el modo Mcrypt. Por defecto se usa **MCRYPT_MODE_CBC**. Ejemplo:

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

Por favor visite php.net para el listado completo de [modos disponibles](#).

\$this->encrypt->sha1()

Función de codificación SHA1. Provee una cadena y devolverá un hash de una sola vía de 160 bits. **Nota:** SHA1, igual que MD5 no es decodificable. Ejemplo:

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

Algunas instalaciones de PHP tienen soporte de SHA1 por defecto, por lo que si lo que necesita es codificar un hash, es más simple usar la función nativa:

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

Si su servidor no soporta SHA1 puede usar la función provista.

\$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/d	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/d	Clave de encriptación. Se lo especifica normalmente en su archivo de configuración como se describe anteriormente.

Validación de Formularios

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

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, ponerlas en cascada, e incluso preparar y preprocessar 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'    => 'Contraseña',
        '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 ejemplo 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 acepte 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:

En programación de computadoras, una devolución de llamada o retrollamada (en inglés: callback) es una función "A" que se usa como argumento de otra función "B". Cuando se llama a "B", ésta ejecuta "A". Para conseguirlo, usualmente lo que se pasa a "B" es el puntero a "A".

```

<?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 == 'test')
        {
            $this->form_validation->set_message('username_check',
                'The %s field can not be the word "test"');
            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' => 'Usuario',
            'rules' => 'required'
        ),
        array(
            'field' => 'password',
            'label' => 'Contraseña',
            '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' => 'PasswordConfirmation',
            'rules' => 'matches[password]'
        )
    )
);
```

```

        '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" />
```

O si usa un array multidimensional:

```
<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.	
matches	Sí	Devuelve FALSE si el elemento de formulario no coincide con el parámetro.	matches [form_item]
min_length	Sí	Devuelve FALSE si el elemento de formulario es más corto que el valor del parámetro.	min_length[6]
max_length	Sí	Devuelve FALSE si el elemento de formulario es más largo que el valor del parámetro.	max_length[12]
exact_length	Sí	Devuelve FALSE si el elemento de formulario no es exactamente el valor del parámetro.	exact_length[8]
greater_than	Sí	Devuelve FALSE si el elemento de formulario es menor que el valor del parámetro o no es numérico.	greater_than[8]
less_than	Sí	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 carácter alfabético .	
alpha_numeric	No	Devuelve FALSE si el elemento de formulario contiene algo que no es un carácter alfanumérico .	
alpha_dash	No	Devuelve FALSE si el elemento de formulario contiene algo que no es un carácter alfanumérico, guión de subrayado o guión común.	
numeric	No	Devuelve FALSE si el elemento de formulario contiene algo que no es un carácter numérico .	
integer	No	Devuelve FALSE si el elemento de formulario contiene algo que no es un entero .	
decimal	Sí	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.	

Regla	Parámetro	Descripción	Ejemplo
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.	
valid_base64	No	Devuelve FALSE si la cadena proporcionada contiene algo que no es un carácter Base64.	

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 opcionalmente 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 parámetro (opcional) le permite establecer un valor por defecto para el formulario. Ejemplo:

```
<input type="text" name="cantidad"
      value="<?php echo set_value('cantidad', '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:

```
<select name="mi_select">
    <option value="uno"
        <?php echo set_select('mi_select', 'uno', TRUE); ?> >Uno
    </option>
    <option value="dos"
        <?php echo set_select('mi_select', 'dos'); ?> >Dos
    </option>
    <option value="tres"
        <?php echo set_select('mi_select', 'tres'); ?> >Tres
    </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('mi_check[]', '1'); ?> />
<input type="checkbox" name="mi_check[]" value="2"
    <?php echo set_checkbox('mi_check[]', '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('mi_radio', '1', TRUE); ?> />
<input type="radio" name="mi_radio" value="2"
    <?php echo set_radio('mi_radio', '2'); ?> />
```

Clase FTP

La Clase **FTP** de CodeIgniter permite transferir archivos a un servidor remoto. Los archivos remotos también se pueden mover, renombrar, y eliminar. La Clase **FTP** también incluye una función de "espejado" que permite recrear remotamente un directorio local entero mediante FTP.

Nota: No están soportados los protocolos SFTP y SSL, solamente el FTP estándar.

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, la Clase **FTP** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez cargada, el objeto FTP estará disponible usando: **\$this->ftp**.

Ejemplos de Uso

En este ejemplo, se abre una conexión a un servidor FTP, se lee y sube un archivo local en modo ASCII. Los permisos de archivo se establecen a 755. **Nota:** Para establecer permisos se necesita PHP 5.

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

$config['hostname'] = 'ftp.ejemplo.com';
$config['username'] = 'su-usuario';
$config['password'] = 'su-contraseña';
$config['debug'] = TRUE;

$this->ftp->connect($config);

$this->ftp->upload('/local/ruta/a/mi_archivo.html',
    '/public_html/mi_archivo.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.ejemplo.com';
$config['username'] = 'su-usuario';
$config['password'] = 'su-contraseña';
$config['debug'] = TRUE;

$this->ftp->connect($config);

$list = $this->ftp->list_files('/public_html/');
print_r($list);

$this->ftp->close();
```

En este ejemplo un directorio local se espeja en el servidor.

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

$config['hostname'] = 'ftp.ejemplo.com';
$config['username'] = 'su-usuario';
$config['password'] = 'su-contraseña';
$config['debug'] = TRUE;

$this->ftp->connect($config);

$this->ftp->mirror('/ruta/a/mi_carpeta/', '/public_html/mi_carpeta/');

$this->ftp->close();
```

Referencia de Funciones

\$this->ftp->connect()

Conecta e inicia sesión en el servidor FTP. Las preferencias de conexión se establecen pasando un array a la función, o se pueden almacenar en un archivo de configuración.

Aquí hay un ejemplo que muestra cómo establecer las preferencias manualmente:

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

$config['hostname'] = 'ftp.ejemplo.com';
$config['username'] = 'su-usuario';
$config['password'] = 'su-contraseña';
$config['port'] = 21;
$config['passive'] = FALSE;
$config['debug'] = TRUE;

$this->ftp->connect($config);
```

Establecer Preferencias FTP en un Archivo de Configuración

Si lo prefiere, puede almacenar sus preferencias FTP en un archivo de configuración. Simplemente cree un nuevo archivo llamado **ftp.php** y agréguele el array **\$config**. Luego guarde el archivo en **config/ftp.php** y se lo usará automáticamente.

Las opciones de conexión disponibles son:

- **hostname** - Nombre del host de FTP. Normalmente algo como: ftp.ejemplo.com
- **username** - Nombre del usuario de FTP.
- **password** - Contraseña de FTP.
- **port** - Número de puerto. Establecido a **21** por defecto.
- **debug** - **TRUE/FALSE** (booleano). Si habilitar la depuración para mostrar mensajes de error.
- **passive** - **TRUE/FALSE** (booleano). Si usar el modo pasivo. Por defecto está establecido automáticamente el modo pasivo.

\$this->ftp->upload()

Sube un archivo a su servidor. Tiene que proporcionar las rutas local y remota. Opcionalmente puede establecer el modo y los permisos. Ejemplo:

```
$this->ftp->upload('/local/ruta/a/mi_archivo.html',
    '/public_html/mi_archivo.html', 'ascii', 0775);
```

Las opciones de modo son: **ascii**, **binary** y **auto** (valor por defecto). Si se usa **auto**, basará el modo en la extensión del archivo origen.

Los permisos están disponibles si está ejecutando 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/mi_archivo.html',
    '/local/ruta/a/mi_archivo.html', 'ascii');
```

Las opciones de modo son: **ascii**, **binary** y **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()

Le permite renombrar un archivo. Proporcione la **ruta/nombre** del archivo origen y la **ruta/nombre** del nuevo archivo.

```
// Renombra verde.html a azul.html
$this->ftp->rename('/public_html/foo/verde.html', '/public_html/foo/azul.html');
```

\$this->ftp->move()

Le permite mover un archivo. Proporcione las rutas de origen y destino:

```
// Mueve blog.html desde "jose" a "lucas"
$this->ftp->move('/public_html/jose/blog.html', '/public_html/lucas/blog.html');
```

Nota: si el nombre del archivo destino es diferente, el archivo se renombrará.

\$this->ftp->delete_file()

Le permite eliminar un archivo. Proporcione la ruta origen con el nombre del archivo.

```
$this->ftp->delete_file('/public_html/jose/blog.html');
```

\$this->ftp->delete_dir()

Le permite eliminar un directorio y todo lo que contiene. Proporcione la ruta origen para el directorio con una barra al final.

Importante: Sea **MUY** cuidadoso con esta función. Eliminará recursivamente todo lo que esté dentro de la ruta proporcionada, incluyendo subcarpetas y sus archivos. Asegúrese absolutamente que la ruta es correcta. Pruebe primero usando la función **list_files()** para comprobar que la ruta es correcta.

```
$this->ftp->delete_dir('/public_html/ruta/a/carpeta/');
```

\$this->ftp->list_files()

Le permite recuperar una lista de archivos en su servidor, devuelta como un **array**. Tiene que 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 (incluyendo subcarpetas) y crea un espejo mediante FTP basado en él. Cualquiera que sea la estructura de directorios de la ruta del archivo original será recreado en el servidor. Debe proporcionar una ruta de origen y una ruta de destino:

```
$this->ftp->mirror('/ruta/a/mi_carpeta/', '/public_html/mi_carpeta/');
```

\$this->ftp->mkdir()

Le permite crear un subdirectorio en su servidor. Proporcione la ruta que termina en el nombre de la carpeta que desea crear con una barra final. Los permisos se pueden establecer al pasar un valor **octal** en el segundo parámetro (si está ejecutando PHP 5).

```
// Crea una carpeta llamada "bar"
$this->ftp->mkdir('/public_html/foo/bar/', DIR_WRITE_MODE);
```

\$this->ftp->chmod()

Le permite establecer los permisos de archivo. Proporcione la ruta al archivo o carpeta a los que desea alterar los permisos:

```
// Chmod "bar" a 777
$this->ftp->chmod('/public_html/foo/bar/', DIR_WRITE_MODE);
```

\$this->ftp->close()

Cierra la conexión al servidor. Se recomienda que la use una vez que termine de subir.

Clase Image_lib

La Clase **Image_lib** de CodeIgniter le permite realizar las siguientes acciones:

- Redimensionamiento de Imágenes
- Creación de Miniaturas
- Recorte de Imágenes
- Rotación de Imágenes
- Marca de Agua en Imágenes

Se soportan las tres principales bibliotecas de imágenes: GD/GD2, NetPBM e ImageMagick.

Nota: La funcionalidad de marca de agua solamente está disponible usando la biblioteca GD/GD2. Además, aún cuando se soportan otras bibliotecas, GD es requerida para que el script calcule las propiedades de la imagen. Sin embargo, el procesamiento de la imagen será realizado con la biblioteca especificada.

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, la clase **Image_lib** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez que la biblioteca esté cargada, estará lista para usarse. El objeto de la biblioteca **Image_lib** que usará para llamar a todas las funciones es: **\$this->image_lib**.

objeto

Procesar una Imagen

Sin importar el tipo de procesamiento que quiera realizar (redimensionar, recortar, rotar o poner marca de agua), el proceso general es idéntico. Establecerá algunas preferencias correspondientes a la acción que tiene pensado ejecutar y luego llamar a una de las cinco funciones de procesamiento de imágenes disponibles. Por ejemplo, para crear una imagen en miniatura, hará esto:

```
$config['image_library'] = 'gd2';
$config['source_image'] = '/ruta/a/imagen/mi_pic.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 una imagen llamada **mi_pic.jpg** ubicada en la carpeta **source_image** y que luego cree una miniatura de 75 X 50 pixeles usando la **image_library GD2**. Como la opción **maintain_ratio** está habilitada, la miniatura estará tan cerca del **ancho** y del **alto** definitivo como sea posible mientras se preserva la relación de aspecto original. La miniatura se llamará **mi_pic_thumb.jpg**.

Nota: Para que la clase **Image_lib** pueda hacer cualquier procesamiento, la carpeta que contiene los archivos de **imagen** tiene que tener permisos de escritura.

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 cinco funciones de procesamiento disponibles:

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

Estas funciones devuelven el booleano **TRUE** en caso de éxito o **FALSE** en caso de falla. Si fallan, Ud puede recuperar el mensaje de error usando esta función:

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

Una buena práctica es usar condicionalmente la función de procesamiento, mostrando el error en caso de falla, de esta forma:

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

Nota: Opcionalmente puede especificar que se aplique el formato HTML a los errores, enviando las etiquetas de apertura/cierre en la función:

```
$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.

Advierta que no todas las preferencias están disponibles para cada función. Por ejemplo, las preferencias del eje x/y solo están disponibles para el recorte de Imágenes. Del mismo modo, las preferencias de ancho y alto no tienen efecto en el recorte. La columna "disponibilidad" indica las funciones soportadas para una dada preferencia.

Disponibilidad (columna Disp.):

- **R** - Redimensionamiento de Imágenes
- **C** - Recorte de Imágenes
- **X** - Rotación de Imágenes
- **W** - Marca de Agua en Imágenes

Preferencia	Valor por defecto	Opciones	Descripción	Disp.
image_library	GD2	GD, GD2, ImageMagick, NetPBM	Establece la biblioteca de imagen a utilizarse.	R, C, X, W
library_path	Ninguno	Ninguno	Establece la ruta del servidor para la biblioteca ImageMagick o NetPBM. Si utiliza cualquiera de las bibliotecas debe proporcionar la ruta.	R, C, X
source_image	Ninguno	Ninguno	Establece la ruta/nombre original de la imagen. La ruta tiene que ser una ruta relativa o absoluta del servidor, no una URL.	R, C, S, W
dynamic_output	FALSE	TRUE/FALSE (booleano)	Determina si la nueva imagen debería escribirse al disco o generarse dinámicamente. Nota: Si elije la opción dinámica, solamente se puede mostrar una imagen por vez y no se puede posicionar en la página. Simplemente imprime dinámicamente la imagen en crudo en el navegador, junto con los encabezados de imagen.	R, C, X, W
quality	90%	1 - 100%	Establece la calidad de la imagen. Cuanto mayor sea la calidad, mayor será el tamaño del archivo.	R, C, X, W
new_image	Ninguno	Ninguno	Establece la ruta/nombre destino de la imagen. Usará esta preferencia cuando cree una nueva copia. La ruta tiene que ser una ruta relativa o absoluta del servidor, no una URL	R, C, X, W
width	Ninguno	Ninguno	Establece el ancho que quisiera que la imagen tenga.	R, C
height	Ninguno	Ninguno	Establece el alto que quisiera que la imagen tenga.	R, C
create_thumb	FALSE	TRUE/FALSE (booleano)	Le dice a la función de procesamiento de imágenes que cree una miniatura.	R
thumb_marker	_thumb	Ninguno	Especifica el indicador de miniatura. Se insertará justo antes de la extensión del archivo, por lo que mi_pic.jpg se convertirá en mi_pic_thumb.jpg	R
maintain_ratio	TRUE	TRUE/FALSE (booleano)	Especifica si mantener la relación de aspecto al redimensionar o usar valores rígidos.	R, C
master_dim	auto	auto, width, height	Especifica que usar como eje maestro al redimensionar o crear miniaturas. Por ejemplo, digamos que quiere redimensionar una imagen a 100 X 75 pixeles. Si el tamaño de la imagen original no permite un redimensionamiento perfecto para esas dimensiones, este valor determina cual eje debería usarse como valor rígido. "auto" establece automáticamente el eje basado en si la imagen es más ancha o viceversa.	R
rotation_angle	Ninguno	90, 180, 270, vrt, hor	Especifica el ángulo de rotación al rotar imágenes. Advierta que PHP rota en sentido antihorario, por lo tanto una rotación de 90 grados se tiene que especificar como 270.	X
x_axis	Ninguno	Ninguno	Establece la coordenada X para el recorte de imágenes. Por ejemplo, un valor de 30 recortará la imagen 30 pixeles desde la izquierda.	C
y_axis	Ninguno	Ninguno	Establece la coordenada Y para el recorte de imágenes. Por ejemplo, un valor de 30 recortará la imagen 30 pixeles desde arriba.	C

Establecer Preferencias en un Archivo de Configuración

Si prefiere no establecer las preferencias usando el método anterior, en su lugar puede ponerlas en un archivo de configuración. Simplemente cree un nuevo archivo llamado **image_lib.php** y agregue el array **\$config** en ese archivo. Luego guarde el archivo en **config/image_lib.php** y se usará automáticamente. **NO** necesitará usar la función **\$this->image_lib->initialize** si guarda las preferencias en un archivo de configuración.

\$this->image_lib->resize()

La función de redimensionamiento de imágenes le permite redimensionar la imagen original, crear una copia (con o sin redimensionarla), o crear una imagen en miniatura.

Para propósitos prácticos, no hay diferencia entre copiar y crear una miniatura, excepto que tendrá un indicador de miniatura como parte del nombre (por ejemplo, **mi_pic_thumb.jpg**).

Todas las preferencias listadas en la tabla anterior están disponibles para esta función, excepto estas tres: **rotation_angle**, **x_axis** y **y_axis**.

Crear una Miniatura

La función de redimensionamiento creará un archivo de miniatura (y preservará el original) si establece esta preferencia a **TRUE**:

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

Esta preferencia simple determina si la miniatura se crea o no.

Crear una Copia

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

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

Notas acerca de esta Preferencia:

- Si se especifica solamente el nuevo nombre de la imagen, ésta se colocará en la misma carpeta que la original
- Si se especifica solamente la ruta, la nueva imagen se colocará en el destino con el mismo nombre que la original.
- Si se especifican tanto la ruta como el nombre de la imagen, ésta se colocará en su propio destino y con el nombre dado.

Redimensionar la Imagen Original

Si no se usa ninguna de las dos preferencias listadas anteriormente (**create_thumb** o **new_image**), en su lugar la función de redimensionamiento se centrará en la imagen original para su procesamiento.

\$this->image_lib->crop()

La función de recorte trabaja casi igual a la función de redimensionamiento, salvo que necesita que se le establezcan las preferencias para los ejes X e Y (en pixeles) que indican donde recortar:

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

Todas las preferencias listadas en la tabla anterior están disponibles para esta función, excepto estas: **rotation_angle, width, height, create_thumb, new_image**.

Aquí hay un ejemplo que muestra cómo puede recortar una imagen:

```
$config['image_library'] = 'imagemagick';
$config['library_path'] = '/usr/X11R6/bin/';
$config['source_image'] = '/ruta/a/imagen/mi_pic.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 interfaz visual es difícil recortar imágenes, por lo tanto esta función no es muy útil a menos que tenga pensado armar esa interfaz. Eso es exactamente lo que hicimos en ExpressionEngine, el CMS que desarrollamos, usando el módulo de galería de fotos. Agregamos una interfaz gráfica en JavaScript que permite seleccionar el área de recorte.

\$this->image_lib->rotate()

La función de rotación de imágenes requiere que el ángulo de rotación sea establecido mediante una preferencia:

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

Hay cinco opciones de rotación:

- **90** - rota 90 grados en forma antihoraria.
- **180** - rota 180 grados en forma antihoraria.
- **270** - rota 270 grados en forma antihoraria.
- **hor** - volteá la imagen horizontalmente.
- **vert** - volteá la imagen verticalmente.

Aquí hay un ejemplo que muestra cómo puede rotar una imagen:

```
$config['image_library'] = 'netpbm';
$config['library_path'] = '/usr/bin/';
$config['source_image'] = '/ruta/a/imagen/mi_pic.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** permite restablecer todos los valores usados al procesar una imagen. Deseará llamar a esta función si está procesando imágenes en un bucle.

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

Marca de Agua en Imágenes

La funcionalidad de marca de agua necesita de la biblioteca GD/GD2.

Dos Tipos de Marca de Agua

Hay dos tipos de marca de agua que se pueden usar:

1. **Text:** El mensaje de marca de agua se generará usando un texto, sea tanto con una fuente True Type que le especifique, o usando una salida de texto nativo que soporte la biblioteca GD. Si usa la versión True Type, su instalación GD tiene que compilarse con soporte para True Type (la mayoría lo están, pero no todas).
2. **Overlay:** El mensaje de marca de agua se generará superponiendo una imagen (usualmente un GIF o PNG transparente) que contiene la marca de agua sobre la imagen original.

Poner Marca de Agua a una Imagen

Al igual que con las otras funciones (redimensionar, recortar y rotar) el proceso general para poner la marca de agua implica establecer las preferencias correspondientes a la acción que intenta ejecutar y luego llamar a la función que pone la marca de agua. Aquí hay un ejemplo:

```
$config['source_image'] = '/ruta/a/imagen/mi_pic.jpg';
$config['wm_text'] = 'Copyright 2011 - Jose Perez';
$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();
```

En el ejemplo anterior usamos una fuente True Type de 16 pixeles para crear el texto "Copyright 2011 - Jose Perez". La marca de agua se ubicará centrada en la parte inferior de la imagen, 20 pixeles desde el borde inferior de la imagen.

Nota: Para que la clase Image_lib pueda hacer cualquier procesamiento, el archivo de imagen tiene que tener permisos de "escritura". Por ejemplo, 777.

Preferencias de Marca de Agua

Esta tabla muestra las preferencias que están disponibles para ambos tipos de marca de agua (text u overlay).

Preferencia	Valor por Defecto	Opciones	Descripción
wm_type	text	text, overlay	Establece el tipo de marca de agua que se debería usar.
source_image	Ninguno	Ninguno	Establece la ruta/nombre origen de la imagen. La ruta tiene que ser una ruta relativa o absoluta del servidor, no una URL.
dynamic_output	FALSE	TRUE/FALSE (booleano)	Determina si el nuevo archivo de imagen debería ser escrito al disco o ser generado dinámicamente. Nota: Si elije la opción dinámica, solamente se puede mostrar una imagen por vez y no se puede posicionarla en la página. Simplemente imprime dinámicamente una imagen en crudo en el navegador, junto con los encabezados de imagen.
quality	90%	1 - 100%	Establece la calidad de la imagen. Cuanto mayor sea la calidad, mayor será el tamaño del archivo.
padding	Ninguno	Un número	Cantidad de relleno, establecido en pixeles, que se aplicará a la marca de agua para alejarla de los bordes de su imagen.
wm_vrt_alignment	bottom	top, middle, bottom	Establece la alineación vertical para la imagen de la marca de agua.
wm_hor_alignment	center	left, center, right	Establece la alineación horizontal para la imagen de la marca de agua.
wm_hor_offset	Ninguno	Ninguno	Especifica el desplazamiento horizontal (en pixeles) a aplicar a la posición de la marca de agua. El desplazamiento mueve normalmente la marca de agua a la derecha, excepto si su alineación está establecida a "right", por lo que el valor de desplazamiento moverá la marca de agua hacia la izquierda de la imagen.
wm_vrt_offset	Ninguno	Ninguno	Especifica el desplazamiento vertical (en pixeles) a aplicar a la posición de la marca de agua. El desplazamiento mueve normalmente la marca de agua hacia abajo, excepto si su alineación está establecida a "bottom", por lo que el valor de desplazamiento moverá la marca de agua hacia la parte superior de la imagen.

Preferencias del Tipo "text"

Esta tabla muestra las preferencias que están disponibles para el tipo "text" de marca de agua.

Preferencia	Valor por Defecto	Opciones	Descripción
wm_text	Ninguno	Ninguno	Texto que quiere mostrar como marca de agua. Comúnmente será un aviso de copyright.
wm_font_path	Ninguno	Ninguno	Ruta del servidor para la fuente True Type que quiere usar. Si no usa esta opción, se usará la fuente GD nativa.
wm_font_size	16	Ninguno	Tamaño del texto. Nota: Si no está usando la opción True Type anterior, el número se establece usando el rango 1 - 5. De lo contrario, se puede usar cualquier tamaño de píxel válido para la fuente que está utilizando.
wm_font_color	FFFFFF	Ninguno	Color de la fuente, especificado en hexadecimal. Advierta que tiene que usar el valor hexadecimal completo de seis caracteres (por ejemplo, 993300), en lugar de la versión abreviada de tres caracteres (por ejemplo, fff).
wm_shadow_color	Ninguno	Ninguno	Color de la sombra, especificado en hexadecimal. Si deja esto en blanco, no se usará la sombra. Advierta que tiene que usar el valor hexadecimal completo de seis caracteres (por ejemplo, 993300), en lugar de la versión abreviada de tres caracteres (por ejemplo, fff).
wm_shadow_distance	3	Ninguno	Distancia (en pixeles) desde la fuente a donde la sombra debería aparecer.

Preferencias del Tipo "overlay"

Esta tabla muestra las preferencias que están disponibles para el tipo "overlay" de marca de agua.

Preferencia	Valor por Defecto	Opciones	Descripción
wm_text	Ninguno	Ninguno	Ruta del servidor a la imagen que desea usar como marca de agua. Requerida solamente si está usando el método "overlay".
wm_font_path	50	1 - 100	Opacidad de la imagen. Puede especificar la opacidad (es decir, transparencia) de su imagen marca de agua. Esto permite que la marca de agua sea apenas visible y que no oculte totalmente los detalles de la imagen original detrás suyo. Lo típico es una opacidad del 50%.
wm_font_size	4	Un número	Si la marca de agua es una imagen PNG o GIF, puede especificar un color en la imagen para que sea "transparente". Este valor (junto con el siguiente) le permitirá especificar ese color. Esto funciona especificando las coordenadas "x" e "Y" del píxel (medido desde la esquina superior izquierda) dentro de la imagen que corresponde al píxel representativo del color que quiere que sea transparente.
wm_font_color	4	Un número	Junto con el valor anterior, le permite especificar la coordenada para el píxel representante del color que quiere que sea transparente.

Clase Input

La Clase **Input** sirve para dos propósitos:

- Por seguridad, **preprocesa** los **datos de entrada globales**.
- Provee algunas **funciones helper** para **recuperar datos de entrada y preprocesarlos**.

Nota: El **sistema** **inicializa** **automáticamente** a **esta clase**, por lo que no hay necesidad de hacerlo manualmente.

Filtrado de Seguridad

La **función** de **filtrado** de **seguridad** se **llama** **automáticamente** cuando se **invoca** un **nuevo** **controlador**. Hace lo siguiente:

- Destruye el **array global GET**. Como CodeIgniter **no usa** **cadenas GET**, no hay razón para permitirlo
- Destruye **todas las variables globales** en caso que **register_globals** esté activada
- **Filtre las claves** de los **arrays POST/COOKIE**, permitiendo solamente **caracteres alfanuméricos** (y unos pocos más)
- **Provee filtrado XSS** (Cross-site Scripting Hacks). Esto se puede habilitar globalmente o bajo pedido
- Estandariza los caracteres de nueva línea a \n

Filtrado 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 o SERVER

CodeIgniter viene con **tres funciones helper** que le **permiten recuperar** **ítems** de **POST, COOKIE o SERVER**. La principal ventaja de **usar las funciones provistas** en lugar de **recuperar un ítem** directamente (**\$_POST['algo']**) es que las **funciones comprobarán** si el **ítem** está establecido y **devolverán FALSE** (booleano) si no lo está. Esto le **permite usar** convenientemente los **datos sin tener** que **probar primero** si el **ítem existe**. En otras palabras, comúnmente haría esto:

```
if ( ! isset($_POST['algo']))  
{  
    $algo = FALSE;  
}  
else  
{  
    $algo = $_POST['algo'];  
}
```

Con las funciones incorporadas de CodeIgniter, simplemente puede 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 de **POST** que está buscando:

```
$this->input->post('algun_dato');
```

La función devuelve **FALSE** (booleano) si el ítem que está intentando recuperar no existe.

El segundo parámetro (opcional) le permite pasar el dato a través del filtro XSS. Se habilita estableciendo el segundo parámetro al 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 pasarlo 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 pasarlo 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 través 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, salvo que recupera datos de cookie:

```
$this->input->cookie('algun_dato', TRUE);
```

\$this->input->server()

Esta función es idéntica a las funciones anteriores, salvo que recupera datos de server:

```
$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 'Inválido';
}
else
{
    echo 'Válido';
}
```

Acepta un segundo parámetro tipo string opcional con valor "IPv4" o "IPv6" para especificar el formato de IP. El default verifica 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. Devuelve un array de encabezados.

```
$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 Javascript

Nota: Este driver es experimental. Su conjunto de características e implementación puede cambiar en futuras versiones.

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 cargarán 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 <div id="notice_area"> 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. Advierte 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");
```

tablesorter()

Descripción pendiente

modal()

Descripción pendiente

calendar()

Descripción pendiente

Clase Lang

La Clase **Lang** provee funciones para recuperar archivos de idioma y líneas de texto a los efectos de la internacionalización.

En la carpeta **system** de CodeIgniter encontrará otra, llamada **language**, que contiene conjuntos de archivos de idioma. Puede crear sus propios archivos de idioma según necesite a fin de mostrar errores y otros mensajes en otros idiomas.

Los archivos de idiomas se almacenan normalmente en su directorio **system/language**. Alternativamente, puede crear una carpeta llamada **language** dentro de su carpeta **application** y almacenarlos allí. CodeIgniter primero buscará en su directorio **application/language**. Si el directorio no existe o el idioma especificado no está ubicado allí, CI en su lugar buscará en la carpeta global **system/language**.

Nota: Cada idioma deberá almacenarse en su propia carpeta. Por ejemplo, los archivos de inglés se ubican en: **system/language/english**.

Crear Archivos de Idioma

Los archivos de idioma se tiene que llamar con **_lang.php** como extensión del archivo. Por ejemplo, digamos que quiere crear un archivo contenido los mensajes de error. Puede llamarlo: **error_lang.php**

Dentro del archivo, asignará cada línea de texto a un array llamado **\$lang** con este prototipo:

```
$lang['clave_idioma'] = "Mensaje a mostrarse";
```

Nota: Es buena práctica usar un prefijo común para todos los mensajes en un dado archivo para evitar colisiones con elementos que se llaman igual en otros archivos. Por ejemplo, si está creando mensajes de error puede prefijarlos con **error_**.

```
$lang['error_email_missing'] = "Tiene que enviar un correo electrónico";
$lang['error_url_missing'] = "Tiene que enviar una URL";
$lang['error_username_missing'] = "Tiene que enviar un nombre de usuario";
```

Cargar un Archivo de Idioma

Al efecto de recuperar una línea de un archivo particular, primero tiene que cargar el archivo. Con el siguiente código se carga un archivo de idioma:

```
$this->lang->load('archivo', 'idioma');
```

Donde **archivo** es el nombre del archivo que desea cargar (sin la extensión de archivo) e **idioma** es el conjunto de idioma que lo contiene (por ejemplo, inglés). Si falta el segundo parámetro, se usará el idioma por defecto que está establecido en su archivo **application/config/config.php**.

Recuperar una Línea de Texto

Una vez que su idioma deseado se cargó, puede acceder a cualquier línea de texto usando esta función:

```
$this->lang->line('clave_idioma');
```

Donde **clave_idioma** es la clave del array correspondiente a la línea que desea mostrar.

Nota: Esta función simplemente devuelve la línea. No hace eco.

Usar Líneas de Idioma como Rótulos de Formulario

Esta funcionalidad se marcó como obsoleta en la biblioteca de idiomas y se movió a la función **lang()** del helper Language.

Carga Automática de Idiomas

Si encuentra la necesidad de tener un idioma cargado globalmente a lo largo de toda la 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 agregarle el idioma al array **\$autoload**.

Clase Load

La clase **Load**, como su nombre sugiere, se **usa** para **cargar** **elementos**. Estos **elementos** pueden **ser bibliotecas** (**clases**), **archivos** de **vistas**, **helpers**, **modelos**, o sus propios **archivos**.

Nota: El **sistema** **initializa** **automáticamente** a **esta clase**, por lo que **no hay necesidad** de **hacerlo** **manualmente**.

Las siguientes **funciones** están **disponibles** en esta clase:

\$this->load->library('nombre_de_clase', \$config, 'nombre_de_objeto')

Esta **función** se usa **para cargar** **clases** del **núcleo**. Donde **nombre_de_clase** es el **nombre** de la **clase** que **desea cargar**. **Nota:** Nosotros **usamos** los términos "**clase**" y "**biblioteca**" en forma intercambiable.

Por ejemplo, si quisiera **enviar** un **email** con CodeIgniter, el primer paso es **cargar** la **clase Email** dentro del **controlador**:

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

Una vez cargada, la biblioteca está **lista** para **usarse**, usando **\$this->email->alguna_funcion()**.

Los **archivos** de **biblioteca** se pueden **almacenar** en **subdirectorios** **dentro** de la **carpeta "libraries"** **principal**, o **dentro** de su **carpeta personal application/libraries**. Para **cargar** un **archivo** **localizado** en un **subdirectorio**, simplemente **incluya** la **ruta**, **relativa** a la **carpeta "libraries"**. Por ejemplo, si tiene localizado un archivo en:

```
libraries/flavors/chocolate.php  
sabores
```

Lo cargará usando:

```
$this->load->library('flavors/chocolate');
```

Puede **anidar** el **archivo** en tantos **subdirectorios** como **desea**.

Además, **se pueden** **cargar** **varias** **bibliotecas** al mismo tiempo **pasando** un **array** de **bibliotecas** a la **función** de **carga**.

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

Configurar opciones

El **segundo parámetro (opcional)** le **permite** **pasar** **opcionalmente** **valores** de **configuración**. Normalmente pasará esos valores como un array:

```
$config = array (  
    'mailtype' => 'html',  
    'charset'  => 'utf-8',  
    'priority' => '1'  
);  
  
$this->load->library('email', $config);
```

Las opciones de configuración normalmente también se pueden establecer mediante un archivo de configuración. Cada biblioteca se explica en detalle en su página, así que lea la información acerca de cada una de la que deseé usar.

Por favor advierta que cuando varias bibliotecas se pasan a un array en el primer parámetro, cada una recibirá la misma información de parámetro.

Asignar una Biblioteca a un Nombre de Objeto Diferente

Si el tercer parámetro (opcional) está en blanco, la biblioteca se asignará normalmente a un objeto con el mismo nombre que la biblioteca. Por ejemplo, si la biblioteca se llama **Session**, se asignará a una variable llamada `$this->session`.

Si prefiere establecer sus propios nombres de clases, puede pasar su valor al tercer parámetro:

```
$this->load->library('session', '', 'mi_session');

// Ahora se accede a la clase Session usando:

$this->mi_session
```

Por favor advierta que cuando varias bibliotecas se pasan a un array en el primer parámetro, este parámetro se descarta.

`$this->load->view('nombre_de_archivo', $data, TRUE/FALSE)`

Esta función se usa para cargar sus archivos de vistas. Si todavía no leyó la sección Vistas de la Guía del Usuario, se le recomienda que lo haga, ya que muestra cómo se usa normalmente esta función.

El primer parámetro es obligatorio. Es el nombre del archivo de vista que Ud quiere cargar. **Nota:** No se necesita especificar la extensión .php, a menos que esté usando otra distinta.

El segundo parámetro **opcional** puede tomar como entrada un array asociativo o un objeto, que se ejecuta mediante la función **extract()** de PHP para convertir a variables que se pueden usar en sus archivos de vistas. Nuevamente, lea la página Vistas para aprender cómo esto puede ser útil.

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

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

`$this->load->model('Nombre_modelo')`

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

Si su modelo está ubicado en una subcarpeta, incluir la ruta relativa de su carpeta de modelos. Por ejemplo, si tiene un modelo ubicado en **application/models/blog/queries.php** lo cargará usando:

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

Si quisiera tener su modelo asignado a un nombre de objeto diferente, puede especificarlo mediante el segundo parámetro de la función de carga:

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

\$this->load->database('opciones', TRUE/FALSE)

Esta función le permite cargar la Clase **Database**. Los dos parámetros son opcionales. Para más información lea la sección Clase **Database**.

\$this->load->vars(\$array)

Esta función toma como entrada un array asociativo y genera variables usando la función **extract()** de PHP. Esta función produce el mismo resultado que usar el segundo parámetro de la función **\$this->load->view()** anterior. La razón por la que puede desear usar esta función independientemente es si quisiera establecer alguna variable global en el constructor de su controlador y tenerlas disponibles en cualquier archivo de vista cargado desde cualquier función. Puede tener varias llamadas a esta función. Los datos se almacenan en caché y fusionan en un array para convertirlos en variables.

\$this->load->get_var(\$key)

Esta función verifica el array asociativo de variables disponible en sus vistas. Es útil si por cualquier razón una variable se establece en una biblioteca u otro método controlador usando **\$this->load->vars()**.

\$this->load->helper('nombre_de_archivo')

Esta función carga archivos de helper, donde **nombre_de_archivo** es el nombre del archivo sin la extensión **_helper.php**.

\$this->load->file('ruta_de_archivo/nombre_de_archivo', TRUE/FALSE)

Esta es una función genérica de carga de archivos. Proporcione la ruta y nombre del archivo en el primer parámetro y la función abrirá y leerá el archivo. Por defecto los datos se envían a su navegador, igual que un archivo de vista, pero si estableció el segundo parámetro a **TRUE** (booleano), en su lugar devolverá los datos como una cadena.

\$this->load->language('nombre_de_archivo')

Esta función es un alias de la función que carga los idiomas: **\$this->lang->load()**.

\$this->load->config('nombre_de_archivo')

Esta función es un alias de la función que carga los archivos de configuración: **\$this->config->load()**.

"Paquetes" de Aplicación

Un paquete de aplicación permite la fácil distribución de conjuntos completos de recursos en un directorio simple, como ser sus propias bibliotecas, modelos, helpers, configuraciones y archivos de idiomas. Se recomienda que estos paquetes se ubiquen en la carpeta **application/third_party**. A continuación se muestra un mapa de un directorio de paquete.

Ejemplo del Mapa del Directorio del Paquete "Foo Bar"

El siguiente es un ejemplo de un directorio para un paquete de aplicación llamado "Foo Bar".

```
/application/third_party/foo_bar

config/
helpers/
language/
libraries/
models/
```

Sea cual sea el propósito del paquete de aplicación "Foo Bar", tiene sus propios archivos de configuración, helpers, archivos de idioma, bibliotecas y modelos. Para usar estos recursos en sus controladores, primero necesita decirle a la Clase Load que va a cargar recursos desde un paquete, al agregarle la ruta del paquete.

\$this->load->add_package_path()

Esta función instruye a la Clase Load a anteponer una ruta dada para las solicitudes subsecuentes de recursos. Como ejemplo, el paquete de aplicación "Foo Bar" anterior tiene una biblioteca llamada **Foo_bar.php**. En su controlador, haremos lo siguiente:

```
$this->load->add_package_path(APPPATH.'third_party/foo_bar/');
$this->load->library('foo_bar');
```

\$this->load->remove_package_path()

Cuando su controlador haya terminado de usar recursos de un paquete de aplicación y particularmente, si tiene otros paquetes de aplicación con los que quiere trabajar, puede desear quitar la ruta del paquete para que la Clase Load no busque más recursos en esa carpeta. Para quitar la última ruta agregada, simplemente llame al método sin parámetros.

\$this->load->remove_package_path()

O para eliminar una ruta de paquete específica, indique la misma ruta dada previamente a **add_package_path()** para un paquete:

```
$this->load->remove_package_path(APPPATH.'third_party/foo_bar');
```

Archivos de Vista del Paquete

Por defecto, las rutas de archivos de vista de paquetes se establecen cuando se llama a **add_package_path()**. Las rutas de vistas ciclan y una vez que se encuentra una coincidencia se carga esa vista.

En este ejemplo, es posible que ocurran colisiones de nombres de vistas y posiblemente se cargue el paquete incorrecto. Para asegurarse contra esto, establezca el segundo parámetro opcional a **FALSE** al llamar a **add_package_path()**.

```
// ... guardar la ruta de la vista original, y establecerla a nuestra carpeta de
// vista del paquete Foo Bar
$orig_view_path = $this->load->ci_view_path;
$this->load->ci_view_path = APPPATH.'third_party/foo_bar/views/';

// ... código que usa los archivos de vista del paquete

// ... luego devolver la ruta de vista a la ruta de vista original de la
// aplicación
$this->load->ci_view_path = $orig_view_path;
```

Clase Migration

Las migraciones son una forma conveniente de alterar la base de datos de una forma estructurada y organizada. Podría editar fragmentos de SQL a mano, pero sería responsable de decirle a otros desarrolladores que necesitan ir y ejecutarlos. También tendría que seguir la pista de los cambios necesarios para ejecutar nuevamente en las máquinas de producción la próxima vez que se implemente.

La migración de tablas de base de datos sabe cuales migraciones ya se ejecutaron, por lo tanto todo lo que tiene que hacer es actualizar los archivos de la aplicación y llamar a `$this->migrate->current()` para resolver qué migraciones se deberían ejecutar. La versión actual se encuentra en `config/migration.php`.

Crear una Migración

Esta será la primera migración para un sitio nuevo que tiene un blog. Todas las migraciones van en la carpeta `application/migrations/` y tienen nombres tales como: `001_add_blog.php`.

```
defined('BASEPATH') OR exit('No direct script access allowed');

class Migration_Add_blog extends CI_Migration {

    public function up()
    {
        $this->dbforge->add_field(array(
            'blog_id' => array(
                'type' => 'INT',
                'constraint' => 5,
                'unsigned' => TRUE,
                'auto_increment' => TRUE
            ),
            'blog_title' => array(
                'type' => 'VARCHAR',
                'constraint' => '100',
            ),
            'blog_description' => array(
                'type' => 'TEXT',
                'null' => TRUE,
            ),
        ));
        $this->dbforge->create_table('blog');
    }

    public function down()
    {
        $this->dbforge->drop_table('blog');
    }
}
```

Entonces establecer `$config['migration_version'] = 1` en `application/config/migration.php`.

Ejemplo de Uso

En este ejemplo, se coloca algo de código simple en `application/controllers/migrate.php` para actualizar el esquema.

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

if ( ! $this->migration->current())
{
    show_error($this->migration->error_string());
}
```

Referencia de Funciones

\$this->migration->current()

La migración actual es la indicada por **\$config['migration_version']** en **application/config/migration.php**.

\$this->migration->latest()

Esto funciona muy parecido a **current()** pero, en lugar de buscar en **\$config['migration_version']**, la clase **Migration** usará la migración más reciente encontrada en el sistema de archivos.

\$this->migration->version()

Version() se puede usar para deshacer los cambios o para avanzar hacia adelante programáticamente a versiones específicas. Funciona igual que **current()**, pero ignora a **\$config['migration_version']**.

```
$this->load->library('migration');
$this->migration->version(5);
```

Preferencias de Migración

La siguiente es la lista de todas las opciones de configuración para las migraciones.

Preferencia	Valor por Defecto	Opciones	Descripción
migration_enabled	FALSE	TRUE / FALSE	Habilita o deshabilita las migraciones.
migration_version	0	Ninguna	Versión actual que su base de datos debería usar.
migration_path	APPPATH.'migrations/'	Ninguna	Ruta a su carpeta de migraciones.

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 notará a 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(codigo, 'texto')

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()

Perfilador = Profiler

Le permite habilitar/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;
```

Clase Pagination

La clase **Pagination** de CodeIgniter es muy fácil de usar y es 100% personalizable, sea dinámicamente o por medio de preferencias almacenadas.

Si no está familiarizado con el término "paginación", se refiere a enlaces que le permiten navegar de página a página, como esto:

```
<< Primera < 1 2 3 4 5 > Última >>
```

Ejemplo

Este es un ejemplo que muestra cómo crear una paginación en una de sus funciones controlador:

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

$config['base_url'] = 'http://ejemplo.com/index.php/prueba/pagina/';
$config['total_rows'] = '200';
$config['per_page'] = '20';

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

echo $this->pagination->create_links();
```

Notas:

El array **\$config** contiene sus variables de configuración. Se lo pasa a la función **\$this->pagination->initialize** según se muestra antes. Aunque hay alrededor de veinte ítems para configurar, como mínimo, se necesitan configurar los tres mostrados. Aquí hay una descripción de esos ítems:

- **base_url:** Esta es la URL completa para la clase/función controlador que contiene su paginación. En el ejemplo anterior, está apuntando a un controlador llamado "prueba" y a una función llamada "pagina". Tenga presente que puede re-rutear su URI si necesita una estructura diferente.
- **total_rows:** Este número representa la cantidad total de filas que hay en el conjunto resultado establecido para el que está creando la paginación. Normalmente este número será la cantidad total de filas que la consulta de base de datos devuelva.
- **per_page:** La cantidad de ítems que piensa mostrar por página. En el ejemplo anterior, se mostrarían 20 ítems por página.

La función **create_links()** devuelve una cadena vacía cuando no hay paginación para mostrar.

Establecer las Preferencias en un Archivo de Configuración

Si prefiere no establecer preferencias usando el método anterior, en su lugar puede ponerlas en un archivo de configuración. Simplemente cree un nuevo archivo llamado **pagination.php** y agregue el array **\$config** en ese archivo. Luego guarde el archivo en **config/pagination.php** y se lo usará automáticamente. NO necesitará usar la función **\$this->pagination->initialize** si guarda las preferencias en un archivo de configuración.

Personalizar la Paginación

La siguiente es la lista de todas las preferencias que puede pasarle a la función de inicialización para adaptar la visualización.

```
$config['uri_segment'] = 3;
```

La función de paginación determina automáticamente qué segmento de URI contiene el número de página. Si necesita algo diferente, puede especificárselo.

```
$config['num_links'] = 2;
```

La cantidad de enlaces de "dígito" que quisiera antes y después del número de la página seleccionada. Por ejemplo, el número 2 pondrá dos dígitos a ambos lados, como en el ejemplo de la página anterior.

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

Por defecto, la biblioteca de paginación asume que está usando segmentos URI y arma los enlaces algo como esto:

```
http://ejemplo.com/index.php/prueba/pagina/20
```

Si tiene `$config['enable_query_strings']` establecido a **TRUE**, sus enlaces se rescribirán automáticamente usando Query Strings. Esta opción también se puede establecer explícitamente. Estableciendo `$config['page_query_string']` a **TRUE**, se convertirá el enlace de paginación.

```
http://ejemplo.com/index.php?c=prueba&m=pagina&per_page=20
```

Advierta que "per_page" es el query string pasado por defecto, sin embargo se puede configurar usando `$config['query_string_segment'] = 'su_cadena'`.

Agregar Marcación de Cierre

Si quisiera rodear la paginación completa con algún marcado, puede hacerlo con estas dos preferencias:

```
$config['full_tag_open'] = '<p>';
```

La etiqueta de apertura colocada en la parte izquierda de todo el resultado.

```
$config['full_tag_close'] = '</p>';
```

La etiqueta de apertura colocada en la parte derecha de todo el resultado.

Personalizar el Enlace "Primera"

```
$config['first_link'] = 'Primera';
```

Texto que quisiera mostrar en el enlace "primera", en la izquierda. Si no quiere que este enlace se muestre, puede establecer su valor a FALSE.

```
$config['first_tag_open'] = '<div>';
```

La etiqueta de apertura para el enlace "primera".

```
$config['first_tag_close'] = '</div>';
```

La etiqueta de cierre para el enlace "primera".

Personalizar el Enlace "Última"

```
$config['last_link'] = 'Última';
```

Texto que quisiera mostrar en el enlace "última", en la derecha. Si no quiere que este enlace se muestre, puede establecer su valor a FALSE.

```
$config['last_tag_open'] = '<div>';
```

La etiqueta de apertura para el enlace "última".

```
$config['last_tag_close'] = '</div>';
```

La etiqueta de cierre para el enlace "última".

Personalizar el Enlace "Siguiente"

```
$config['next_link'] = '&gt;';
```

Texto que quisiera mostrar en el enlace de página "siguiente". Si no quiere que este enlace se muestre, puede establecer su valor a FALSE.

```
$config['next_tag_open'] = '<div>';
```

La etiqueta de apertura para el enlace "siguiente".

```
$config['next_tag_close'] = '</div>';
```

La etiqueta de cierre para el enlace "siguiente".

Personalizar el Enlace "Anterior"

```
$config['prev_link'] = '&lt;';
```

Texto que quisiera mostrar en el enlace de página "anterior". Si no quiere que este enlace se muestre, puede establecer su valor a FALSE.

```
$config['prev_tag_open'] = '<div>';
```

La etiqueta de apertura para el enlace "anterior".

```
$config['prev_tag_close'] = '</div>';
```

La etiqueta de cierre para el enlace "anterior".

Personalizar el Enlace "Página Actual"

```
$config['cur_tag_open'] = '<b>';
```

La etiqueta de apertura para el enlace "actual".

```
$config['cur_tag_close'] = '</b>';
```

La etiqueta de cierre para el enlace "actual".

Personalizar el Enlace "Dígito"

```
$config['num_tag_open'] = '<div>';
```

La etiqueta de apertura para el enlace "dígito".

```
$config['num_tag_close'] = '</div>';
```

La etiqueta de cierre para el enlace "dígito".

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 Parser

La Clase **Parser** le permite analizar seudo-variables contenidas dentro de los archivos de vistas. Puede analizar variables simples o pares de etiquetas variables. Si nunca usó un motor de plantillas, las seudo-variables se ven así:

```
<html>
<head>
<title>{blog_title}</title>
</head>
<body>

<h3>{blog_heading}</h3>

{blog_entries}
<h5>{title}</h5>
<p>{body}</p>
{/blog_entries}
</body>
</html>
```

Estas variables no son realmente variables de PHP, sino representaciones en texto plano que le permiten eliminar el PHP de sus plantillas (archivos de vistas).

Nota: CodeIgniter no le obliga a usar esta clase ya que usando PHP puro en sus páginas de vista les permite correr un poco más rápido. Sin embargo, algunos desarrolladores prefieren usar un motor de plantillas si trabajan con diseñadores quienes sienten alguna confusión al trabajar con PHP.

Advierta también: La Clase **Parser** no es una solución de análisis de plantillas completamente desarrollada. Hemos tratado de mantenerla magra a fin de mantener el máximo rendimiento.

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, la clase **Parser** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez cargada, el objeto de la biblioteca **Parser** estará disponible usando: **\$this->parser**.

Las siguientes funciones están disponibles en esta biblioteca:

\$this->parser->parse()

Este método acepta un nombre de plantilla y un array de datos como entrada y genera la versión analizada. Ejemplo:

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

$data = array(
    'blog_title' => 'El Titulo de mi Blog',
    'blog_heading' => 'El Encabezado de mi Blog'
);

$this->parser->parse('blog_template', $data);
```

El primer parámetro contiene el nombre del archivo de vista (en este ejemplo el archivo se llamaría **blog_template.php**), y el segundo parámetro contiene un array asociativo de datos a reemplazarse en la plantilla. En el ejemplo anterior, la plantilla contendría dos variables: **{blog_title}** y **{blog_heading}**

No hay necesidad de hacer "echo" o hacer algo con los datos devueltos por **\$this->parser->parse()**. Se lo pasa automáticamente a la Clase Output para ser enviado al navegador. Sin embargo, si quiere que los datos sean devueltos en lugar de ser enviados a la Clase Output, puede pasar **TRUE** (booleano) al tercer parámetro:

```
$string = $this->parser->parse('blog_template', $data, TRUE);
```

\$this->parser->parse_string()

Este método trabaja exactamente igual que **parse()**, solo que acepta una cadena como primer parámetro en lugar de un archivo de vista.

Pares de Variables

El ejemplo de código anterior permite que se reemplacen variables simples. ¿Qué pasa si desea que se repita un bloque entero de variables, con cada iteración conteniendo nuevos valores? Considere el ejemplo de la plantilla que mostramos al comienzo de la página:

```
<html>
<head>
<title>{blog_title}</title>
</head>
<body>

<h3>{blog_heading}</h3>

{blog_entries}
<h5>{title}</h5>
<p>{body}</p>
{/blog_entries}
</body>
</html>
```

En el código anterior advertirá un par de variables: **{blog_entries}** datos... **{/blog_entries}**. En un caso como este, la porción entera de datos entre ese par se repetiría varias veces, correspondiendo a la cantidad de filas en un resultado.

El análisis de pares de variables se realiza usando el mismo código mostrado antes para analizar variables simples, excepto que agregará un array multidimensional correspondiente a los datos de par de variables. Considere este ejemplo:

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

$data = array(
    'blog_title' => 'My Blog Title',
    'blog_heading' => 'My Blog Heading',
    'blog_entries' => array(
        array('title' => 'Title 1', 'body' => 'Body 1'),
        array('title' => 'Title 2', 'body' => 'Body 2'),
        array('title' => 'Title 3', 'body' => 'Body 3'),
        array('title' => 'Title 4', 'body' => 'Body 4'),
        array('title' => 'Title 5', 'body' => 'Body 5')
    )
);

$this->parser->parse('blog_template', $data);
```

Si sus datos de "par" vienen de los resultados de una base datos, lo que ya es un array multidimensional, simplemente puede usar la función de base de datos **result_array()**:

```
$query = $this->db->query("SELECT * FROM blog");

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

$data = array(
    'blog_title' => 'El Titulo de mi Blog',
    'blog_heading' => 'El Encabezado de mi Blog',
    'blog_entries' => $query->result_array()
);

$this->parser->parse('blog_template', $data);
```

Clase Security

La Clase **Security** contiene **métodos** que lo **ayudan** a **crear** una **aplicación segura**, **procesando** los **datos** de **entrada** para brindarle seguridad.

Filtrado XSS

CodeIgniter viene con un filtro para evitar Cross Site Scripting Hack que puede correr automáticamente tanto para filtrar **todos** los **datos POST** y **COOKIE** que se **encuentren** como por **ítem**. Por defecto, **no corre globalmente** ya que genera una pequeña sobrecarga de procesamiento, y puede que no lo necesite en todos los casos.

El filtro XSS busca técnicas comúnmente usadas para disparar Javascript u otro tipo de código que intenta secuestrar cookies o hacer cosas maliciosas. Si se encuentra algo no permitido, se lo presenta en forma segura al convertir los datos a entidades de carácter.

Nota: Esta función solamente se debería usar para tratar los datos enviados. No es algo que se debería usar para procesamiento general en tiempo de ejecución dada la sobrecarga de procesamiento que causa.

Para filtrar datos mediante el filtro XSS, use esta función:

\$this->security->xss_clean()

Aquí hay un ejemplo de uso:

```
$data = $this->security->xss_clean($data);
```

Si desea que el **filtro** funcione **automáticamente** cada vez que **encuentra** **datos POST** o **COOKIE**, puede **habilitarlo** al abrir su **archivo application/config/config.php** y configurando esto:

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

Nota: Si usa la Clase **Form_validation**, ésta también le da la posibilidad de **Filtrado XSS**.

Un segundo **parámetro opcional**, **is_image**, le **permite** a esta **función** **ser usada** para **probar imágenes** contra **potenciales ataques XSS**, útil para la **seguridad** de **subidas** de **archivos**. Cuando se establece a **TRUE** este **segundo parámetro**, en lugar de devolver una cadena alterada, la **función** **devuelve** **TRUE** si la **imagen** es **segura** y **FALSE** si contiene **información** potencialmente **maliciosa** que un **navegador** podría ejecutar.

```
if ($this->security->xss_clean($file, TRUE) === FALSE)
{
    // el archivo falló la prueba XSS
}
```

\$this->security->sanitize_filename()

Al **aceptar** **nombres** de **archivo** desde la **entrada** del **usuario**, es mejor descontaminarlos para **evitar** el **recorrido** de **directorios** y otros problemas de seguridad. Para hacerlo, use el **método** **sanitize_filename()** de la Clase **Security**. Este es un ejemplo:

```
$filename = $this->security->sanitize_filename($this->input->post('filename'));
```

Si es aceptable para la entrada del usuario incluir rutas relativas, por ejemplo **archivo/en/alguna/carpeta/aprobada.txt**, puede establecer el segundo parámetro opcional, **\$relative_path** a **TRUE**.

```
$filename = $this->security->sanitize_filename($this->input->post('filename'),  
    TRUE);
```

Cross-site request forgery (CSRF)

Puede habilitar la protección CSRF abriendo su archivo **application/config/config.php** y configurando esto:

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

Si usa el helper form la función **form_open()** insertará automáticamente un campo CSRF oculto en su formulario.

Clase Session

La Clase **Session** le permite mantener el "estado" del usuario y seguir su actividad mientras visita su sitio. La Clase **Session** almacena la información de las sesiones para cada usuario como datos serializados (y opcionalmente encriptados) en una cookie. También puede almacenar los datos de sesión en una tabla de base de datos para mayor seguridad, ya que esto permite que el ID de sesión en la cookie del usuario se compare con el ID de sesión almacenado. Por defecto solamente se guarda la cookie. Si elige usar la opción de base de datos, necesitará crear la tabla de sesión como se indica más abajo.

Nota: La Clase **Session** no usa sesiones nativas de PHP. Genera sus propios datos de sesión, ofreciendo más flexibilidad a 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.

Inicializar una Sesión

Las sesiones se ejecutarán normalmente en forma global con cada carga de página, por lo que la Clase **Session** tiene que inicializarse en los constructores de sus controladores o ser cargada automáticamente por el sistema. Para la mayor parte, la Clase **Session** se ejecutará desatendida en segundo plano, por lo que simplemente inicializar la clase provocará leer, crear y actualizar sesiones.

Para inicializar manualmente la Clase **Session** en el constructor de su controlador, use la función `$this->load->library`:

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

Una vez cargada, el objeto de la biblioteca **Session** estará disponible usando: `$this->session`.

¿Cómo Trabajan las Sesiones?

Cuando se carga una página, la Clase **Session** verificará si existen datos de sesión válidos en la cookie de sesión del usuario. Si no existen los datos de sesión (o si expiraron) se creará una nueva sesión y se guardará en la cookie. Si existe la sesión, su información se actualizará y la cookie también se actualizará. Con cada actualización, se regenerará el `session_id`.

Es importante comprender que una vez inicializada, la Clase **Session** corre automáticamente. No hay nada que se necesite hacer para que ocurra el comportamiento anterior. Como veremos a continuación, puede trabajar con los datos de sesión o incluso agregar sus propios datos a la sesión del usuario, pero el proceso de lectura, escritura y actualización de una sesión es automático.

¿Qué son los Datos de Sesión?

Una `sesión`, por lo que a CodeIgniter se refiere, es simplemente un array que contiene la siguiente información:

- El ID único de Sesión del usuario (que es una cadena estadísticamente aleatoria con una entropía muy fuerte, codificada con MD5 para darle portabilidad, y regenerada - por defecto - cada cinco minutos)
- La dirección IP del usuario
- Los datos del Agente de Usuario (los primeros 120 caracteres de la cadena de datos del navegador)
- La marca de tiempo de la "última actividad".

Los datos anteriores se almacenan en una cookie como un array serializado con el siguiente prototipo:

ARRAY SERIALIZADO

```
[array]
(
    'session_id'      => código aleatorio,
    'ip_address'     => 'cadena - dirección IP del usuario',
    'user_agent'      => 'cadena - datos del agente de usuario',
    'last_activity'   => marca de tiempo
)
```

Si tiene habilitada la opción de encriptación, el array serializado se encriptará antes de ser almacenado en la cookie, haciendo los datos altamente seguros e impermeables para que alguien los lea o altere. Aquí se puede encontrar mas información acerca de encriptación, aunque la Clase Session se encargará de inicializar y encriptar los datos automáticamente.

Nota: Las cookies de sesión solamente se actualizan cada cinco minutos por defecto para reducir la carga del procesador. Si carga repetidamente una página advertirá que la hora "última actividad" solo se actualiza si pasaron cinco minutos o más desde la última vez que se escribió la cookie. Este tiempo es configurable cambiando la línea `$config['sess_time_to_update']` en su archivo `application/config/config.php`.

session

Recuperar los Datos de Sesión

Cualquier pieza de información desde un array de sesión está disponible usando la siguiente función:

```
$this->session->userdata('item');
```

Donde `item` es el índice del array correspondiente al ítem que desea recuperar. Por ejemplo, para recuperar el ID de sesión, hará esto:

```
$session_id = $this->session->userdata('session_id');
```

Nota: La función devuelve `FALSE` (booleano) si el ítem que está tratando de acceder no existe.

Agregar Datos de Sesión Personalizados

Un aspecto útil del array de sesiones es que Ud puede agregar sus propios datos a él y se guardará en la cookie del usuario. ¿Por qué querría hacer esto? Aquí hay un ejemplo:

Digamos que un usuario en particular inicia sesión en su sitio. Una vez autenticado, Ud podría agregar su nombre de usuario y dirección de email a la cookie de sesión, haciendo que esos datos sean globales, sin tener que correr una consulta de base de datos cuando los necesite.

Agregar sus datos al array de sesión implica pasar un array contenido sus nuevos datos a esta función:

```
$this->session->set_userdata($array);
```

Donde `$array` es el array asociativo que contiene sus nuevos datos. Aquí hay un ejemplo:

```
$newdata = array(
    'username'  => 'joseperez',
    'email'     => 'joseperez@algunsitio.com',
    'logged_in' => TRUE
);
$this->session->set_userdata($newdata);
```

Si quiere **agregar** los **datos del usuario** de a **un dato por vez**, **set_userdata()** también soporta esta sintaxis.

```
$this->session->set_userdata('algun_nombre', 'algun_valor');
```

Nota: Las **cookies** solamente **pueden mantener** **4KB** de **datos**, por lo que hay que ser cuidadoso de **no exceder** la **capacidad**. En particular, el proceso de encriptación produce cadenas de datos más grandes que el original, por lo que hay que prestar atención a cómo se almacenan los datos.

Recuperar todos los Datos de Sesión

Se puede **recuperar** un **array** de **todos los userdata** como se muestra a continuación:

```
$this->session->all_userdata()
```

Y **devuelve** un **array asociativo** como el siguiente:

```
Array
(
    [session_id] => 4a5a5dca22728fb0a84364eeb405b601
    [ip_address] => 127.0.0.1
    [user_agent] => Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_7;
    [last_activity] => 1303142623
)
```

Eliminar Datos de Sesión

Así como **set_userdata()** se **puede usar** para **agregar información** a una **sesión**, **unset_userdata()** se **puede usar** para **eliminarla** pasándole la **clave de sesión**. Por ejemplo, si **quiere eliminar** 'algun_nombre' de la **información de sesión**:

```
$this->session->unset_userdata('algun_nombre');
```

A esta función también se le **puede pasar** un **array asociativo de ítems** para **eliminar**.

```
$array_items = array('username' => '', 'email' => '');
$this->session->unset_userdata($array_items);
```

Flashdata

CodeIgniter soporta "flashdata", o **datos de sesión** que **estarán disponibles** solamente para la **próxima solicitud** del **servidor**, y **se eliminan automáticamente**. Estos pueden ser muy útiles y se **usan** normalmente para **mensajes** de **estado o informativos** (por ejemplo: "registro 2 borrado").

Nota: Las **variables flash** están **precedidas** con "**flash_**" por lo tanto **evite usar este prefijo** en sus nombres de sesión.

Para agregar flashdata:

```
$this->session->set_flashdata('item', 'valor');
```

También se le puede **pasar** un **array** a **set_flashdata()**, de la misma forma que **set_userdata()**.

Para leer una variable flashdata:

```
$this->session->flashdata('item');
```

Si ve que necesita **preservar** una **variable flashdata** a través de una **solicitud adicional**, puede hacerlo usando la función **keep_flashdata()**.

```
$this->session->keep_flashdata('item');
```

Guardar Datos de Sesión en una Base de Datos

Mientras que el **array** almacenado de **datos de sesión** en la **cookie del usuario** contiene un **ID de sesión**, no hay forma de **validarlo**, a menos que almacene los **datos de sesión** en una **base de datos**. Para algunas **aplicaciones** que tienen **poca o ninguna seguridad**, puede **no necesitarse la validación** del **ID de sesión**, pero **si la aplicación necesita seguridad**, la **validación** es **obligatoria**. Si no, el **usuario** podría **restaurar** una **sesión vieja** modificando las **cookies**.

Cuando los **datos de sesión** están **disponibles** en una **base de datos**, cada vez que se **encuentra** una **sesión válida** en la **cookie del usuario**, se **ejecuta** una **consulta de base de datos** para **ver si coincide**. Si el **ID de sesión** no **coincide**, la **sesión se destruye**. Los **ID de sesión** **nunca** se **pueden actualizar**, solamente se **pueden generar** cuando **se crea** una **sesión nueva**.

Con el fin de **almacenar sesiones**, primero tiene que **crear** una **tabla** de **base de datos** para este propósito. Aquí está el prototipo básico (para MySQL) requerido por la Clase **Session**:

```
CREATE TABLE IF NOT EXISTS `ci_sessions` (
session_id varchar(40) DEFAULT '0' NOT NULL,
ip_address varchar(16) DEFAULT '0' NOT NULL,
user_agent varchar(50) NOT NULL,
last_activity int(10) unsigned DEFAULT 0 NOT NULL,
user_data text DEFAULT '' NOT NULL,
PRIMARY KEY (session_id)
);
```

Nota: La tabla se llama por defecto **ci_sessions**, pero puede llamarla como quiera siempre y cuando actualice el archivo **application/config/config.php** para que **contenga** el **nombre** que eligió. Una vez que **creó la tabla** de **base de datos** puede **habilitar la opción de base de datos** en su archivo **config.php**, de este modo:

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

Una vez **habilitada**, la clase **Session** **almacenará datos de sesión** en la **base de datos**.

Asegúrese de especificar el nombre de la tabla también en su archivo de configuración:

```
$config['sess_table_name'] = 'ci_sessions';
```

Nota: La clase **Session** tiene **incorporado** un **recolector de basura** que **borra** las **sesiones** que **caducaron**, por lo que no necesita escribir su propia rutina para hacerlo.

Destruir una Sesión

Para **borrar** la sesión actual:

```
$this->session->sess_destroy();
```

Nota: Esta función debería ser la última en llamarse. Si lo único que desea es destruir algunos ítems, pero no todos, use **unset_userdata()**.

Preferencias de Sesiones

Encontrará las siguientes preferencias relacionadas a las sesiones en su archivo **application/config/config.php**:

Preferencia	Por Defecto	Opciones	Descripción
sess_cookie_name	ci_session	Ninguno	Nombre con el que quiere guardar la sesión.
sess_expiration	7200	Ninguno	La 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_cookie	FALSE	TRUE/FALSE (booleano)	Si encriptar o no los datos de sesión.
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 SQL	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 Table

La clase **Table** provee funciones para permitirle generar tablas HTML automáticamente desde arrays o conjuntos de resultados de base de datos.

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, la clase **Table** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez cargada, el objeto de la biblioteca **Table** estará disponible usando: **\$this->table**.

Ejemplos

Aquí hay un ejemplo que muestra cómo se puede crear una tabla desde un array multidimensional. Advierte que el primer índice del array se convertirá en el encabezado de la tabla (o puede establecer sus propios encabezados usando la función **set_heading()** descripta en la Referencia de Funciones más abajo).

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

$data = array(
    array('Nombre', 'Color', 'Tamaño'),
    array('Lucas', 'Azul', 'S'),
    array('Miriam', 'Rojo', 'L'),
    array('Jose', 'Verde', 'M')
);

echo $this->table->generate($data);
```

Aquí está un ejemplo de una tabla creada desde el resultado de una consulta de base de datos. La clase **Table** generará automáticamente los encabezados basados en los nombres de tabla (o puede establecer sus propios encabezados usando la función **set_heading()** descripta en la Referencia de Funciones más abajo).

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

$query = $this->db->query("SELECT * FROM mi_tabla");
echo $this->table->generate($query);
```

Aquí hay un ejemplo que muestra cómo podría crear una tabla usando parámetros discretos:

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

$this->table->set_heading('Nombre', 'Color', 'Tamaño');

$this->table->add_row('Lucas', 'Azul', 'S');
$this->table->add_row('Miriam', 'Rojo', 'L');
$this->table->add_row('Jose', 'Verde', 'M');

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

Aquí está el mismo ejemplo, salvo que en lugar de parámetros individuales, se usan arrays:

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

$this->table->set_heading(array('Nombre', 'Color', 'Tamaño'));

$this->table->add_row(array('Lucas', 'Azul', 'S'));
$this->table->add_row(array('Miriam', 'Rojo', 'L'));
$this->table->add_row(array('Jose', 'Verde', 'M'));

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

Cambiar el Aspecto de su Tabla

La Clase **Table** le permite establecer una plantilla de tabla con la que puede especificar el diseño de su distribución. Este es el prototipo de la plantilla:

```
$tmpl = array (
    'table_open'          => '<table border="0" cellpadding="4"
cellspacing="0">',
    'heading_row_start'   => '<tr>',
    'heading_row_end'     => '</tr>',
    'heading_cell_start'  => '<th>',
    'heading_cell_end'    => '</th>',

    'row_start'           => '<tr>',
    'row_end'             => '</tr>',
    'cell_start'          => '<td>',
    'cell_end'            => '</td>',

    'row_alt_start'       => '<tr>',
    'row_alt_end'         => '</tr>',
    'cell_alt_start'      => '<td>',
    'cell_alt_end'        => '</td>',

    'table_close'         => '</table>'
);

$this->table->set_template($tmpl);
```

Nota: Advertirá que hay dos conjuntos de bloques de "filas" en la plantilla. Estos le permiten crear colores alternativos de filas o elementos de diseño que alternan con cada iteración de los datos de la fila.

NO está obligado a enviar la plantilla completa. Si solo necesita cambiar partes del diseño, simplemente puede enviar esos elementos. En el ejemplo, se cambia solamente la etiqueta de apertura de la tabla:

```
$tmpl = array ( 'table_open' => '<table border="1" cellpadding="2"
cellspacing="1" class="mytable">' );

$this->table->set_template($tmpl);
```

Referencia de Funciones

\$this->table->generate()

Devuelve una cadena que contiene la tabla generada. Acepta un parámetro opcional que puede ser un array o un objeto de resultado de base de datos.

\$this->table->set_caption()

Le permite agregar un título a la tabla.

```
$this->table->set_caption('Colores');
```

\$this->table->set_heading()

Le permite establecer el encabezado de la tabla. Puede enviar un array o parámetros discretos:

```
$this->table->set_heading('Nombre', 'Color', 'Tamaño');
```

```
$this->table->set_heading(array('Nombre', 'Color', 'Tamaño'));
```

\$this->table->add_row()

Le permite agregar una fila a su tabla. Puede enviar un array o parámetros discretos:

```
$this->table->add_row('Azul', 'Rojo', 'Verde');
```

```
$this->table->add_row(array('Azul', 'Rojo', 'Verde'));
```

Si quisiera establecer atributos de etiquetas de una celda individual, puede usar un array asociativo para esa celda. La clave asociativa 'data' define los datos de la celda. Cualquier otro par clave => valor se agregará como atributo clave='valor' a la etiqueta:

```
$cell = array('data' => 'Azul', 'class' => 'highlight', 'colspan' => 2);
$this->table->add_row($cell, 'Rojo', 'Verde');

// genera
// <td class='highlight' colspan='2'>Azul</td><td>Rojo</td><td>Verde</td>
```

\$this->table->make_columns()

Esta función toma como entrada un array unidimensional y crea un array multidimensional con una profundidad igual a la cantidad deseada de columnas. Esto permite que un array simple con varios elementos se muestre en una tabla que tiene una cantidad fija de columnas. Considere este ejemplo:

```
$list = array('uno', 'dos', 'tres', 'cuatro', 'cinco', 'seis', 'siete', 'ocho',
'nueve', 'diez', 'once', 'doce');
                                                 numero de filas
$new_list = $this->table->make_columns($list, 3);

$this->table->generate($new_list);

// Genera una tabla con este prototipo

<table border="0" cellpadding="4" cellspacing="0">
    <tr>
        <td>uno</td><td>dos</td><td>tres</td>
    </tr><tr>
        <td>cuatro</td><td>cinco</td><td>seis</td>
    </tr><tr>
        <td>siete</td><td>ocho</td><td>nueve</td>
    </tr><tr>
        <td>diez</td><td>once</td><td>doce</td>
    </tr>
</table>
```

\$this->table->set_template()

Le permite establecer su plantilla. Puede enviar una plantilla completa o parcial.

```
$tmpl = array ( 'table_open' => '<table border="1" cellpadding="2"
    cellspacing="1" class="mytable">' );

$this->table->set_template($tmpl);
```

\$this->table->set_empty()

Le permite establecer un valor por defecto para usar en cualquier celda de tabla que esté vacía. Puede, por ejemplo, establecer un espacio de no separación:

```
$this->table->set_empty(" ");
```

\$this->table->clear()

Le permite vaciar el encabezado de la tabla y los datos de fila. Si necesita mostrar varias tablas con datos diferentes, debería llamar a esta función después que cada tabla se haya generado para vaciar la información anterior de la tabla. Ejemplo:

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

$this->table->set_heading('Nombre', 'Color', 'Tamaño');
$this->table->add_row('Lucas', 'Azul', 'S');
$this->table->add_row('Miriam', 'Rojo', 'L');
$this->table->add_row('Jose', 'Verde', 'M');

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

$this->table->clear();

$this->table->set_heading('Nombre', 'Día', 'Entrega');
$this->table->add_row('Lucas', 'Miércoles', 'Expreso');
```

```
$this->table->add_row('Miriam', 'Lunes', 'Aéreo');  
$this->table->add_row('Jose', 'Sábado', 'Nocturno');  
  
echo $this->table->generate();
```

\$this->table->function

Le permite especificar funciones nativas de PHP o un objeto, array o función válidos para aplicarse a todos los datos de las celdas.

```
$this->load->library('table');  
  
$this->table->set_heading('Nombre', 'Color', 'Tamaño');  
$this->table->add_row('Lucas', '<strong>Blue</strong>', 'Small');  
  
$this->table->function = 'htmlspecialchars';  
echo $this->table->generate();
```

En el ejemplo anterior, a todos los datos de las celdas se les ejecutó la función **htmlspecialchars()** de PHP, resultando en:

```
<td>Lucas</td><td>&lt;strong&gt;Azul&lt;/strong&gt;</td><td>S</td>
```

Clase Trackback

La Clase **Trackback** provee funciones que le permiten enviar y recibir datos Trackback.

Si no está familiarizado con los Trackbacks, encontrará más información [aquí](#).

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, la Clase **Trackback** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez cargada, el objeto de la biblioteca **Trackback** estará disponible usando: **\$this->trackback**.

Enviar Trackbacks

Se puede enviar un Trackback desde cualquier función controlador usando un código similar a este ejemplo:

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

$tb_data = array(
    'ping_url'  => 'http://ejemplo.com/trackback/456',
    'url'        => 'http://www.mi-ejemplo.com/blog/entrada/123',
    'title'      => 'Título de mi Entrada',
    'excerpt'    => 'Contenido de la Entrada.',
    'blog_name'  => 'Mi Nombre de Blog',
    'charset'    => 'utf-8'
);

if ( ! $this->trackback->send($tb_data) )
{
    echo $this->trackback->display_errors();
}
else
{
    echo 'Se envió el Trackback!';
}
```

Descripción de los datos del array:

- **ping_url** - URL de su sitio a donde está enviando el Trackback. Puede enviar Trackbacks a varias URLs, separando cada URL con una coma.
- **url** - URL a SU sitio donde se puede ver la entrada del weblog.
- **title** - Título de su entrada del weblog.
- **Excerpt** - Contenido de su entrada del weblog. **Nota:** La Clase Trackback enviará automáticamente solo los primeros 500 caracteres de su entrada. También eliminará todo el HTML.
- **blog_name** - Nombre de su weblog.
- **charset** - Codificación de caracteres con la que se escribe en su weblog. Si se omite, se usará **UTF-8**.

La función que envía un Trackback devuelve **TRUE/FALSE** (booleano) en caso de éxito o falla. Si falla, se puede recuperar un mensaje de error usando:

```
$this->trackback->display_errors();
```

Recibir Trackbacks

Antes de que pueda recibir Trackbacks debe crear un weblog. Si usted no tiene un blog que no hay razón para continuar.

Recibir Trackbacks es un poco más complejo que enviarlos, solamente porque necesitará una tabla de base de datos donde almacenarlos y necesitará validar los datos del trackback entrante. Lo animamos a implementar un proceso cuidadoso de validación para protegerse contra el spam y los datos duplicados. También puede desear limitar la cantidad de Trackbacks que permite desde una IP en particular en un lapso de tiempo determinado para reducir aún más el spam. El proceso de recibir un Trackback es bastante simple; la validación es lo que se lleva el mayor esfuerzo.

Su URL de Ping

A fin de aceptar Trackbacks tiene que mostrar la URL del Trackback al lado de cada entrada de su weblog. Esta será la URL que la gente usará para enviarle sus Trackbacks (nos referimos a esto como su "URL de Ping").

Su URL de Ping tiene que apuntar a una función controlador donde esté localizado su código que recibe Trackbacks y la URL tiene que contener el número ID para cada entrada en particular, de modo que cuando se recibe el Trackback usted será capaz de asociarlo con una entrada en particular.

Por ejemplo, si su clase controlador se llama **Trackback** y la función que recibe se llama **receive**, sus URLs de Ping lucirán como esto:

```
http://ejemplo.com/index.php/trackback/receive/entry_id
```

Donde **entry_id** representa el número ID individual para cada una de las entradas.

Crear una Tabla de Trackback

Antes que pueda recibir los Trackbacks tiene que crear una tabla donde almacenarlos. Aquí hay un prototipo básico de esa tabla:

```
CREATE TABLE trackbacks (
    tb_id int(10) unsigned NOT NULL auto_increment,
    entry_id int(10) unsigned NOT NULL default 0,
    url varchar(200) NOT NULL,
    title varchar(100) NOT NULL,
    excerpt text NOT NULL,
    blog_name varchar(100) NOT NULL,
    tb_date int(10) NOT NULL,
    ip_address varchar(16) NOT NULL,
    PRIMARY KEY `tb_id`(`tb_id`),
    KEY `entry_id`(`entry_id`)
);
```

La especificación de Trackback solo requiere cuatro piezas de información a enviarse en un Trackback (url, title, excerpt, blog_name), pero para hacer más útiles los datos, hemos agregado unos pocos campos más en el esquema de la tabla anterior (date, IP address, etc.).

Procesar el Trackback

Aquí hay un ejemplo de cómo recibirá y procesará un Trackback. El siguiente código está pensado para usarse dentro de la función controlador donde espera recibir Trackbacks.

```
$this->load->library('trackback');
$this->load->database();

if ($this->uri->segment(3) == FALSE)
{
    $this->trackback->send_error("Imposible determinar el ID de la entrada");
}

if ( ! $this->trackback->receive())
{
    $this->trackback->send_error("El Trackback no contenía datos válidos");
}

$data = array(
    'tb_id'      => '',
    'entry_id'   => $this->uri->segment(3),
    'url'        => $this->trackback->data('url'),
    'title'       => $this->trackback->data('title'),
    'excerpt'     => $this->trackback->data('excerpt'),
    'blog_name'   => $this->trackback->data('blog_name'),
    'tb_date'     => time(),
    'ip_address'  => $this->input->ip_address()
);

$sql = $this->db->insert_string('trackbacks', $data);
$this->db->query($sql);

$this->trackback->send_success();
```

Notas:

El número **entry_id** se espera en el tercer segmento de su URL. Está basado en el ejemplo de URI que dimos antes:

```
http://ejemplo.com/index.php/trackback/receive/entry_id
```

Advierta que **entry_id** está en el tercer segmento URI, el cual puede recuperar usando:

```
$this->uri->segment(3);
```

En nuestro código anterior que recibe el Trackback, si falta el tercer segmento, se emitirá un error. Sin un ID de entrada válido, no hay razón para continuar.

La función **\$this->trackback->receive()** es simplemente una función de validación que mira al dato entrante y se asegura que contenga las cuatro piezas de datos requeridas (url, title, excerpt, blog_name). Devuelve **TRUE** en caso de éxito y **FALSE** en caso de falla. Si falla, se emitirá un mensaje de error.

Los datos entrantes del Trackback se pueden recuperar usando esta función:

```
$this->trackback->data('ítem');
```

Donde **ítem** representa una de estas cuatro piezas de información: url, title, excerpt, o blog_name

Si los datos del Trackback se recibieron correctamente, se emitirá un mensaje de éxito usando:

```
$this->trackback->send_success();
```

Nota: El código anterior no contiene validación de datos, lo que le recomendamos agregarlo.

Clase Typography

La clase **Typography** provee funciones que le **ayudan a formatear texto**.

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, la clase **Typography** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez cargada, el objeto de la biblioteca **Typography** estará disponible usando: **\$this->typography**.

auto_typography()

Formatea texto para que sea HTML semántica y tipográficamente correctos. Toma como entrada una cadena y la devuelve con el siguiente formato:

- Rodea párrafos dentro de `<p></p>` (busca saltos de línea dobles para identificar párrafos).
- Los saltos de línea simples se convierten en `
`, excepto aquellos que aparecen dentro de etiquetas `<pre>`.
- Los elementos a nivel de bloque, como las etiquetas `<div>`, no tienen saltos de línea automático dentro de los párrafos, pero sí su texto contenido, si contiene párrafos.
- Las comillas se convierten correctamente en comillas tipográficas enfrentadas, excepto aquellas que aparecen dentro de las etiquetas.
- Los apóstrofes se convierten en apóstrofes tipográficos.
- Los guiones dobles (sea como -- estos o como estos--otros) se convierten en estos—guiones.
- Tres puntos correlativos sea precediendo o siguiendo a una palabras, se convierten en puntos suspensivos ...
- Los espacios dobles siguiendo sentencias se convierten en ` `; que imitan el doble espacioado.

Ejemplo de Uso:

```
$string = $this->typography->auto_typography($string);
```

Parámetros

Hay un parámetro opcional que determina si el analizador sintáctico debería reducir de dos saltos de línea consecutivos a dos. Usar el booleano **TRUE** o **FALSE**.

Por defecto, el analizador sintáctico no reduce los saltos de línea. En otras palabras, si no se envían parámetros es lo mismo que hacer esto:

```
$string = $this->typography->auto_typography($string, FALSE);
```

Nota: El formateo tipográfico puede ser intensivo en el uso del procesador, particularmente si hay mucho contenido a formatear. Si elige esta función puede querer considerar cachear sus páginas.

format_characters()

Esta función es similar a la **auto_typography()** anterior, excepto que solamente hace conversión de caracteres:

- Las comillas se convierten correctamente a comillas tipográficas enfrentadas, excepto para aquellas que aparecen dentro de etiquetas.
- Los apóstrofes se convierten a apóstrofes tipográficos.
- Los guiones dobles (sea como -- estos o como estos--otros) se convierten en estos—guiones.
- Tres puntos correlativos sea precediendo o siguiendo a una palabras, se convierten en puntos suspensivos ...
- Los espacios dobles siguiendo sentencias se convierten en que imitan el doble espacioado.

Ejemplo de Uso:

```
$string = $this->typography->format_characters($string);
```

nl2br_except_pre()

Convierte caracteres de nueva línea en etiquetas **
** a menos que aparezcan dentro de etiquetas **<pre>**. Esta función es idéntica a la función **nl2br()** nativa de PHP, excepto que ignora las etiquetas **<pre>**.

Ejemplo de Uso:

```
$string = $this->typography->nl2br_except_pre($string);
```

protect_braced_quotes

Al usar la biblioteca **Typography** junto con la Clase **Parser**, a veces puede ser deseable proteger las comillas simples y dobles con llaves. Para habilitar esto, establecer la propiedad **protect_braced_quotes** de la clase a **TRUE**.

Ejemplo de Uso:

```
$this->load->library('typography');
$this->typography->protect_braced_quotes = TRUE;
```

Clase Unit_test

Las pruebas de unidad son un enfoque de la ingeniería del software en el que se escriben **pruebas** para cada **función** en su **aplicación**. Si no está familiarizado con este concepto, podría hacer una pequeña investigación googleando el tema.

La clase **Unit_test** de CodeIgniter es bastante simple, consistiendo de una función de evaluación y dos funciones de resultado. No es la intención que sea un conjunto de pruebas completamente desarrollado, sino más bien un mecanismo sencillo para evaluar el código para determinar si se está produciendo el tipo de datos correcto y el resultado.

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, la clase **Unit_test** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez cargada, el objeto **Unit_test** estará disponible usando: **\$this->unit**.

Correr Pruebas

La ejecución de una prueba consiste en suministrar una prueba y un resultado esperado a la siguiente función:

\$this->unit->run(prueba, resultado esperado, 'nombre de la prueba', 'notas')

Donde **prueba** es el resultado del código que desea probar, **resultado esperado** es el tipo de dato que espera, **nombre de la prueba** es un nombre opcional que le da a su prueba y **notas** son notas opcionales. Ejemplo:

```
$prueba = 1 + 1;  
$resultado_esperado = 2;  
$nombre_prueba = 'Suma una más uno';  
  
$this->unit->run($prueba, $resultado_esperado, $nombre_prueba);
```

El resultado esperado que suministra puede ser tanto una coincidencia literal como de tipo de datos. Aquí hay un ejemplo con un literal:

```
$this->unit->run('Foo', 'Foo');
```

Aquí hay un ejemplo de coincidencia de tipo de dato:

```
$this->unit->run('Foo', 'is_string');
```

¿Advierte el uso de "is_string" en el segundo parámetro? Le dice a la función a evaluar si la prueba está produciendo una cadena como resultado. Aquí está la lista de tipos de comparaciones permitidas:

- `is_object`
- `is_string`
- `is_bool`
- `is_true`
- `is_false`
- `is_int`
- `is_numeric`
- `is_float`
- `is_double`
- `is_array`
- `is_null`

Generación de Informes

Puede tanto mostrar los resultados después de cada prueba, como correr varias pruebas y generar un informe al final. Para mostrar un informe directamente, simplemente imprima o devuelva la función `run()`:

```
echo $this->unit->run($test, $expected_result);
```

Para correr un informe completo de todas las pruebas, haga esto:

```
echo $this->unit->report();
```

El informe se formateará en una tabla HTML. Si prefiere los datos en crudo puede recuperar un array usando:

```
echo $this->unit->result();
```

Modo Estricto

Por defecto, la clase **Unit_test** evalúa las coincidencias literales vagamente. Considere este ejemplo:

```
$this->unit->run(1, TRUE);
```

La prueba se evalúa como entero, pero el resultado esperado es booleano. PHP, sin embargo, debido a sus tipos de dato relajados evaluará el código anterior como **TRUE** usando la prueba igualdad normal:

```
if (1 == TRUE) echo 'Esto se evalúa como TRUE';
```

Si lo prefiere, puede poner la clase **Unit_test** en modo estricto, lo que comparará los tipos de dato tanto como el valor:

```
if (1 === TRUE) echo 'Esto se evalúa como FALSE';
```

Para habilitar el modo estricto, use esto:

```
$this->unit->use_strict(TRUE);
```

Habilitar/Deshabilitar la Prueba de Unidad

Si quisiera dejar algunas pruebas en sus scripts, pero no tener que correrlas a menos que lo necesite, puede deshabilitar las pruebas de unidad usando:

```
$this->unit->active(FALSE);
```

Visualización de la Prueba de Unidad

Al visualizar los resultados de la prueba de unidad, se muestran los siguientes ítems por defecto:

- Nombre de la Prueba (test_name)
- Tipo de Dato de Prueba (test_datatype)
- Tipo de Dato Esperado (res_datatype)
- Resultado (result)
- Nombre de Archivo (file)
- Número de Línea (line)
- Cualquier nota que haya ingresado para la prueba (notes)

Puede personalizar lo que de esos ítems se muestre usando `$this->unit->set_items()`. Por ejemplo, si solo quiere el nombre de la prueba y el resultado mostrado:

Personalizar las Pruebas Mostradas

```
$this->unit->set_test_items(array('test_name', 'result'));
```

Crear una Plantilla

Si quisiera que los resultados de sus pruebas se formatearan de modo diferente, entonces puede establecer su propia plantilla. Aquí hay un ejemplo de una plantilla sencilla. Advierta las seudo-variables requeridas:

```
$str = '


|        |          |
|--------|----------|
| {item} | {result} |
|--------|----------|

';
$this->unit->set_template($str);
```

Nota: Su plantilla tiene que estar declarada antes de correr el proceso de la prueba de unidad.

Clase Upload

La Clase **Upload** de CodeIgniter permite subir archivos. Se pueden establecer varias preferencias restringiendo el tipo y tamaño de los archivos.

El Proceso

Subir un archivo involucra el siguiente proceso general:

- Se muestra un formulario para subir archivos, permitiéndole al usuario elegir un archivo para subir.
- Cuando se envía el formulario, el archivo se sube al destino especificado.
- Además, el archivo se valida para asegurarse que se le permite subir en base a las preferencias establecidas.
- Una vez subido, se le muestra al usuario un mensaje de éxito.

Para mostrar este proceso, aquí hay un breve tutorial. Después encontrará información de referencia.

Crear el Formulario de Subida

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

```
<html>
<head>
<title>Formulario de Subida</title>
</head>
<body>

<?php echo $error;?>

<?php echo form_open_multipart('upload/do_upload');?>

<input type="file" name="userfile" size="20" />

<br /><br />

<input type="submit" value="upload" />

</form>

</body>
</html>
```

Advertirá que estamos usando un helper **form** para crear la etiqueta de apertura del formulario. Las subidas de archivos necesitan de un formulario "multipart", por lo tanto el helper crea la sintaxis adecuada para ello. También advertirá que usamos la variable **\$error**. Esto es para que podamos mostrar mensajes de error en caso que el usuario haga algo mal.

La Página de Éxito

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

```
<html>
<head>
<title>Formulario de Subida</title>
</head>
<body>

<h3>El archivo se transfirió correctamente!</h3>

<ul>
<?php foreach ($upload_data as $item => $value):?>
<li><?php echo $item;?>: <?php echo $value;?></li>
<?php endforeach; ?>
</ul>

<p><?php echo anchor('upload', 'Transferir otro archivo!'); ?></p>

</body>
</html>
```

El Controlador

Usando un editor de texto, crear un controlador llamado **upload.php**. Dentro suyo, colocar este código y guardarla en su carpeta **application/controllers/**:

```
<?php

class Upload extends CI_Controller {

    function __construct()
    {
        parent::__construct();
        $this->load->helper(array('form', 'url'));
    }

    function index()
    {
        $this->load->view('upload_form', array('error' => ' '));
    }

    function do_upload()
    {
        $config['upload_path'] = './uploads/';
        $config['allowed_types'] = 'gif|jpg|png';
        $config['max_size']     = '100';
        $config['max_width']   = '1024';
        $config['max_height']  = '768';

        $this->load->library('upload', $config);

        if ( ! $this->upload->do_upload())
        {
            $error = array('error' => $this->upload->display_errors());

            $this->load->view('upload_form', $error);
        }
        else
        {
            $data = array('upload_data' => $this->upload->data());
        }
    }
}
```

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

La Carpeta Upload

Necesitará una **carpeta destino** para las **imágenes subidas**. Crear una **carpeta** en la **raíz** de su **instalación** de CodeIgniter **llamada uploads** y establezca sus permisos de archivo a 777.

Pruébelo!

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

```
ejemplo.com/index.php/upload/
```

Debería ver un formulario de subida. Intente cargar un archivo de imagen (formatos jpg, gif, o png). Si la ruta en su controlador es correcta, debería funcionar.

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

Preferencia	Valor por Defecto	Opciones	Descripción
upload_path	Ninguno	Ninguno	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	Ninguno	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	Ninguno	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	Ninguno	Ancho máximo (en pixeles) que el archivo puede tener. Establecer a cero para sin límite.
max_height	0	Ninguno	Alto máximo (en pixeles) que el archivo puede tener. Establecer a cero para sin límite.
max_filename	0	Ninguno	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 guion 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 **guardé 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 siguientes 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="alguna_accion" 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 cualquier error dentro de etiquetas <p>. 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.

Ítem	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 URI

La Clase **URI** provee funciones que los ayudan a recuperar información de las cadenas **URI**. Si usa ruteo **URI**, también puede recuperar información acerca de los segmentos re-ruteados.

Nota: El sistema inicializa automáticamente a esta clase, por lo que no hay necesidad de hacerlo manualmente.

\$this->uri->segment(n)

Le permite recuperar un segmento específico. Donde **n** es el número de segmento que desea recuperar. Los segmentos se numeran de izquierda a derecha. Por ejemplo si la URL completa es esta:

 http://ejemplo.com/index.php/noticias/locale/deportes/river-descendio

Los números de segmento deberían ser estos:

1. noticias
2. locales
3. deportes
4. river-descendio

Por defecto, la función devuelve **FALSE** (booleano) si el segmento no existe. Hay un segundo parámetro opcional que le permite establecer su propio valor por defecto si falta el segmento. Por ejemplo, esto le diría a la función que devuelva el número **0** en caso de falla:

```
$product_id = $this->uri->segment(3, 0);
```

Esto ayuda a evitar tener que codificar algo así:

```
if ($this->uri->segment(3) === FALSE)
{
    $product_id = 0;
}
else
{
    $product_id = $this->uri->segment(3);
}
```

\$this->uri->rsegment(n)

Esta función es igual a la anterior, excepto que le permite recuperar un segmento específico de la URI re-ruteada en caso que esté usando la funcionalidad Ruteo **URI** de CodeIgniter.

\$this->uri->slash_segment(n)

Esta función es casi idéntica a **\$this->uri->segment()**, excepto que agrega las barras al comienzo y/o final según el valor del segundo parámetro. Si no se usa este parámetro, se agrega una barra al final. Ejemplos:

```
$this->uri->slash_segment(3);
$this->uri->slash_segment(3, 'leading'); destacado
$this->uri->slash_segment(3, 'both');
```

Devuelve:

1. segmento/
2. /segmento
3. /segmento/

\$this->uri->slash_rsegment(n)

Esta función es idéntica a la anterior, excepto que le permite agregar barras a un segmento específico desde su URI re-ruteada en el caso de que esté usando la funcionalidad Ruteo URI de CodeIgniter.

\$this->uri->uri_to_assoc(n)

Esta función le permite convertir segmentos URI en arrays asociativos de pares clave/valor. Considere esta URI:

```
index.php/usuario/buscar/nombre/jose/lugar/UK/sexo/masculino
```

Usando esta función, puede convertir la URI en un array asociativo con este prototipo:

```
[array]
(
    'nombre' => 'jose'
    'lugar' => 'UK'
    'sexo' => 'masculino'
)
```

El primer parámetro de la función le permite establecer un desplazamiento. Por defecto, se establece a 3 ya que normalmente su URI contendrá un controlador/función en los primero y segundo segmentos. Ejemplo:

```
$array = $this->uri->uri_to_assoc(3);
echo $array['nombre'];
```

El segundo parámetro le permite establecer nombres de clave por defecto, por lo que el array devuelto por la función siempre contendrá los índices esperados, aún si desaparecen de la URI. Ejemplo:

```
$default = array('nombre', 'sexo', 'lugar', 'tipo', 'orden');
$array = $this->uri->uri_to_assoc(3, $default);
```

Si la URI no contiene un valor en su defecto, un índice de array se establecerá a ese nombre, con el valor de FALSE.

Por último, si no se encuentra un valor correspondiente para una clave dada (si hay un número impar de segmentos de URI) el valor se establecerá a FALSE (booleano).

\$this->uri->ruri_to_assoc(n)

Esta función es idéntica a la anterior, excepto que crea un array asociativo usando la URI re-ruteada en el caso que esté usando la funcionalidad Ruteo URI de CodeIgniter.

\$this->uri->assoc_to_uri()

Toma como entrada un array asociativo y genera una cadena URI de él. Las claves del array se incluirán en la cadena. Ejemplo:

```
$array = array('producto' => 'zapatos', 'tamaño' => 'large', 'color' => 'rojo');  
$str = $this->uri->assoc_to_uri($array);  
// Produce: producto/zapatos/tamaño/large/color/rojo
```

\$this->uri->uri_string()

Devuelve una cadena con la URI completa. Por ejemplo, si esta es su URI completa:

```
http://ejemplo.com/index.php/noticias/locales/345
```

La función devolvería esto:

```
/noticias/locales/345
```

\$this->uri->ruri_string()

Esta función es idéntica a la anterior, excepto que devuelve la URI re-ruteada en el caso que esté usando la funcionalidad Ruteo URI de CodeIgniter.

\$this->uri->total_segments()

Devuelve la cantidad total de segmentos.

\$this->uri->total_rsegments()

Esta función es idéntica a la anterior, excepto que devuelve la cantidad total de segmentos en su URI re-ruteada en el caso que esté usando la funcionalidad Ruteo URI de CodeIgniter.

\$this->uri->segment_array()

Devuelve un array conteniendo los segmentos URI. Por ejemplo:

```
$segs = $this->uri->segment_array();  
  
foreach ($segs as $segment)  
{  
    echo $segment;  
    echo '<br />';  
}
```

\$this->uri->rsegment_array()

Esta función es igual a la anterior, salvo que devuelve el array de segmentos en su URI re-ruteada en caso de usar la funcionalidad Ruteo URI de CodeIgniter.

Clase User_agent

La Clase **User_agent** proporciona funciones que ayudan a identificar información acerca de navegadores, dispositivos móviles, o robots que visitan su sitio. Además puede obtener información del referente, así como del idioma y conjunto de caracteres soportados.

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, the Clase **User_agent** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez cargada, el objeto estará disponible usando: **\$this->agent**.

Definiciones del Agente de Usuario

Las definiciones de nombre de agente de usuario se localizan en un archivo de configuración ubicado en: **application/config/user_agents.php**. Si se necesita, puede agregar ítems a varios arrays de agentes de usuario.

Ejemplo

Cuando se inicializa la Clase **User_agent** se intentará determinar si el agente de usuario navegando su sitio es un navegador web, un dispositivo móvil o un robot. También se reunirá la información de plataforma si está disponible.

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

if ($this->agent->is_browser())
{
    $agent = $this->agent->browser() . ' ' . $this->agent->version();
}
elseif ($this->agent->is_robot())
{
    $agent = $this->agent->robot();
}
elseif ($this->agent->is_mobile())
{
    $agent = $this->agent->mobile();
}
else
{
    $agent = 'Agente de Usuario NO Identificado';
}

echo $agent;

// Información de la plataforma (Windows, Linux, Mac, etc.)
echo $this->agent->platform();
```

Referencia de Funciones

\$this->agent->is_browser()

Devuelve **TRUE/FALSE** (booleano) si el agente de usuario es un navegador conocido.

```
if ($this->agent->is_browser('Safari'))  
{  
    echo 'Está usándose Safari.';  
}  
else if ($this->agent->is_browser())  
{  
    echo 'Está usando un navegador.';  
}
```

Nota: La cadena "Safari" en este ejemplo es una clave de array en la lista de definiciones de navegadores. Puede encontrar esta lista en **application/config/user_agents.php** si quiere agregar nuevos navegadores o cambiar las cadenas.

\$this->agent->is_mobile()

Devuelve **TRUE/FALSE** (booleano) si el agente de usuario es un dispositivo móvil conocido.

```
if ($this->agent->is_mobile('iphone'))  
{  
    $this->load->view('iphone/home');  
}  
else if ($this->agent->is_mobile())  
{  
    $this->load->view('mobile/home');  
}  
else  
{  
    $this->load->view('web/home');  
}
```

\$this->agent->is_robot()

Devuelve **TRUE/FALSE** (booleano) si el agente de usuario es un robot conocido.

Nota: La biblioteca **User_agent** solamente contiene las definiciones de robot más comunes. No es una lista completa de bots. Hay cientos de ellos, por lo que buscar a cada uno podría no ser muy eficiente. Si encuentra que algunos bots que visitan normalmente su sitio están faltando en la lista, puede agregarlos a su archivo **application/config/user_agents.php**.

\$this->agent->is_referral()

Devuelve **TRUE/FALSE** (booleano) si el agente de usuario fue remitido desde otro sitio.

\$this->agent->browser()

Devuelve una cadena que contiene el nombre del navegador que está viendo su sitio.

\$this->agent->version()

Devuelve una cadena conteniendo el número de versión del navegador que está viendo su sitio.

\$this->agent->mobile()

Devuelve una cadena que contiene el nombre del dispositivo móvil que está viendo su sitio.

\$this->agent->robot()

Devuelve una cadena que contiene el nombre del robot que está viendo su sitio.

\$this->agent->platform()

Devuelve una cadena que contiene la plataforma que está viendo su sitio (Linux, Windows, OS X, etc.).

\$this->agent->referrer()

El referente, si el agente de usuario fue remitido desde otro sitio. Normalmente se prueba de la siguiente manera:

```
if ($this->agent->is_referral())
{
    echo $this->agent->referrer();
}
```

\$this->agent->agent_string()

Devuelve una cadena conteniendo el agente de usuario completo. Normalmente será algo como esto:

```
Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.0.4) Gecko/20060613
Camino/1.0.2
```

\$this->agent->accept_lang()

Le permite determinar si el agente de usuario acepta un idioma en particular. Ejemplo:

```
if ($this->agent->accept_lang('en'))
{
    echo 'Acepta Inglés!';
}
```

Nota: Normalmente esta función no es muy confiable, ya que algunos navegadores no proveen información del idioma e incluso entre aquellos que lo hacen, no siempre es exacta.

\$this->agent->accept_charset()

Le permite determinar si el agente de usuario acepta un conjunto de caracteres en particular. Ejemplo:

```
if ($this->agent->accept_charset('utf-8'))  
{  
    echo 'Su navegador soporta UTF-8!';  
}
```

Nota: Normalmente esta función no es muy confiable, ya que algunos navegadores no proveen información acerca del conjunto de caracteres e incluso entre aquellos que lo hacen, no siempre es exacta.

Clases XML-RPC y Servidor de XML-RPC

La Clase **XMLRPC** de CodeIgniter le permite enviar solicitudes a otro servidor, o configurar su propio servidor XML-RPC para recibir solicitudes.

¿Qué es XML-RPC?

Simplemente es una forma en que dos computadoras se comuniquen por Internet usando XML. Una computadora, que llamaremos **cliente**, envía una **solicitud** XML-RPC a otra computadora, que llamaremos **servidor**. Una vez que el servidor recibe y procesa la solicitud, enviará de regreso al cliente una **respuesta**.

Por ejemplo, usando la API de MetaWeblog, un Cliente XML-RPC (usualmente una herramienta de publicación de escritorio) enviará una solicitud a un Servidor XML-RPC que corre en su sitio. Esta solicitud puede ser una nueva entrada al weblog enviada por la publicación, o podría ser una solicitud para editar una entrada existente. Cuando el Servidor XML-RPC recibe la solicitud, la examinará para determinar qué clase/método debería llamar para procesar la solicitud. Una vez procesada, el servidor envía de regreso un mensaje de respuesta.

Para leer las especificaciones detalladas, puede visitar el sitio de [XML-RPC](#).

MetaWeblog - permite escribir, editar y eliminar artículos de blog usando clientes de blogging o servicios web que ofrezcan la misma funcionalidad

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, las clases **XML-RPC** y **XML-RPCS** se inicializan en su controlador usando la función **\$this->load->library**:

Para cargar la Clase XMLRPC se usará:

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

Una vez cargada, el objeto de la biblioteca **xml-rpc** estará disponible usando: **\$this->xmlrpc**.

Para cargar la Clase del Servidor XML-RPC se usará:

```
$this->load->library('xmlrpc');
$this->load->library('xmlrpcs');
```

Una vez cargada, el objeto de la biblioteca **xml-rpcs** estará disponible usando: **\$this->xmlrpcs**.

Nota: Al usar la Clase del Servidor XML-RPC tiene que cargar **AMBAS** Clases XMLRPC.

Enviar Solicitudes XML-RPC

Para enviar una solicitud a un Servidor XML-RPC tiene que especificar la siguiente información:

- La URL del servidor
- El método del servidor que desea invocar
- Los datos de la solicitud (se explica más abajo)

Este es un ejemplo que envía un ping simple de Weblogs.com para [Ping-o-Matic](#).

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

$this->xmlrpc->server('http://rpc.pingomatic.com/', 80);
$this->xmlrpc->method('weblogUpdates.ping');

$request = array('Mi Photoblog', 'http://www.mi-sitio.com/photoblog/');
$this->xmlrpc->request($request);

if ( ! $this->xmlrpc->send_request())
{
    echo $this->xmlrpc->display_error();
}
```

Explicación

El código anterior inicializa la Clase **XMLRPC**, establece la URL del servidor y el método a llamarse (weblogUpdates.ping). La solicitud (en este caso, el título y la URL de su sitio) se coloca en un array para transportarlo, y se compila usando la función **request()**. Finalmente, se envía la solicitud completa. Si el método **send_request()** devuelve **FALSE** mostraremos en mensaje de error devuelto por el Servidor XML-RPC.

Anatomía de una Solicitud

Una **solicitud** XML-RPC es simplemente los datos enviados al Servidor XML-RPC. Un **parámetro de solicitud** es cada pieza de datos en una solicitud. El ejemplo anterior tiene dos parámetros: la URL y el título de su sitio. Cuando el Servidor XML-RPC recibe su solicitud, buscará los parámetro que necesita.

Los parámetro de solicitud se tienen que poner en un array para transportarlos y cada parámetro solo puede ser de uno de los siete tipos de dato (cadenas, números, fechas, etc). Si sus parámetros son algo distinto a cadenas, tendrá que incluir los tipos de datos en el array de la solicitud.

Este es un ejemplo de un array simple con tres parámetros:

```
$request = array('Jose', 'Perez', 'www.algun-sitio.com');
$this->xmlrpc->request($request);
```

Si usa tipos de datos distintos a cadenas, o si tiene varios tipos de datos diferentes, colocará cada parámetro en su propio array, con el tipo de dato en la segunda posición:

```
$request = array (
    array('Jose', 'string'),
    array('Perez', 'string'),
    array(FALSE, 'boolean'),
    array(12345, 'int')
);
$this->xmlrpc->request($request);
```

La sección tipos de Datos más abajo tiene la lista completa de los tipos de datos.

Crear un Servidor XML-RPC

Un Servidor XML-RPC actúa como un policía de tránsito, esperando solicitudes entrantes y redireccionándolas a la función adecuada para su procesamiento.

Crear su propio Servidor XML-RPC implica inicializar la Clase de Servidor XML-RPC en su controlador donde espera que aparezca la solicitud entrante, luego configurar un array con instrucciones mapeadas para que las solicitudes

entrantes se puedan enviar a la clase y método apropiados para procesarse.

Aquí hay un ejemplo:

```
$this->load->library('xmlrpc');
$this->load->library('xmlrpcs');

$config['functions']['nuevo_mensaje'] = array('function' =>
'Mi_blog.nueva_entrada'),
$config['functions']['actualiza_mensaje'] = array('function' =>
'Mi_blog.actualiza_entrada');
$config['object'] = $this;

$this->xmlrpcs->initialize($config);
$this->xmlrpcs->serve();
```

El ejemplo anterior contiene un array que especifica dos solicitudes de método que el Servidor permite. Los métodos permitidos están en el lado izquierdo del array. Cuando cualquiera de ellos se recibe, será mapeado a la clase y método en la derecha.

La clave '**object**' es una clave especial a la que se le pasa un objeto de clase instanciado, que es necesario cuando el método que se está mapeando no es parte del super objeto CodeIgniter.

En otras palabras si un Cliente XML-RPC envía una solicitud para el método **nuevo_mensaje**, su servidor cargará la clase **Mi_blog** y llamará a la función **nueva_entrada**. Si la solicitud es para el método **actualiza_mensaje**, su servidor cargará la clase **Mi_blog** y llamará a la función **actualiza_entrada**.

Los nombres de función en el ejemplo anterior son arbitrarios. Ud decidirá cómo se deberían llamar en su servidor o si está usando APIs estandarizadas, como Blogger o MetaWeblog, usará sus nombres de función.

Hay dos claves de configuración adicionales que puede usar al inicializar la clase del servidor: **debug** se puede establecer a **TRUE** a fin de habilitar la depuración y **xss_clean** que se puede establecer a **FALSE** para evitar el envío de datos data a través de la función **xss_clean()** de la biblioteca Security.

Procesar las Solicitudes del Servidor

Cuando el Servidor XML-RPC recibe una solicitud y carga la clase/método para procesar, pasará un objeto a ese método contenido los datos enviados por el cliente.

Al usar el ejemplo anterior, si se solicita el método **nuevo_mensaje**, el servidor esperará una clase que exista con este prototipo:

```
class Mi_blog extends CI_Controller {

    function nuevo_mensaje($request)
    {
        // ...
    }
}
```

La variable **\$request** es un objeto compilado por el Servidor, que contiene los datos enviados por el Cliente XML-RPC. Usando este objeto, tendrá acceso a los parámetros de solicitud permitiéndole procesar la solicitud. Cuando lo haya hecho, enviará una respuesta de regreso al Cliente.

Abajo hay un ejemplo del mundo real que usa la API de Blogger. Uno de los métodos en la API de Blogger es **getUserInfo()**. Usando este método, un Cliente XML-RPC puede enviar al Servidor un nombre de usuario y contraseña y de regreso el Servidor devuelve información acerca de ese usuario en particular (apodo, ID de

usuario, dirección de email, etc). Aquí se muestra cómo luciría la función de procesamiento:

```
class Mi_blog extends CI_Controller {

    function getUserInfo($request)
    {
        $username = 'pirulo';
        $password = 'pirulo_pass';

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

        $parameters = $request->output_parameters();

        if ($parameters['1'] != $username AND $parameters['2'] != $password)
        {
            return $this->xmlrpc->send_error_message('100', 'Acceso inválido');
        }

        $response = array(array(
            'nickname' => array('Pirulo','string'),
            'userid' => array('99','string'),
            'url' => array('http://www.su-
                sitio.com','string'),
            'email' => array('jperez@su-sitio.com','string'),
            'lastname' => array('Perez','string'),
            'firstname' => array('Jose','string')
        ),
        'struct');

        return $this->xmlrpc->send_response($response);
    }
}
```

Notas:

La función **output_parameters()** recupera un array indexado correspondiente a los parámetros de solicitud enviados por el cliente. En el ejemplo anterior, los parámetros de salida serán el nombre del usuario y la contraseña.

Si el nombre del usuario y contraseña enviados por el cliente no fueran válidos, se devuelve un mensaje de error usando **send_error_message()**.

Si la operación fue exitosa, al cliente le será enviado un array de respuesta conteniendo la información del usuario.

Formatear una Respuesta

Similar a las *Solicitudes*, las *Respuestas* se tienen que formatear como un array. Sin embargo, a diferencia de las solicitudes, una respuesta es un **array que contiene un ítem simple**. Este ítem puede ser un array con varios arrays adicionales, pero solamente puede haber un índice primario. En otras palabras, el prototipo básico es este:

```
$response = array('Datos de respuesta', 'array');
```

Sin embargo, las respuestas normalmente contienen varias piezas de información. Para lograr esto hay que poner la respuesta en su propio array para que el array primario siga conteniendo una sola pieza de datos. Este es un ejemplo que muestra cómo se puede lograr:

```
$response = array (
    array(
        'first_name' => array('Jose', 'string'),
        'last_name' => array('Perez', 'string'),
        'member_id' => array(123435, 'int'),
        'todo_list' => array(array('limpiar la casa',
            'llamar a mamá', 'regar las plantas'), 'array'),
    ),
    'struct'
);
```

Adverta que el array anterior está formateado como una **struct**. Este es el tipo de dato más común para respuestas.

Como con las solicitudes, una respuesta tiene que ser de uno de los siete tipos de datos listados en la sección Tipos de Dato.

Enviar una Respuesta de Error

Si necesita enviar al cliente un respuesta de error, usará lo siguiente:

```
return $this->xmlrpc->send_error_message('123', 'Dato solicitado no disponible');
```

El primer parámetro es el número del error, mien

Para ayudarlo a comprender todo lo tratado hasta el momento, crearemos un par de controladores que actúan

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

```
<?php

class Xmlrpc_client extends CI_Controller {

    function index()
    {
        $this->load->helper('url');
        $server_url = site_url('xmlrpc_server');

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

        $this->xmlrpc->server($server_url, 80);
        $this->xmlrpc->method('Greetings');

        $request = array('¿Cómo le está yendo?')
        $this->xmlrpc->request($request);

        if ( ! $this->xmlrpc->send_request())
        {
            echo $this->xmlrpc->display_error();
        }
    }
}
```

```
        }
    else
    {
        echo '<pre>';
        print_r($this->xmlrpc->display_response());
        echo '</pre>';
    }
}
?>
```

Nota: En el código anterior se está usando un "helper url". Puede encontrar más información en la página Funciones Helper.

El Servidor

Usando un editor de texto, crear un controlador llamado **xmlrpc_server.php**. Dentro suyo, colocar este código y guardarla en su carpeta **application/controllers/**:

```
<?php

class Xmlrpc_server extends CI_Controller {

    function index()
    {
        $this->load->library('xmlrpc');
        $this->load->library('xmlrpcs');

        $config['functions']['Greetings'] = array('function' =>
'Xmlrpc_server/process');

        $this->xmlrpcs->initialize($config);
        $this->xmlrpcs->serve();
    }

    function process($request)
    {
        $parameters = $request->output_parameters();

        $response = array(
            array(
                'you_said' => $parameters['0'],
                'i_respond' => 'No del todo mal.'),
            'struct');

        return $this->xmlrpc->send_response($response);
    }
}
?>
```

Pruébelo!

Ahora visite su sitio usando una URL similar a esta:

```
ejemplo.com/index.php/xmlrpc_client/
```

Ahora debería ver el mensaje que envía al servidor y la respuesta que le devuelve.

El cliente que creó envía un mensaje ("¿Cómo le está yendo?") al servidor, junto con una solicitud para el método "Greetings". El Servidor recibe la solicitud y la mapea en la función **process()**, donde se devuelve una respuesta.

Usar Arrays Asociativos en un Parámetro de Solicitud

Si desea usar un array asociativo en los parámetros del método, necesitará usar un tipo de dato struct:

```
$request = array(
    array(
        // Param 0
        array(
            'nombre'=>'Jose'
        ),
        'struct'
    ),
    array(
        // Param 1
        array(
            'tamaño'=>'L',
            'forma'=>'redonda'
        ),
        'struct'
    )
);
$this->xmlrpc->request($request);
```

Puede recuperar un array asociativo al procesar la solicitud en el Servidor.

```
$parameters = $request->output_parameters();
$name = $parameters['0']['nombre'];
$size = $parameters['1']['tamaño'];
$size = $parameters['1']['forma'];
```

Referencia de Funciones XML-RPC

\$this->xmlrpc->server()

Establece la URL y el número de puerto del servidor al que se enviará una solicitud:

```
$this->xmlrpc->server('http://www.aveces.com/pings.php', 80);
```

\$this->xmlrpc->timeout()

Establece el tiempo de espera (en segundos) después del cual se cancelará la solicitud:

```
$this->xmlrpc->timeout(6);
```

\$this->xmlrpc->method()

Establece el método que se solicitará desde el servidor XML-RPC:

```
$this->xmlrpc->method( 'metodo' );
```

Donde **metodo** es el nombre del método.

\$this->xmlrpc->request()

Toma un array de datos y construye la solicitud a enviarse al servidor XML-RPC:

```
$request = array(array('Mi Photoblog', 'string'),
                  'http://www.su-sitio.com/photoblog/');
$this->xmlrpc->request($request);
```

\$this->xmlrpc->send_request()

La función de envío de la solicitud. Devuelve el booleano **TRUE** o **FALSE** basado en el éxito o fracaso, permitiendo usarlo en sentencias condicionales.

```
$this->xmlrpc->set_debug(TRUE)
```

Habilita la depuración, mostrando una variedad de información y datos de error útiles durante el desarrollo.

\$this->xmlrpc->display_error()

Devuelve un mensaje de error como una cadena si la solicitud falla por alguna razón.

```
echo $this->xmlrpc->display_error();
```

```
$this->xmlrpc->display_response()
```

Devuelve la respuesta desde el servidor remoto una vez que se recibe la solicitud. La respuesta normalmente será un array asociativo.

```
$this->xmlrpc->display_response();
```

`$this->xmlrpc->send_error_message()`

Esta función le permite enviar un mensaje de error desde su servidor al cliente. El primer parámetro es el número del error, mientras que el segundo parámetro es el mensaje de error.

```
return $this->xmlrpc->send_error_message('123',
    'Datos solicitados no disponibles');
```

\$this->xmlrpc->send_response()

Le permite enviar una respuesta desde su servidor al cliente. Con este método se tiene que enviar un array de datos válidos.

```
$response = array(
    array(
        'ferror' => array(FALSE, 'boolean'),
        'message' => "Thanks for the ping!"
    )
    'struct');

return $this->xmlrpc->send_response($response);
```

Tipos de Dato

De acuerdo a la [especificación XML-RPC](#) hay siete tipos de valores que se pueden enviar mediante XML-RPC:

- int o i4
- boolean
- string
- double
- dateTime.iso8601
- base64
- struct (contiene array de valores)
- array (contiene array de valores)

Clase Zip

La Clase **Zip** de CodeIgniter le **permite crear archivos Zip**. Los archivos se puede descargar a su escritorio o guardarlos en un directorio.

Inicializar la Clase

Como la mayoría de las clases en CodeIgniter, la clase **Zip** se inicializa en su controlador usando la función **\$this->load->library**:

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

Una vez cargado, el objeto de la biblioteca **Zip** estará disponible usando: **\$this->zip**.

Ejemplo de Uso

Este ejemplo muestra como **comprimir** un **archivo**, **guardarlo** en una **carpeta de su servidor** y **descargarlo** a su **escritorio**.

```
$name = 'mis_datos.txt';
$data = 'Una cadena de info';

$this->zip->add_data($name, $data);

// Escribe el archivo zip a una carpeta en el servidor. Llámalo "mi_backup.zip"
$this->zip->archive('/ruta/al/directorio/mi_backup.zip');

// Descarga el archivo a su escritorio. Llámalo "mi_backup.zip"
$this->zip->download('mi_backup.zip');
```

Referencia de Funciones

\$this->zip->add_data()

Le permite **agregar datos** al **archivo Zip**. El primer parámetro tiene que contener el nombre que quisiera darle al archivo y el segundo parámetro tiene que contener los datos en forma de cadena:

```
$name = 'mi_bio.txt';
$data = 'Nací en un ascensor...';

$this->zip->add_data($name, $data);
```

Se le permite varias llamadas a esta función con el fin de añadir varios archivos a su archivo comprimido. Ejemplo:

```
$name = 'mis_datos1.txt';
$data = 'Una cadena de datos';
$this->zip->add_data($name, $data);

$name = 'mis_datos2.txt';
$data = 'Otra cadena de datos';
$this->zip->add_data($name, $data);
```

O puede pasar varios archivos usando un array:

```
$data = array(
    'mis_datos1.txt' => 'Una cadena de datos',
    'mis_datos2.txt' => 'Otra cadena de datos'
);

$this->zip->add_data($data);

$this->zip->download('mi_backup.zip');
```

Si quisiera organizar sus datos comprimidos en subcarpetas, incluir la ruta como parte del archivo:

```
$name = 'personal/mi_bio.txt';
$data = 'Naci en un ascensor...';

$this->zip->add_data($name, $data);
```

El ejemplo anterior ubicará **mi_bio.txt** dentro de la carpeta llamada **personal**.

\$this->zip->add_dir()

Le permite agregar un directorio. Usualmente esta función es innecesaria ya que puede ubicar sus datos en carpetas al usar **\$this->zip->add_data()**, pero si quisiera crear una carpeta vacía puede hacerlo. Ejemplo:

```
$this->zip->add_dir('mi_carpeta'); // Crea una carpeta llamada "mi_carpeta"
```

\$this->zip->read_file()

Le permite **comprimir** un **archivo** que ya **existe** en **alguna** parte de su **servidor**. Suministre la ruta al archivo y la clase Zip que leerá y agregará el archivo:

```
$path = '/ruta/a/la/foto.jpg';

$this->zip->read_file($path);

// Descarga el archivo a su escritorio. Llámelo "mi_backup.zip"
$this->zip->download('mi_backup.zip');
```

Si quisiera que el archivo Zip mantenga la estructura de directorios del archivo en él, pase **TRUE** (booleano) en el segundo parámetro. Ejemplo:

```
$path = '/ruta/a/la/foto.jpg';

$this->zip->read_file($path, TRUE);

// Descarga el archivo a su escritorio. Llámelo "mi_backup.zip"
$this->zip->download('mi_backup.zip');
```

En el ejemplo anterior, **foto.jpg** se ubicará dentro de tres carpetas: **ruta/a/la/**

\$this->zip->read_dir()

Le permite **comprimir** una **carpeta** (y sus componentes) que ya existen en alguna parte del servidor. Suministre la ruta de archivo al directorio y la clase zip que recursivamente lo leerá y recreará como archivo Zip. Todos los archivos contenidos dentro de la ruta suministrada se codificarán, así como cualquier subcarpeta contenida dentro suyo. Ejemplo:

```
$path = '/ruta/a/su/directorio/';  
  
$this->zip->read_dir($path);  
  
// Descarga el archivo a su escritorio. Llámelo "mi_backup.zip"  
$this->zip->download('mi_backup.zip');
```

Por defecto, el archivo Zip ubicará todos los directorios listados en el primer parámetro dentro del zip. Si quiere que el árbol que precede a la carpeta destino se ignore, puede pasar **FALSE** (booleano) en el segundo parámetro. Ejemplo:

```
$path = '/ruta/a/su/directorio/';  
  
$this->zip->read_dir($path, FALSE);
```

Esto creará un ZIP con la carpeta **directorio** dentro, luego todas las subcarpetas almacenadas correctamente dentro de ella, pero no incluirá las carpetas **/ruta/a/su**.

\$this->zip->archive()

Escribe al directorio de su **servidor** un **archivo codificado** en **Zip**. Envía una ruta válida del servidor terminando en el nombre del archivo. Asegúrese que el directorio es escribible (usualmente 666 o 777 está bien). Ejemplo:

```
$this->zip->archive('/ruta/a/la/carpeta/mi_archivo.zip');  
// Crea un archivo llamado mi_archivo.zip
```

\$this->zip->download()

Provoca que el archivo Zip sea **descargado** de su **servidor**. A la función debería pasársele el nombre que quisiera que el archivo Zip tuviera. Ejemplo:

```
$this->zip->download('algun_nombre.zip');  
// El archivo se llamará "algun_nombre.zip"
```

Nota: No mostrar ningún dato en el controlador en que llamada esta función, ya que envía varios encabezados de servidor que causan la descarga ocurra y el archivo sea tratado como binario.

\$this->zip->get_zip()

Devuelve los **datos** del **archivo comprimido** con Zip. En general, usted no necesitará esta función a menos que quiera hacer algo único con los datos. Ejemplo:

```
$name = 'mi_bio.txt';
$data = 'Nací en un ascensor...';

$this->zip->add_data($name, $data);

$zip_file = $this->zip->get_zip();
```

\$this->zip->clear_data()

La clase **Zip** cachea sus datos zip para que no se necesite recompilar el archivo Zip para cada función que use. Sin embargo, si necesita crear varios Zips, cada uno con datos diferentes, puede borrar el caché entre llamadas. Ejemplo:

```
$name = 'mi_bio.txt';
$data = 'Nací en un ascensor...';

$this->zip->add_data($name, $data);
$zip_file = $this->zip->get_zip();

$this->zip->clear_data();

$name = 'foto.jpg';
$this->zip->read_file("/ruta/a/foto.jpg"); // Lee el contenido del archivo

$this->zip->download('mis_fotos.zip');
```

Driver de Almacenamiento en Caché

CodeIgniter cuenta con wrappers para las formas más populares de cachés dinámicos y rápidos. Todos, excepto el caché basados en archivos, necesitan requisitos de servidor específicos. Se lanzará una Excepción Fatal si no se cumplen los requisitos del servidor.

Ejemplo de Uso

El ejemplo siguiente cargará el driver de caché de APC y, si no se encuentra disponible en el entorno de alojamiento, recurrirá al caché basado en archivo.

```
$this->load->driver('cache', array('adapter' => 'apc', 'backup' => 'file'));  
  
if ( ! $foo = $this->cache->get('foo'))  
{  
    echo 'Guardando en el caché!<br />';  
    $foo = 'foobarbaz!';  
  
    // Guardar en el cache por 5 minutos  
    $this->cache->save('foo', $foo, 300);  
}  
  
echo $foo;
```

Referencia de Funciones

is_supported(driver['string'])

Esta función se activa automáticamente cuando se accede a los drivers mediante `$this->cache->get()`. Sin embargo, si se usa un driver individual, llame a esta función para estar seguro que el entorno de alojamiento soporta este driver.

```
if ($this->cache->apc->is_supported())  
{  
    if ($data = $this->cache->apc->get('my_cache'))  
    {  
        // hacer cosas.  
    }  
}
```

get(id['string'])

Esta función intentará obtener un ítem desde el caché. Si el ítem no existe, la función devolverá FALSE.

```
$foo = $this->cache->get('ítem_en_caché');
```

save(id['string'], data['mixed'], ttl['int'])

Esta función guardará un ítem en el caché. Si la operación falla, la función devolverá FALSE. El tercer parámetro opcional ("Time To Live") se establece por defecto a 60 segundos.

```
$this->cache->save('cache_item_id', 'datos_para_guardar');
```

delete(id['string'])

Esta función borrará un ítem específico del caché. Si la operación falla, la función devolverá **FALSE**.

```
$this->cache->delete('cache_item_id');
```

clean()

Esta función **limpiará** todo el caché. Si la operación falla, la función devolverá **FALSE**.

```
$this->cache->clean();
```

cache_info()

Esta función devolverá información del caché entero.

```
var_dump($this->cache->cache_info());
```

get_metadata(id['string'])

Esta función devolverá información detallada de un ítem específico en el caché.

```
var_dump($this->cache->get_metadata('ítem_en_caché'));
```

Drivers

Caché Alternativo de PHP (APC)

Todas las funciones mencionadas anteriormente se pueden acceder sin pasar un adaptador específico al cargador del driver, de la siguiente manera:

```
$this->load->driver('cache');
$this->cache->apc->save('foo', 'bar', 10);
```

Para más información acerca de APC, por favor ver <http://php.net/apc>

Caché basado en archivos

A diferencia del almacenamiento en caché de la Clase **Output**, el driver del caché basado en archivos permite que se almacenen en caché porciones de archivos de vistas. Use esto con cuidado y asegúrese medir el rendimiento de su aplicación, para determinar el punto en que las operaciones E/S al disco neutralicen el beneficio obtenido por el caché.

Todas las funciones mencionadas anteriormente se pueden acceder sin pasar un adaptador específico al cargador del driver, de la siguiente manera:

```
$this->load->driver('cache');
$this->cache->file->save('foo', 'bar', 10);
```

Caché Memcached

Se pueden especificar varios servidores de Memcached en el archivo de configuración **memcached.php**, ubicado en el directorio **application/config/**.

Todas las funciones mencionadas anteriormente se pueden acceder sin pasar un adaptador específico al cargador del driver, de la siguiente manera:

```
$this->load->driver('cache');
$this->cache->memcached->save('foo', 'bar', 10);
```

Para más información acerca de Memcached, por favor ver <http://php.net/memcached>

Caché ficticio

Se trata de un backend de caché que siempre 'faltará'. No almacena datos, pero le permite mantener el código del almacenamiento en caché en entornos que no soportan el caché que Ud eligió.



Referencia de Helpers

Helper Array

El archivo del Helper Array contiene funciones que lo ayudan a trabajar con arrays.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

element()

Le permite extraer un elemento desde un array. La función prueba si el índice del array está establecido y si tiene valor. Si el valor existe se devuelve. Si el valor no existe, devuelve FALSE, o lo que haya especificado como valor por defecto en el tercer parámetro. Ejemplo:

```
$array = array('color' => 'red', 'shape' => 'round', 'size' => '');

// Devuelve "red"
echo element('color', $array);

// Devuelve NULL
echo element('size', $array, NULL);
```

random_element()

Toma como entrada un array y devuelve un elemento suyo al azar. Ejemplo de uso:

```
$quotes = array(
    "Me parece que cuanto trabajo más duro, más suerte parezco tener.
        - Thomas Jefferson",
    "No te quedes en la cama, a menos que puedas hacer dinero en ella.
        - George Burns",
    "No perdimos el juego, simplemente nos quedamos sin tiempo.
        - Vince Lombardi",
    "Si todo parece bajo control, usted no va lo suficientemente rápido.
        - Mario Andretti",
    "La realidad es simplemente una ilusión, aunque una muy persistente.
        - Albert Einstein",
    "La suerte favorece a la mente preparada.
        - Louis Pasteur",
    "Hay algo mal que no está bien.
        - Ricardo Bravo"
);

echo random_element($quotes);
```

elements()

Le permite extraer una cantidad de elementos desde un array. La función prueba si cada uno de los índices del array está establecido. Si un índice no existe se establece a FALSE, o lo que sea que se haya especificado como valor por defecto a través del tercer parámetro. Ejemplo:

```
$array = array(
    'color' => 'red',
    'shape' => 'round',
    'radius' => '10',
    'diameter' => '20'
);

$my_shape = elements(array('color', 'shape', 'height'), $array);
```

Lo anterior devolverá el siguiente array:

```
$array(
    'color' => 'red',
    'shape' => 'round',
    'height' => FALSE
);
```

Puede establecer el tercer parámetro a cualquier valor por defecto que quiera:

```
$my_shape = elements(array('color', 'shape', 'height'), $array, NULL);
```

Lo anterior devolverá el siguiente array:

```
array(
    'color' => 'red',
    'shape' => 'round',
    'height' => NULL
);
```

Esto es útil al enviar el array `$_POST` a uno de sus Modelos. Esto impide que los usuarios envíen datos POST adicionales para ingresarse en las tablas:

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

$this->post_model->update(elements(array('id', 'title', 'content'), $_POST));
```

Esto asegura que solamente los campos id, title y content se enviarán para actualizarse.

Helper CAPTCHA

El archivo del Helper CAPTCHA contiene **funciones** que lo **ayudan** en la **creación** de **imágenes** **CAPTCHA**.

Cargar este Helper

Este **helper** se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

create_captcha(\$data)

Toma como **entrada** un **array** de **información** para **generar** el **CAPTCHA** y **crea** la **imagen** para sus **especificaciones**, **devolviendo** un **array asociativo** de **datos** acerca de la **imagen**.

```
[array]
(
    'image' => IMAGE TAG
    'time' => TIMESTAMP (in microtime)
    'word' => CAPTCHA WORD
)
```

image es la **etiqueta** de **imagen real**:

```

```

time es la **marca de tiempo** usada como **nombre** de la **imagen** sin la **extensión** del **archivo**. Será un número como este: 1139612155.3422

word es la **palabra** que **aparece** en la **imagen captcha**, que si no se la suministra a la función, será una cadena aleatoria.

Usar el Helper CAPTCHA

Una vez **cargado**, puede **generar** un **captcha** así:

```
$vals = array(
    'word' => 'Random word',
    'img_path' => './captcha/',
    'img_url' => 'http://example.com/captcha/',
    'font_path' => './path/to/fonts/texb.ttf',
    'img_width' => '150',
    'img_height' => 30,
    'expiration' => 7200
);

$cap = create_captcha($vals);
echo $cap['image'];
```

- La función captcha necesita la biblioteca de imagen GD.
- Solamente son obligatorios **img_path** e **img_url**.
- Si no se suministra "word", la función generará una cadena ASCII aleatoria. Podría adjuntar su propia biblioteca de palabras de la que dibujar aleatoriamente.
- Si no se especifica una ruta a una fuente True Type, se usará la espantosa fuente nativa de GD.
- La carpeta "captcha" tiene que ser escribible (666 o 777)
- **expiration** (en segundos) indica cuánto tiempo una imagen permanecerá en la carpeta captcha antes que se la borre. El valor por defecto es dos horas.

Agregar a una Base de Datos

Para que la función captcha evite que alguien pueda enviar, necesitará agregar la información devuelta por la función **create_captcha()** a la base de datos. Luego, cuando el usuario envía los datos del formulario, necesitará verificar que los datos existen en la base de datos y que no expiraron.

Este es el prototipo de la tabla:

```
CREATE TABLE captcha (
    captcha_id bigint(13) unsigned NOT NULL auto_increment,
    captcha_time int(10) unsigned NOT NULL,
    ip_address varchar(16) default '0' NOT NULL,
    word varchar(20) NOT NULL,
    PRIMARY KEY `captcha_id`(`captcha_id`),
    KEY `word`(`word`)
);
```

Este es un ejemplo de uso con una base de datos. En la página donde se mostrará el CAPTCHA, habrá algo como esto:

```
$this->load->helper('captcha');
$vals = array(
    'img_path' => './captcha/',
    'img_url' => 'http://example.com/captcha/'
);

$cap = create_captcha($vals);

$data = array(
    'captcha_time' => $cap['time'],
    'ip_address' => $this->input->ip_address(),
    'word' => $cap['word']
);

$query = $this->db->insert_string('captcha', $data);
$this->db->query($query);

echo 'Submit the word you see below:';
echo $cap['image'];
echo '<input type="text" name="captcha" value="" />';
```

Luego, en la página que acepta el envío, que tendrá algo como esto:

```
// Primero, borrar las captchas viejas
$expiration = time() - 7200; // Límite de dos horas
$this->db->query("DELETE FROM captcha WHERE captcha_time < ".$expiration);

// Luego, ver si existe un captcha:
```

```
$sql = "SELECT COUNT(*) AS count FROM captcha WHERE word = ? AND ip_address = ?  
AND captcha_time > ?";  
$binds = array($_POST['captcha'], $this->input->ip_address(), $expiration);  
$query = $this->db->query($sql, $binds);  
$row = $query->row();  
  
if ($row->count == 0)  
{  
    echo "Tiene que suministrar la palabra que aparece en la imagen";  
}
```

Helper Cookie

El archivo del Helper Cookie contiene **funciones** que lo ayudan a **trabajar** con **cookies**.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

set_cookie()

Esta **función** le da al **archivo** de **vista** una **sintaxis amigable** para **configurar** las **cookies** del **navegador**. Referirse a la Clase Input para **obtener** un **descripción** de **uso** ya que esta **función** es un **alias** para **\$this->input->set_cookie()**.

get_cookie()

Esta **función** le da al **archivo** de **vista** una **sintaxis amigable** para **obtener** las **cookies** del **navegador**. Referirse a la Clase Input para **obtener** un **descripción** de **uso** ya que esta **función** es un **alias** para **\$this->input->cookie()**.

delete_cookie()

Le **permite borrar** una **cookie**. A **menos** que **haya establecido** una **ruta personalizada** u **otros valores**, sólo se necesita el nombre de la cookie:

```
delete_cookie("name");
```

Esta función es idéntica a **set_cookie()**, salvo que no tiene los parámetros de valor y caducidad. Puede **enviar** un **array de valores** en el **primer parámetro** o **puede establecer** **parámetros discretos**.

```
delete_cookie($name, $domain, $path, $prefix);
```

Helper Date

El archivo del Helper Date contiene **funciones** que lo **ayudan** a **trabajar** con **fechas**.

Cargar este Helper

Este **helper** se **carga** usando el siguiente código:

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

Están disponibles las siguientes funciones:

now()

Devuelve la hora actual como una marca de tiempo de Unix (sea la hora local de su servidor como la GMT) basada en el valor "**time reference**" del archivo de configuración. Si no tiene la intención de establecer su tiempo de referencia maestro a GMT (lo que normalmente haría si ejecuta un sitio que le permite a cada usuario establecer la configuración de su propia zona horaria), no hay beneficio de usar esta función respecto de la función **time()** de PHP.

mdate()

Esta **función** es **idéntica** a la **date()** de PHP, salvo que le **permite** **usar** **códigos** de **fecha** de **estilo MySQL**, donde **cada letra de código** **está precedida por un signo de porcentaje**: %Y %m %d etc.

El beneficio de hacer las fechas de esta forma es que no hay que preocuparse de escapar los caracteres que no son códigos de fecha, como tendría que hacerlo normalmente con la función **date()**. Ejemplo:

```
$datestring = "Year: %Y Month: %m Day: %d - %h:%i %a";
$time = time();

echo mdate($datestring, $time);
```

Si no se incluye una marca de tiempo en el segundo parámetro, se usará la fecha/hora actual.

standard_date()

Le **permite** **generar** una **cadena** de **fecha** en uno de **varios formatos estandarizados**. Ejemplo:

```
$format = 'DATE_RFC822';
$time = time();

echo standard_date($format, $time);
```

El primer parámetro tiene que contener el formato y el segundo la fecha como marca de tiempo de Unix.

Formatos Soportados:

Constante	Descripción	Ejemplo
DATE_ATOM	Atom	2005-08-15T16:13:03+0000
DATE_COOKIE	Cookies HTTP	Dom, 14 Ago 2005 16:13:03 UTC
DATE_ISO8601	ISO-8601	2005-08-14T16:13:03+00:00
DATE_RFC822	RFC 822	Dom, 14 Ago 05 16:13:03 UTC
DATE_RFC850	RFC 850	Domingo, 14-Ago-05 16:13:03 UTC
DATE_RFC1036	RFC 1036	Domingo, 14-Ago-05 16:13:03 UTC
DATE_RFC1123	RFC 1123	Dom, 14 Ago 2005 16:13:03 UTC
DATE_RFC2822	RFC 2822	Dom, 14 Ago 2005 16:13:03 +0000
DATE_RSS	RSS	Dom, 14 Ago 2005 16:13:03 UTC
DATE_W3C	Consorcio World Wide Web	2005-08-14T16:13:03+0000

local_to_gmt()

Toma como entrada una marca de tiempo de Unix y la devuelve como GMT. Ejemplo:

```
$now = time();
$gmt = local_to_gmt($now);
```

gmt_to_local()

Toma como entrada una marca de tiempo de Unix (que hace referencia a la hora GMT), y la convierte a una marca de tiempo localizada basada en zona horaria y el horario de verano enviados. Ejemplo:

```
$timestamp = '1140153693';
$timezone = 'UM8';
$daylight_saving = TRUE;

echo gmt_to_local($timestamp, $timezone, $daylight_saving);
```

Nota: Para ver la lista de zonas horarias, vaya al final de este capítulo.

mysql_to_unix()

Toma una marca de tiempo de MySQL como entrada y la devuelve como de Unix. Ejemplo:

```
$mysql = '20061124092345';
$unix = mysql_to_unix($mysql);
```

unix_to_human()

Toma una marca de tiempo de Unix como entrada y devuelve un formato humanamente legible con este prototipo:

```
YYYY-MM-DD HH:MM:SS AM/PM
```

Es útil cuando se necesita mostrar una fecha en un campo de formulario para enviarse.

Esta hora se puede formatear con o sin segundos, y puede estar en formato Europeo o de EEUU. Si se envía solamente la marca de tiempo, devolverá la hora sin segundos formateados para EEUU. Ejemplos:

```
$now = time();  
  
echo unix_to_human($now); // Hora de EEUU sin segundos  
  
echo unix_to_human($now, TRUE, 'us'); // Hora de EEUU con segundos  
  
echo unix_to_human($now, TRUE, 'eu'); // Hora de Europea con segundos
```

human_to_unix()

Lo opuesto a la función anterior. Toma una hora "humana" como entrada y la devuelve como Unix. Esta función es útil si acepta fechas formateadas como "humanas" enviadas desde un formulario. Devuelve **FALSE** (booleano) si la cadena de la fecha pasada no está formateada como se indica más arriba. Ejemplo:

```
$now = time();  
  
$human = unix_to_human($now);  
  
$unix = human_to_unix($human);
```

timespan()

Formatea una marca de tiempo de Unix para que parezca similar a esto:

```
1 año, 10 meses, 2 semanas, 5 días, 10 horas, 16 minutos
```

El primer parámetro tiene que contener una marca de tiempo de Unix. El segundo parámetro tiene que contener una marca de tiempo mayor que la primera. Si el segundo parámetro está vacío, se usará la hora actual. El propósito más común para esta función es mostrar cuánto tiempo pasó desde algún momento en el pasado hasta ahora. Ejemplo:

```
$post_date = '1079621429';  
$now = time();  
  
echo timespan($post_date, $now);
```

Nota: El texto contenido en el menú está en el siguiente archivo de idioma:
language/<su_idioma>/date_lang.php.

days_in_month()

Devuelve la cantidad de días en un dado mes/año. Tiene en cuenta los años bisiestos. Ejemplo:

```
echo days_in_month(06, 2005);
```

Si el segundo parámetro está vacío, se usará el año actual.

timezones()

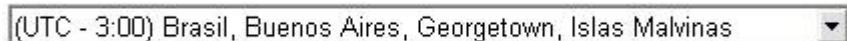
Toma una referencia de zona horaria como entrada (para la lista de zonas horarias válidas ver más abajo "Referencia de Zonas Horarias") y devuelve el desfasaje de horas respecto de UTC.

```
echo timezones('UM5');
```

Esta función es útil al usarse con **timezone_menu()**.

timezone_menu()

Genera un menú desplegable de zonas horarias como este:



Este menú es útil para sitios donde se les permite a los usuarios establecer el valor de su zona horaria local.

El primer parámetro le permite establecer el estado por defecto del menú. Por ejemplo, para establecer por defecto la hora del Pacífico deberá hacer esto:

```
echo timezone_menu('UM8');
```

Por favor lea más abajo la referencia de zonas horarias para conocer los valores de este menú.

El segundo parámetro le permite establecer un nombre de clase CSS para el menú.

Nota: El texto contenido en el menú está en el siguiente archivo de idioma:
language/<su_idioma>/date_lang.php.

Referencia de Zonas Horarias

La siguiente tabla indica cada zona horaria y su ubicación.

Zona Horaria	Ubicación
UM12	(UTC - 12:00) Enitwetok, Kwajalien
UM11	(UTC - 11:00) Nome, Isla Midway, Samoa
UM10	(UTC - 10:00) Hawaii
UM9	(UTC - 9:00) Alaska
UM8	(UTC - 8:00) Hora del Pacífico
UM7	(UTC - 7:00) Hora de la Montaña
UM6	(UTC - 6:00) Hora del Centro, Ciudad de México
UM5	(UTC - 5:00) Hora del Este, Bogotá, Lima, Quito
UM4	(UTC - 4:00) Hora del Atlántico, Caracas, La Paz
UM25	(UTC - 3:30) Nuevafoundland
UM3	(UTC - 3:00) Brasil, Buenos Aires, Georgetown, Islas Malvinas
UM2	(UTC - 2:00) Atlántico Medio, Isla Ascensión, Santa Helena

Zona Horaria	Ubicación
UM1	(UTC - 1:00) Islas Azores, Islas Cabo Verde
UTC	(UTC) Casablanca, Dublin, Edinburgo, Londres, Lisboa, Monrovia
UP1	(UTC + 1:00) Berlín, Bruselas, Copenhagen, Madrid, París, Roma
UP2	(UTC + 2:00) Kaliningrado, Sudáfrica, Varsovia
UP3	(UTC + 3:00) Bagdad, Rihad, Moscú, Nairobi
UP25	(UTC + 3:30) Teherán
UP4	(UTC + 4:00) Adu Dhabi, Baku, Muscat, Tbilisi
UP35	(UTC + 4:30) Kabul
UP5	(UTC + 5:00) Islamabad, Karachi, Tashkent
UP45	(UTC + 5:30) Bombay, Calcutta, Madrás, Nueva Delhi
UP6	(UTC + 6:00) Almaty, Colombia, Dhaka
UP7	(UTC + 7:00) Bangkok, Hanoi, Jakarta
UP8	(UTC + 8:00) Beijing, Hong Kong, Perth, Singapur, Taipei
UP9	(UTC + 9:00) Osaka, Sapporo, Seúl, Tokio, Yakutsk
UP85	(UTC + 9:30) Adelaida, Darwin
UP10	(UTC + 10:00) Melbourne, Papua Nueva Guinea, Sydney, Vladivostok
UP11	(UTC + 11:00) Magadan, Nueva Caledonia, Islas Solomon
UP12	(UTC + 12:00) Auckland, Wellington, Fiji, Isla Marshall

Helper Directory

El archivo del Helper Directory **contiene funciones** que lo **ayudan a trabajar con directorios**.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

directory_map('directorio fuente')

Esta función lee la ruta del directorio especificado en el primer parámetro y arma un array con él y todo su contenido. Ejemplo:

```
$map = directory_map('./midirectorio/');
```

Nota: Las rutas son casi siempre relativas al archivo **index.php** principal.

Las subcarpetas contenidas dentro del directorio también se mapearán. Si desea controlar la profundidad de la recursión, puede hacer eso usando el segundo parámetro (un entero). Una profundidad de **1** mapeará solamente el directorio de nivel superior:

```
$map = directory_map('./midirectorio/', 1);
```

Por defecto, los archivos ocultos no se incluirán en el array devuelto. Para anular este comportamiento, puede establecer un tercer parámetro a **TRUE** (booleano):

```
$map = directory_map('./midirectorio/', FALSE, TRUE);
```

Cada nombre de carpeta será un índice del array, mientras que sus archivos contenidos se indexarán numéricamente. Este es un ejemplo de un array típico:

```
Array
(
    [libraries] => Array
    (
        [0] => benchmark.html
        [1] => config.html
        [database] => Array
        (
            [0] => active_record.html
            [1] => binds.html
            [2] => configuration.html
            [3] => connecting.html
            [4] => examples.html
            [5] => fields.html
            [6] => index.html
            [7] => queries.html
        )
    )
)
```

```
        )
[2] => email.html
[3] => file_uploading.html
[4] => image_lib.html
[5] => input.html
[6] => language.html
[7] => loader.html
[8] => pagination.html
[9] => uri.html
)
```

Helper Download

El Helper Download le **permite descargar datos** a su **escritorio**.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

force_download('nombre_de_archivo', 'datos')

Genera **encabezados de servidor** que **fuerzan** a que los **datos** se **descarguen** a su **escritorio**. Útil con descargas de archivos. El **primer parámetro** es el **nombre** que quiere **darle** al **archivo descargado**, el **segundo parámetro** son los **datos** del **archivo**. Ejemplo:

```
$data = 'Esto es un texto cualquiera';
$name = 'mitexto.txt';

force_download($name, $data);
```

Si quiere descargar un archivo existente desde el servidor, necesitará leer el archivo a una cadena:

```
$data = file_get_contents("/path/to/photo.jpg"); // Lee el contenido del archivo
$name = 'myphoto.jpg';

force_download($name, $data);
```

Helper Email

El Helper Email provee algunas funciones que lo ayudan a trabajar con emails. Para una solución de email más robusta, vea la Clase Email de CodeIgniter.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

[valid_email\('email'\)](#)

Verifica si un email es un email correctamente formateado. Advierta que en realidad no prueba que el email se vaya a recibir, solamente que la dirección está correctamente formada.

Devuelve TRUE/FALSE

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

if (valid_email('email@somesite.com'))
{
    echo 'el email es válido';
}
else
{
    echo 'el email no es válido';
}
```

[send_email\('destinatario', 'asunto', 'mensaje'\);](#)

Envía un email usando la función [mail\(\)](#) nativa de PHP. Para una solución de email más robusta, vea la Clase Email de CodeIgniter.

Una lista de los modos posibles de fopen() usando mode

modeDescripción

'r'Apertura para sólo lectura; coloca el puntero al fichero al principio del fichero.

'r+'Apertura para lectura y escritura; coloca el puntero al fichero al principio del fichero.

'w'Apertura para sólo escritura; coloca el puntero al principio del fichero y trunca el fichero a longitud cero. Si el fichero no existe se intenta crear.

'w+'Apertura para lectura y escritura; coloca el puntero al principio de

l fichero y trunca el fichero a longitud cero. Si el fichero no existe se intenta crear.

'a'Apertura para sólo escritura; coloca el puntero del fichero al final del mismo.

Si el fichero no existe, se intenta crear. En este modo, fseek() solamente afecta a la posición de lectura; las lecturas siempre son pospuestas.

'a+'Apertura para lectura y escritura; coloca el puntero del fichero al final del mismo.

Si el fichero no existe, se intenta crear. En este modo, fseek() no tiene efecto, las escrituras siempre son pospuestas.

'x'Creación y apertura para sólo escritura; coloca el puntero del fichero al principio del mismo.

Si el fichero ya existe, la llamada a fopen() fallará devolviendo FALSE y generando un error de nivel E_WARNING.

Si el fichero no existe se intenta crear.

Esto es equivalente a especificar las banderas O_EXCL|O_CREAT para la llamada al sistema de open(2) subyacente.

'x+'Creación y apertura para lectura y escritura; de otro modo tiene el mismo comportamiento que 'x'.

'c' Abrir el fichero para sólo escritura. Si el fichero no existe, se crea. Si existe no es truncado (a diferencia de 'w'), ni la llamada a esta función falla (como en el caso con 'x'). El puntero al fichero se posiciona en el principio del fichero.

Esto puede ser útil si se desea obtener un bloqueo asistido (véase flock()) antes de intentar modificar el fichero, ya que al usar 'w' se podría truncar el fichero antes de haber obtenido el bloqueo (si se desea truncar el fichero, se puede usar ftruncate() después de solicitar el bloqueo).

'c+'Abrir el fichero para lectura y escritura; de otro modo tiene el mismo comportamiento que 'c'.

Helper File

El archivo del Helper File contiene funciones que lo ayudan a trabajar con archivos.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

read_file('ruta')

Devuelve los datos contenidos en el archivo especificado en la ruta. Ejemplo:

```
$string = read_file('./ruta/al/archivo.php');
```

La ruta puede ser relativa o absoluta. Devuelve FALSE (booleano) en caso de fallar.

Nota: La ruta es relativa a su archivo index.php principal de su sitio, NO a sus archivos de controladores o vistas. CodeIgniter usa un controlador frontal, por lo que las rutas siempre son relativas al índice principal del sitio.

Si el servidor está ejecutando una restricción open_basedir esta función puede no funcionar si Ud está tratando de acceder a un archivo antes de llamar al script.

write_file('ruta', \$data)

Escribe los datos al archivo especificado en la ruta. Si el archivo no existe, la función lo creará. Ejemplo:

```
$data = 'Algún dato del archivo';

if ( ! write_file('./ruta/al/archivo.php', $data) )
{
    echo 'No se puede escribir el archivo';
}
else
{
    echo 'Se escribió el archivo!';
}
```

Opcionalmente se puede establecer el modo de escritura mediante el tercer parámetro:

```
write_file('./ruta/al/archivo.php', $data, 'r+');
```

El modo por defecto es wb. Por favor ver la [Guía del Usuario de PHP](#) para obtener las opciones del modo.
w Apertura para sólo escritura

Nota: Para que esta función escriba los datos a un archivo sus permisos de archivo tiene que estar establecidos de la forma que se pueda escribir (666, 777, etc.). Si el archivo aún no existe, el directorio contenedor tiene que ser escribible.

Nota: La ruta es relativa a su archivo index.php principal de su sitio, NO a sus archivos de controladores o vistas. CodeIgniter usa un controlador frontal, por lo que las rutas siempre son relativas al índice principal del sitio.

delete_files('ruta')

Borra TODOS los archivos contenidos en la ruta suministrada. Ejemplo:

```
delete_files('./ruta/al/directorio/');
```

Si el segundo parámetro se establece a TRUE, también se borrará cualquier directorio contenido dentro de la ruta suministrada. Ejemplo:

```
delete_files('./ruta/al/directorio/', TRUE);
```

Nota: Los archivos tienen que ser escribibles o ser de propiedad del sistema para que se puedan borrar.

get_filenames('ruta/al/directorio/')

Toma como entrada una ruta de servidor y devuelve un array conteniendo los nombres de todos los archivos contenidos en él. La ruta de archivo se puede agregar opcionalmente a los nombres de archivos al establecer el segundo parámetro como TRUE.

get_dir_file_info('ruta/al/directorio/', \$top_level_only = TRUE)

Lee el directorio especificado y arma un array conteniendo los nombres, tamaños, fechas y permisos de los archivos. Las subcarpetas contenidas dentro de la ruta especificada solamente se leen si se lo fuerza enviando el segundo parámetro, \$top_level_only en FALSE, ya que esta puede ser una operación intensiva.

get_file_info('ruta/al/archivo', \$file_information)

Dados un archivo y una ruta, devuelve el nombre, ruta, tamaño, y fecha de modificación. El segundo parámetro le permite declarar explícitamente la información que quiere devolver. Las opciones son: name, server_path, size, date, readable, writable, executable, fileperms. Devuelve FALSE si no se encuentra el archivo.

Nota: La opción "writable" usa la función **is_writable()** de PHP que es conocida por tener problemas en el servidor web IIS. En su lugar, considere usar "fileperms", que devuelve información de la función **fileperms()** de PHP.

get_mime_by_extension('file')

Traduce una extensión de archivo a un tipo mime basado en config/mimes.php. Devuelve FALSE si no se puede determinar al tipo, o abrir el archivo de configuración mime.

```
$file = "somefile.png";
echo $file . ' tiene un tipo mime de ' . get_mime_by_extension($file);
```

Nota: Esta no es una forma exacta de determinar el tipo mime del archivo, estando aquí estrictamente por conveniencia. Por cuestiones de seguridad, no se debería usar.

symbolic_permissions(\$perms)

Toma permisos numéricos (tales como los devueltas por **fileperms()**) y devuelve la notación simbólica estándar de los permisos de archivo.

```
echo symbolic_permissions(fileperms('./index.php')); // -rw-r--r--
```

octal_permissions(\$perms)

Toma permisos numéricos (tales como los devueltas por **fileperms()**) y devuelve la notación octal de tres caracteres de los permisos de archivo.

```
echo octal_permissions(fileperms('./index.php')); // 644
```

Helper Form

El archivo del Helper Form contiene funciones que lo ayudan a trabajar con formularios.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

form_open()

Crea una etiqueta de apertura de formulario con una URL base construida a partir de sus preferencias de configuración. Opcionalmente se le permitirá agregar atributos de formulario y campos de entrada ocultos, y siempre se agregará el atributo accept-charset basado en el valor del conjunto de caracteres en su archivo de configuración.

El beneficio principal de usar esta etiqueta en lugar de codificar su propio HTML, es que le permite a su sitio ser más portable si cambian las URLs.

Aquí hay un ejemplo simple:

```
echo form_open('email/send');
```

El ejemplo anterior crearía un formulario que apunta a su URL base, más el segmento de URI "email/send":
segmento

```
<form method="post" accept-charset="utf-8"  
      action="http://example.com/index.php/email/send" />  
      controlador    segmento
```

Agregar Atributos

Los atributos se pueden agregar al pasar un array asociativo al segundo parámetro:

```
$attributes = array('class' => 'email', 'id' => 'mi_form');  
echo form_open('email/send', $attributes);
```

El ejemplo anterior crearía un formulario similar a esto:

```
<form method="post" accept-charset="utf-8"  
      action="http://example.com/index.php/email/send" class="email" id="mi_form" />
```

Agregar Campos de Entrada Ocultos

Los campos ocultos se pueden agregar al pasar un array asociativo al tercer parámetro:

```
$hidden = array('username' => 'Jose', 'member_id' => '234');  
echo form_open('email/send', '', $hidden);
```

El ejemplo anterior crearía un formulario similar a esto:

```
<form method="post" accept-charset="utf-8"
action="http://example.com/index.php/email/send">
<input type="hidden" name="username" value="Jose" />
<input type="hidden" name="member_id" value="234" />
```

form_open_multipart()

Esta función es absolutamente idéntica a **form_open()** anterior, excepto que agrega un atributo **multipart** que es necesario si quisiera usar el formulario para subir archivos.

form_hidden()

Le permite generar campos de entrada ocultos. Puede presentar una cadena "name"/"value" para crear un campo:

```
form_hidden('usuario', 'natalia_natalia');

// Produciría:

<input type="hidden" name="usuario" value="natalia_natalia" />
```

O puede presentar un array asociativo para crear varios campos:

```
$data = array(
    'name'  => 'Natalia Natalia',
    'email' => 'nn@example.com',
    'url'   => 'http://example.com'
);

echo form_hidden($data);

// Produciría:

<input type="hidden" name="name" value="Natalia Natalia" />
<input type="hidden" name="email" value="nn@example.com" />
<input type="hidden" name="url" value="http://example.com" />
```

form_input()

Le permite generar un campo de entrada de texto estándar. Mínimamente puede pasarle el nombre del campo y el valor en el primer y segundo parámetros:

```
echo form_input('usuario', 'natalia_natalia');
```

O puede pasarle un array asociativo conteniendo cualquier dato que deseé que su formulario contenga:

```
$data = array(
    'name'      => 'usuario',
    'id'        => 'usuario',
    'value'     => 'natalia_natalia',
    'maxlength' => '100',
    'size'       => '50',
    'style'     => 'width:50%',
```

```

    );
echo form_input($data);
// Produciría:

<input type="text" name="usuario" id="usuario" value="natalia_natalia"
       maxlength="100" size="50" style="width:50%" />

```

Si quisiera que su formulario contuviera algún dato adicional, como Javascript, podría pasárselo en una cadena como tercer parámetro:

```

$js = 'onClick="alguna_funcion()"' ;
echo form_input('usuario', 'natalia_natalia', $js);

```

form_password()

Esta función es idéntica en todos los aspectos a la función **form_input()** anterior salvo que permite generar un tipo "password".

form_upload()

Esta función es idéntica en todos los aspectos a la función **form_input()** anterior salvo que permite generar un tipo "file", permitiéndole usarla para subir archivos.

form_textarea()

Esta función es idéntica en todos los aspectos a la función **form_input()** anterior salvo que permite generar un tipo "textarea". **Nota:** En lugar de los atributos "maxlength" y "size" del ejemplo anterior, deberá establecer "rows" y "cols".

form_dropdown()

Le permite crear un campo de lista desplegable estándar. El primer parámetro contendrá el nombre del campo, el segundo parámetro contendrá un array asociativo de opciones y el tercer parámetro contendrá el valor que deseé que se seleccione. También puede pasársele un array de varios elementos mediante el tercer parámetro y CodeIgniter creará un selector múltiple. Ejemplo:

```

$opciones = array(
    's' => 'Camisa S',
    'm' => 'Camisa M',
    'l' => 'Camisa L',
    'xl'=> 'Camisa XL',
);
$camisas_en_venta = array('s', 'l');
echo form_dropdown('camisas', $opciones, 'l');

// Produciría:

select name="camisas">
<option value="s">Camisa S</option>
<option value="m">Camisa M</option>
<option value="l" selected="selected">Camisa L</option>

```

```

<option value="xl">Camisa XL</option>
</select>

echo form_dropdown('camisas', $opciones, $camisas_en_venta);

// Produciría:

<select name="camisas" multiple="multiple">
<option value="s" selected="selected">Camisa S</option>
<option value="m">Camisa M</option>
<option value="l" selected="selected">Camisa L</option>
<option value="xl">Camisa XL</option>
</select>

```

Si quisiera que la apertura `<select>` contuviera datos adicionales, como un atributo `id` o JavaScript, puede pasársela una cadena como cuarto parámetro:

```

$js = 'id="camisas" onChange="alguna_funcion()";';

echo form_dropdown('camisas', $opciones, 'large', $js);

```

Si el array pasado como `$opciones` es multidimensional, `form_dropdown()` producirá un `<optgroup>` con la clave del array como rótulo.

form_multiselect()

Le permite crear un campo de selección múltiple estándar. El primer parámetro contendrá el nombre del campo, el segundo parámetro contendrá un array asociativo de opciones, y el tercer parámetro contendrá el valor o valores que desea que estén seleccionados. El uso de parámetros es idéntico a usar el anterior `form_dropdown()`, salvo por supuesto que el nombre del campo tendrá que usar la sintaxis del array `POST`, por ejemplo `foo[]`.

form_fieldset()

Le permite generar los campos "fieldset"/"legend".

```

echo form_fieldset('Información de la dirección');
echo "<p>contenido del conjunto de campos aquí</p>\n";
echo form_fieldset_close();

// Produce
<fieldset>
<legend>Información de la dirección</legend>
<p>contenido del conjunto de campos aquí</p>
</fieldset>

```

Similar a otras funciones, puede enviar un array asociativo en el segundo parámetro si prefiere establecer atributos adicionales.

```

$attributes = array('id' => 'address_info', 'class' => 'address_info');
echo form_fieldset('Información de la dirección', $attributes);
echo "<p>contenido del conjunto de campos aquí</p>\n";
echo form_fieldset_close();

// Produce
<fieldset id="address_info" class="address_info">

```

```
<legend>Información de la dirección</legend>
<p>contenido del conjunto de campos aquí</p>
</fieldset>
```

form_fieldset_close()

Produce una etiqueta `</fieldset>` de cierre. La única ventaja de usar esta función es que le permite pasar datos a ella, los que se agregarán debajo de la etiqueta. Por ejemplo:

```
$string = "</div></div>";

echo fieldset_close($string);

// Produciría:
</fieldset>
</div></div>
```

form_checkbox()

Le permite generar un campo de casilla de verificación. Ejemplo sencillo:

```
echo form_checkbox('newsletter', 'accept', TRUE);

// Produciría:

<input type="checkbox" name="newsletter" value="accept" checked="checked" />
```

El tercer parámetro contiene un booleano **TRUE/FALSE** para determinar si la caja se debería marcar o no.

Similar a las otras funciones de formulario en este helper, también puede pasarle a la función un array de atributos:

```
$data = array(
    'name'         => 'newsletter',
    'id'           => 'newsletter',
    'value'        => 'accept',
    'checked'      => TRUE,
    'style'        => 'margin:10px',
);

echo form_checkbox($data);

// Produciría:



```

Como con otras funciones, si quisiera que la etiqueta contenga datos adicionales, como JavaScript, puede pasarle una cadena como cuarto parámetro:

```
$js = 'onClick="alguna_funcion()"';

echo form_checkbox('newsletter', 'accept', TRUE, $js)
```

form_radio()

Esta función es idéntica en todos los aspectos a la función **form_checkbox()** anterior, salvo que establece un tipo "radio".

form_submit()

Le permite generar un botón enviar estándar. Ejemplo sencillo:

```
echo form_submit('mi_submit', 'Enviar mensaje');

// Produciría:
<input type="submit" name="mi_submit" value="Enviar mensaje" />
```

Similar a otras funciones, puede pasar un array asociativo en el primer parámetro si prefiere establecer sus propios atributos. El tercer parámetro le permite agregar datos adicionales al formulario, como JavaScript.

form_label()

Le permite generar un <label>. Ejemplo sencillo:

```
echo form_label('Cual es su nombre?', 'username');

// Produciría:
<label for="username">Cual es su nombre?</label>
```

Similar a otras funciones, puede pasar un array asociativo en el tercer parámetro si prefiere establecer sus propios atributos.

```
$attributes = array(
    'class' => 'mi_clase',
    'style' => 'color: #000;',
);
echo form_label('Cual es su nombre?', 'username', $attributes);

// Produciría:
<label for="username" class="mi_clase" style="color: #000;">Cual es su nombre?
</label>
```

form_reset()

Le permite generar un botón estándar de reset. El uso es idéntico a **form_submit()**.

form_button()

Le permite generar un elemento de botón estándar. Mínimamente puede pasar el nombre del botón y contenido en el primer y segundo parámetros:

```
echo form_button('nombre', 'contenido');

// Produciría
<button name="nombre" type="button">contenido</button>
```

O puede **pasarle un array asociativo** contenido cualquier dato que deseé que su formulario contenga:

```
$data = array(
    'name' => 'button',
    'id' => 'button',
    'value' => 'true',
    'type' => 'reset',
    'content' => 'Reset'
);

echo form_button($data);

// Produciría:
<button name="button" id="button" value="true" type="reset">Reset</button>
```

Si desea que su formulario contenga algunos datos adicionales, como JavaScript, puede pasarlo como una cadena en el tercer parámetro:

```
$js = 'onClick="alguna_funcion()"';

echo form_button('mi_boton', 'Clic aquí', $js);
```

form_close()

Produce una etiqueta **</form>** de cierre. La única ventaja de usar esta función es que le permite pasar datos a ella, los que se agregarán debajo de la etiqueta. Por ejemplo:

```
$string = "</div></div>";

echo form_close($string);

// Produciría:
</form>
</div></div>
```

form_prep()

Le permite **usar** con **seguridad HTML** y **caracteres tales** como **comillas dentro de elementos de formulario** sin **romper el formulario**. Considere este ejemplo:

```
$string = 'Aquí hay una cadena que contiene texto entre "comillas".';

<input type="text" name="mi_formulario" value="$string" />
```

Como la cadena anterior contiene un conjunto de comillas, ese causará que el formulario se rompa. La función **form_prep** convierte HTML para que se pueda usar de forma segura:

```
<input type="text" name="mi_formulario"
      value="<?php echo form_prep($string); ?>" />
```

Nota: Si usa alguna de las funciones del Helper Form listadas en esta página los valores del formulario se prepararán automáticamente, por lo que no hay necesidad de llamar a esta función. Úsela solamente si está

creando sus propios elementos de formulario.

set_value()

Le permite establecer el valor de una entrada de formulario o de un textarea. Tiene que suministrar el nombre del campo mediante el primer parámetro de la función. El segundo parámetro (opcional) le permite establecer un valor por defecto para el formulario. Ejemplo:

```
<input type="text" name="cantidad"
      value=<?php echo set_value('cantidad', '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 elemento 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 elemento y el tercer parámetro (opcional) le permite establecer un elemento por defecto (usar el booleano TRUE/FALSE). Ejemplo:

```
<select name="mi_select">
<option value="uno" <?php echo set_select('mi_select', 'uno', TRUE); ?> >
    Uno</option>
<option value="dos" <?php echo set_select('mi_select', 'dos'); ?> >
    Dos</option>
<option value="tres" <?php echo set_select('mi_select', 'tres'); ?> >
    Tres</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 elemento por defecto (usar el booleano TRUE/FALSE). Ejemplo:

```
<input type="checkbox" name="mi_casilla" value="1"
      <?php echo set_checkbox('mi_casilla', '1'); ?> />
<input type="checkbox" name="mi_casilla" value="2"
      <?php echo set_checkbox('mi_casilla', '2'); ?> />
```

set_radio()

Le permite mostrar botones de radio en el estado en que se enviaron. Esta función es idéntica a la función set_checkbox() anterior.

```
<input type="radio" name="mi_radio" value="1"
      <?php echo set_radio('mi_radio', '1', TRUE); ?> />
<input type="radio" name="mi_radio" value="2"
      <?php echo set_radio('mi_radio', '2'); ?> />
```

Helper HTML

El archivo del Helper HTML **contiene funciones** que lo **ayudan** a **trabajar** con **HTML**.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

br() *deprecated*

Genera etiquetas de corte de línea (`
`) basada en el número que le suministre. Ejemplo:

```
echo br(3);
```

Lo anterior produciría: `

`

heading()

Le permite crear etiquetas `<h1>` de HTML. El primer parámetro contendrá el dato y el segundo el tamaño del encabezado. Ejemplo:

```
echo heading('Hola!', 3);
```

Lo anterior produciría: `<h3>Hola!</h3>`

img()

Le permite **crear** **etiquetas** `` de **HTML**. El primer parámetro **contiene la ruta** de la **imagen**. Ejemplo:

```
echo img('images/picture.jpg');
// produce 
```

Hay un **segundo parámetro opcional** que es un valor **TRUE/FALSE** que **especifica** si **src** **debería tener** la **página** **especificada** por **\$config['index_page']** agregada a la **dirección** que **crea**. Presumiblemente, esto **sería** si **estuviera usando** un **controlador de medios**.

```
echo img('images/picture.jpg', TRUE);
// produce 
```

Adicionalmente, se puede **pasar** un **array asociativo** a la **función img()** para obtener control total de todos los atributos y valores. Si **no se provee** un **atributo alt**, CodeIgniter generará una cadena vacía.

```
$image_properties = array(
    'src' => 'images/picture.jpg',
    'alt' => 'Yo, demostrando como comer cuatro porciones de pizza de una vez',
    'class' => 'post_images',
```

```

'width' => '200',
'height' => '200',
'title' => 'Esa fue una gran noche',
'rel' => 'lightbox',
);



```

link_tag()

Le permite crear etiquetas `<link />` de HTML. Esto es útil para enlaces de hojas de estilo, así como para otros enlaces. Los parámetros son `href`, con `rel` opcional, `type`, `title`, `media` e `index_page`. `index_page` es un valor `TRUE/FALSE` que especifica si el `href` debería tener la página especificada por `$config['index_page']` agregada a la dirección que crea.

```

echo link_tag('css/mis_estilos.css');

// produce <link href="http://site.com/css/mis_estilos.css" rel="stylesheet"
// type="text/css" />

```

Otros ejemplos:

```

echo link_tag('favicon.ico', 'shortcut icon', 'image/ico');

// <link href="http://site.com/favicon.ico" rel="shortcut icon"
// type="image/ico" />

echo link_tag('feed', 'alternate', 'application/rss+xml', 'My RSS Feed');

// <link href="http://site.com/feed" rel="alternate" type="application/rss+xml"
// title="My RSS Feed" />

```

Adicionalmente, se puede pasar un array asociativo a la función `link()` para obtener control total de todos los atributos y valores.

```

$link = array(
    'href' => 'css/printer.css',
    'rel' => 'stylesheet',
    'type' => 'text/css',
    'media' => 'print'
);

echo link_tag($link);

// <link href="http://site.com/css/printer.css" rel="stylesheet" type="text/css"
// media="print" />

```

nbs()

Genera espacios sin cortes (` `) basados en el número suministrado. Ejemplo:

```
echo nbs(3);
```

Lo anterior produciría: ` `

ol() y ul()

Le permite generar listas ordenadas y sin orden de HTML a partir de un array simple o multidimensional. Ejemplo:

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

$list = array(
    'red',
    'blue',
    'green',
    'yellow'
);

$attributes = array(
    'class' => 'boldlist',
    'id'     => 'mylist'
);

echo ul($list, $attributes);
```

El código anterior producirá esto:

```
<ul class="boldlist" id="mylist">
    <li>red</li>
    <li>blue</li>
    <li>green</li>
    <li>yellow</li>
</ul>
```

Este es un ejemplo más complejo, usando un array multidimensional:

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

$attributes = array(
    'class' => 'boldlist',
    'id'     => 'mylist'
);

$list = array(
    'colors' => array(
        'red',
        'blue',
        'green'
    ),
    'shapes' => array(
        'round',
        'square',
        'circles' => array(
            'circle1',
            'circle2',
            'circle3'
        )
    )
);
```

```
        'ellipse',
        'oval',
        'sphere'
    )
),
'moods' => array(
    'happy',
    'upset' => array(
        'defeated' => array(
            'dejected',
            'disheartened',
            'depressed'
        ),
        'annoyed',
        'cross',
        'angry'
    )
)
);
echo ul($list, $attributes);
```

El código anterior producirá esto:

```
<ul class="boldlist" id="mylist">
<li>colors
  <ul>
    <li>red</li>
    <li>blue</li>
    <li>green</li>
  </ul>
</li>
<li>shapes
  <ul>
    <li>round</li>
    <li>square</li>
    <li>circles
      <ul>
        <li>ellipse</li>
        <li>oval</li>
        <li>sphere</li>
      </ul>
    </li>
  </ul>
</li>
<li>moods
  <ul>
    <li>happy</li>
    <li>upset
      <ul>
        <li>defeated
          <ul>
            <li>dejected</li>
            <li>disheartened</li>
            <li>depressed</li>
          </ul>
        </li>
      </ul>
    </li>
    <li>annoyed</li>
    <li>cross</li>
  </ul>
</li>
```

```

        <li>angry</li>
    </ul>
</li>
</ul>
</li>
</ul>
```

meta()

Le ayuda a generar meta tags. A la función le puede pasar cadenas, arrays simples o multidimensionales. Ejemplos:

```

echo meta('description', 'Mi bonito sitio');
// Genera: <meta name="description" content="Mi bonito sitio" />

echo meta('Content-type', 'text/html; charset=utf-8', 'equiv');
// Advierta el tercer parámetro. Puede ser "equiv" o "name"
// Genera: <meta http-equiv="Content-type" content="text/html; charset=utf-8" />

echo meta(array('name' => 'robots', 'content' => 'no-cache'));
// Genera: <meta name="robots" content="no-cache" />

$meta = array(
    array('name' => 'robots', 'content' => 'no-cache'),
    array('name' => 'description', 'content' => 'Mi bonito sitio'),
    array('name' => 'keywords', 'content' => 'amor, pasión, intriga,
decepción'),
    array('name' => 'robots', 'content' => 'no-cache'),
    array('name' => 'Content-type', 'content' => 'text/html; charset=utf-8',
'type' => 'equiv')
);

echo meta($meta);
// Genera:
// <meta name="robots" content="no-cache" />
// <meta name="description" content="Mi bonito sitio" />
// <meta name="keywords" content="amor, pasión, intriga, decepción" />
// <meta name="robots" content="no-cache" />
// <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
```

doctype()

Le ayuda a generar declaraciones de tipo de documento, o DTD. Por defecto se usa XHTML 1.0 Strict, pero hay varios tipos de documentos disponibles.

```

echo doctype();
// <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
// "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

echo doctype('html4-trans');
// <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
// "http://www.w3.org/TR/html4/strict.dtd">
```

La siguiente es la lista de las opciones de doctype. Estas son configurables y se extraen de **application/config/doctypes.php**.

Doctype	Opción	Resultado
XHTML 1.1	doctype('xhtml11')	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">
XHTML 1.0 Strict	doctype('xhtml1-strict')	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
XHTML 1.0 Transitional	doctype('xhtml1-trans')	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
XHTML 1.0 Frameset	doctype('xhtml1-frame')	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
HTML 5	doctype('html5')	<!DOCTYPE html>
HTML 4 Strict	doctype('html4-strict')	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
HTML 4 Transitional	doctype('html4-trans')	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
HTML 4 Frameset	doctype('html4-frame')	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">

Helper Inflector

El archivo del Helper Inflector contiene **funciones** que le **permiten cambiar** las **palabras** a plural, singular, camel case, etc.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

singular()

Cambia a singular una palabra en plural. Ejemplo:

```
$word = "dogs";
echo singular($word); // Devuelve "dog"
```

plural()

Cambia a plural una palabra en singular. Ejemplo:

```
$word = "dog";
echo plural($word); // Devuelve "dogs"
```

Para forzar a una palabra a que termine en "es" use **TRUE** como segundo argumento.

```
$word = "pass";
echo plural($word, TRUE); // Devuelve "passes"
```

camelize()

Cambia una cadena de palabras separadas por espacios o guiones de subrayado a camel case. Ejemplo:

```
$word = "my_dog_spot";
echo camelize($word); // Devuelve "myDogSpot"
```

underscore()

Cambia los espacios que separan las palabras por guiones de subrayado. Ejemplo:

```
$word = "my dog spot";
echo underscore($word); // Devuelve "my_dog_spot"
```

humanize()

Cambia los guiones de subrayado que unen las palabras por espacios e inicia cada palabra con mayúsculas.
Ejemplo:

```
$word = "my_dog_spot";
echo humanize($word); // Devuelve "My Dog Spot"
```

Helper Language

El archivo del Helper Language contiene funciones que lo ayudan a trabajar con archivos de idiomas.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

lang('language line', 'element id')

Esta función devuelve una línea de texto desde un archivo de idioma cargado con una sintaxis simplificada que puede ser más deseable para ver archivos que llamar a **\$this->lang->line()**. El segundo parámetro opcional generará también una etiqueta **<label>**. Ejemplo:

```
echo lang('language_key', 'form_item_id');
// se convierte en <label for="form_item_id">language_key</label>
```

Helper Number

El archivo del Helper Number contiene funciones que lo ayudan a trabajar con datos numéricos.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

byte_format()

Formatea un número como bytes, basado en el tamaño, y agrega el sufijo adecuado. Ejemplos:

```
echo byte_format(456); // Devuelve 456 Bytes
echo byte_format(4567); // Devuelve 4.5 KB
echo byte_format(45678); // Devuelve 44.6 KB
echo byte_format(456789); // Devuelve 447.8 KB
echo byte_format(3456789); // Devuelve 3.3 MB
echo byte_format(12345678912345); // Devuelve 1.8 GB
echo byte_format(123456789123456789); // Devuelve 11,228.3 TB
```

Un segundo parámetro opcional le permite establecer la precisión del resultado.

```
echo byte_format(45678, 2); // Devuelve 44.61 KB
```

Nota: El texto generado por esta función está en el siguiente archivo de idioma: **language/<su_idioma>/number_lang.php**.

Helper Path

El archivo del Helper Path contiene **funciones** que le permiten trabajar con **rutas** de **archivos en el servidor**.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

set realpath()

Verifica si la ruta existe. Esta función devolverá una ruta de servidor sin enlaces simbólicos o estructuras de directorio relativas. Un segundo parámetro causará que se dispare un error si no se puede resolver la ruta.

```
$directorio = '/etc/passwd';
echo set realpath($directorio);
// Devuelve "/etc/passwd"

$directorio_inexistente = '/ruta/a/ninguna_parte';
echo set realpath($directorio_inexistente, TRUE);
// Devuelve un error, porque no se pudo resolver la ruta

echo set realpath($directorio_inexistente, FALSE);
// Devuelve "/ruta/a/ninguna_parte"
```

Helper Security

El archivo del Helper Security contiene **funciones** relacionadas con la **seguridad**.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

xss_clean()

Provee **filtrado** contra **piratería** de **scripts** de **sitio cruzado**. Esta **función** es un **alias a otra** en la Clase Input. Se puede encontrar más **información** ahí.

sanitize_filename()

Provee **protección** contra el **recorrido** de **directorios**. Esta función es un **alias a otra** en la Clase Security. Se puede encontrar más información ahí.

do_hash()

Le permite **crear hashes** SHA1 o MD5 **adecuados** para **encriptar contraseñas**. Creará SHA1 por defecto. Ejemplos:

```
$str = do_hash($str); // SHA1  
$str = do_hash($str, 'md5'); // MD5
```

Nota: Esta función se llamaba anteriormente **dohash()**, la cual se marcó como obsoleta en favor de **do_hash()**.

strip_image_tags()

Esta es una función de seguridad que quitará las etiquetas de imagen de una cadena. Deja la URL de la imagen como texto plano.

```
$string = strip_image_tags($string);
```

encode_php_tags()

Esta es una función de seguridad que convierte etiquetas PHP a entidades. **Nota:** La función de filtrado XSS hace esto automáticamente.

```
$string = encode_php_tags($string);
```

Helper Smiley

El archivo del Helper Smiley contiene funciones que le permiten administrar smileys (emoticones).

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Introducción

El Helper Smiley tiene un procesador que toma smileys de texto plano como :-) y los convierte en imágenes como 😊

También le permite mostrar un conjunto de imágenes de smiley que al cliquearlas se insertarán en un campo de formulario. Por ejemplo, si tiene un blog que permite comentarios del usuario, puede mostrar los smileys cerca del formulario de comentarios. Los usuarios pueden hacer clic sobre un smiley y con la ayuda de algún JavaScript se ubicarán dentro del campo de formulario.

Tutorial de Smileys Cliqueables

Aquí hay un ejemplo que muestra como tiene que crear un conjunto de smileys cliqueables cerca de un campo de formulario. Este ejemplo necesita que primero descargue e instale las imágenes de smiley y luego cree un controlador y la vista según se describe.

Importante: Antes que comience, por favor descargue las imágenes de smiley y póngalas en un lugar accesible públicamente en el servidor. Este helper asume también que Ud tiene un array de reemplazo de smileys en **application/config/smileys.php**.

El Controlador

En la carpeta **application/controllers/** cree un archivo llamado **smileys.php** y ubique en él el código que está más abajo.

Importante: Cambiar la URL en la función **get_clickable_smileys()** para que apunte a la carpeta **smiley**.

Advertirás que además del Smiley estamos usando la Clase Table.

```
class Smileys extends CI_Controller {

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

    function index()
    {
        $this->load->helper('smiley');
        $this->load->library('table');

        $image_array = get_clickable_smileys('http://example.com/images/smileys/');
```

```

        'comments');
$col_array = $this->table->make_columns($image_array, 8);
$data['smiley_table'] = $this->table->generate($col_array);
$this->load->view('smiley_view', $data);
}
?>

```

En su carpeta **application/views/**, crear un archivo llamado **smiley_view.php** y ubicar este código en él:

```

<html>
<head>
<title>Smileys</title>

<?php echo smiley_js(); ?>

</head>
<body>

<form name="blog">
<textarea name="comments" id="comments" cols="40" rows="4"></textarea>
</form>

<p>Hacer clic para insertar un smiley!</p>

<?php echo $smiley_table; ?>

</body>
</html>

```

Cuando haya creado el controlador y la vista, cárguelos visitando <http://www.example.com/index.php/smileys/>

Alias de Campo

Al hacer cambios en la vista puede ser inconveniente tener un id de campo en el controlador. Para evitar esto, puede darle a sus enlaces de smiley un nombre genérico que se vinculará a un id específico en la vista.

```
$image_array = get_smiley_links("http://example.com/images/smileys/",
"comment_textarea_alias");
```

Para mapear el alias al id del campo, páselos juntos en la función **smiley_js()**:

```
$image_array = smiley_js("comment_textarea_alias", "comments");
```

Referencia de Funciones

get_clickable_smileys()

Devuelve un array que contiene las imágenes smiley envueltas en un enlace. Tiene que suministrar la URL a la carpeta de smileys y un id de campo o un alias de campo.

```
$image_array = get_smiley_links("http://example.com/images/smileys/", "comment");
```

Nota: El uso de esta función sin el segundo parámetro junto con **js_insert_smiley** fue marcado como obsoleto.

smiley_js()

Genera el JavaScript que permite que las imágenes se puedan cliquear e insertar en un campo de formulario. Si suministra un alias en lugar de un id al generar los enlaces del smiley, necesita pasar el alias y el correspondiente id de formulario a la función. Esta función está diseñada para ubicarse dentro del área <head> de la página web.

```
<?php echo smiley_js(); ?>
```

Nota: Esta función reemplaza a **js_insert_smiley**, la que fue marcada como obsoleta.

parse_smileys()

Toma una cadena de texto como entrada y reemplaza cualquier smiley de texto plano con la imagen equivalente. El primer parámetro debe contener la cadena, el segundo tiene que contener la URL a la carpeta de smileys:

```
$str = 'Estos son algunos smileys: :-) ;-)';
$str = parse_smileys($str, "http://example.com/images/smileys/");
echo $str;
```

Helper String

El archivo del Helper String contiene **funciones** que lo ayudan a **trabajar** con **cadenas** de **caracteres**.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

random_string()

Genera una **cadena aleatoria** basada en el tipo y longitud especificadas. **Útil** para **crear contraseñas** o **generar hashes aleatorios**.

El **primer parámetro** especifica el **tipo** de **cadena**, el **segundo parámetro** especifica la **longitud**. Están disponibles las siguientes alternativas:

alpha, alnum, numeric, nozero, unique, md5, encrypt y sha1

- **alpha:** Una cadena con letras mayúsculas y minúsculas solamente.
- **alnum:** Una cadena alfanumérica con caracteres en mayúsculas y minúsculas.
- **numeric:** Una cadena numérica.
- **nozero:** Una cadena numérica sin ceros.
- **unique:** Una cadena encriptada con MD5 y **uniqid()**. **Nota:** el parámetro longitud no está disponible para este tipo. Devuelve una cadena de 32 caracteres de longitud (fija).
- **sha1:** Un número aleatorio basado en **do_hash()** del Helper Security.

Ejemplo de Uso:

```
echo random_string('alnum', 16);
```

increment_string()

Incrementa una cadena al agregarle un número o incrementando el número. Util para crear "copias" de un archivo o duplicar el contenido de una base de datos que tiene ID o título únicos.

Ejemplo de uso:

```
echo increment_string('file', '_'); // "file_1"  
echo increment_string('file', '-', 2); // "file-2"  
echo increment_string('file-4'); // "file-5"
```

alternator()

Permite que dos o más elementos alternen entre sí al ciclar dentro de un bucle. Ejemplo:

```
for ($i = 0; $i < 10; $i++)  
{  
    echo alternator('cadena 1', 'cadena 2');  
}
```

Puede agregar tantos parámetros como quiera y con cada iteración del bucle se devolverá el siguiente elemento.

```
for ($i = 0; $i < 10; $i++)
{
    echo alternator('uno', 'dos', 'tres', 'cuatro', 'cinco');
```

Nota: Para usar varias llamadas separadas a esta función simplemente llame la función sin argumentos para reinicializar.

repeater()

Genera copias repetidas de los datos a enviar. Ejemplo:

```
$string = "\n";
echo repeater($string, 30);
```

Lo anterior generaría 30 nuevas líneas.

reduce_double_slashes()

Convierte las barras dobles en una cadena a barras simples, salvo para aquellas encontradas en http://. Ejemplo:

```
$string = "http://example.com//index.php";
echo reduce_double_slashes($string); // devuelve: "http://example.com/index.php"
```

trim_slashes()

Quita las barras de inicio y fin de una cadena. Ejemplo:

```
$string = "/esto/aquello/lootro/";
echo trim_slashes($string); // devuelve: esto/aquello/lootro
```

reduce_multiples()

Reduce múltiples instancias de un carácter particular que ocurren uno detrás de otro. Ejemplo:

```
$string = "Batman, Robin,, Gatúbela, Batichica";
$string = reduce_multiples($string,",");
//devuelve: "Batman, Robin, Gatúbela, Batichica"
```

La función acepta los siguiente parámetros:

```
reduce_multiples(string: texto en el que buscar, string: carácter a reducir,
                  boolean: si quitar el carácter del inicio y fin de la cadena)
```

El primer parámetro contiene la cadena de la que quiere reducir las redundancias. El segundo parámetro contiene el carácter que quiere reducir. El tercer parámetro es **FALSE** por defecto; si se establece a **TRUE** quitará las ocurrencias del carácter al inicio y fin de la cadena. Ejemplo:

```
$string = ",Batman, Robin,, Gatúbela, Batichica,";  
$string = reduce_multiples($string, ",", TRUE);  
//devuelve: "Batman, Robin, Gatúbela, Batichica"
```

quotes_to_entities()

Convierte las comillas simples y dobles en una cadena a las entidades HTML correspondientes. Ejemplo:

```
$string = "Joe's \"dinner\"";  
$string = quotes_to_entities($string); //devuelve: "Joe's &quot;dinner&quot;"
```

strip_quotes()

Quita las comillas simples y dobles de una cadena. Ejemplo:

```
$string = "Joe's \"dinner\"";  
$string = strip_quotes($string); //devuelve: "Joes dinner"
```

Helper Text

El archivo del Helper Text contiene **funciones** que lo **ayudan** a **trabajar** con **texto**.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

word_limiter()

Trunca la **cadena** a la **cantidad** de **palabras especificada**. Ejemplo:

```
$string = "Esta es una hermosa cadena de texto que tiene once palabras.";  
  
$string = word_limiter($string, 4);  
  
// Devuelve: Esta es una hermosa...
```

El tercer parámetro es un sufijo opcional que se agrega a la cadena. Por defecto se le agregan puntos suspensivos.

character_limiter()

Trunca la **cadena** a la **cantidad** de **caracteres especificada**. Mantiene la **integridad** de las **palabras** por lo que la **cantidad de caracteres** puede ser **ligeramente mayor o menor** a la **especificada**. Ejemplo:

```
$string = "Esta es una hermosa cadena de texto que tiene once palabras.";  
  
$string = character_limiter($string, 20);  
  
// Devuelve: Esta es una hermosa cadena...
```

El tercer parámetro es un sufijo opcional que se agrega a la cadena. Por defecto se le agregan puntos suspensivos.

ascii_to_entities()

Convierte **valores ASCII** en **entidades de carácter**, **incluyendo caracteres** de MS Word y ASCII alto que pueden causar problemas se usan en una página web de forma que se puedan mostrar consistentemente independientemente de la configuración del navegador o almacenar fiablemente en una base de datos. Hay alguna dependencia con el conjunto de caracteres del servidor, por lo que puede no ser 100% confiable en todos los casos, pero para la mayor parte se deberían identificar correctamente los caracteres fuera del rango normal (como los carácter acentuados). Ejemplo:

```
$string = ascii_to_entities($string);
```

entities_to_ascii()

Esta función hace lo opuesto a la anterior; cambia las entidades de carácter a ASCII.

convert_accented_characters()

Transcribe caracteres altos de ASCII a sus equivalentes bajos de ASCII. Útil cuando se necesitan caracteres no ingleses para usarse donde se usan solamente caracteres estándar ASCII seguros, por ejemplo, en URLs.

```
$string = convert_accented_characters($string);
```

Esta función usa el archivo de configuración **application/config/foreign_chars.php** para definir el array de la transcripción.

word_censor()

Le permite censurar palabras dentro de una cadena de texto. El primer parámetro contendrá la cadena original. El segundo contendrá el array de palabras a rechazar. El tercer parámetro puede contener valores de reemplazo para las palabras. Si no se especifica, se reemplazan por signos numerales: #####. Ejemplo:

```
$disallowed = array('damier', 'joraca', 'locu', 'tate');  
  
$string = word_censor($string, $disallowed, 'Beep!');
```

highlight_code()

Colorea una cadena de código (PHP, HTML, etc.). Ejemplo:

```
$string = highlight_code($string);
```

Esta función usa la función **highlight_string()** de PHP, por lo que los colores usados son los especificados en el archivo **php.ini**.

highlight_phrase()

Resaltará una frase dentro de una cadena de texto. El primer parámetro contendrá la cadena original y el segundo contendrá la frase que se desea resaltar. Los parámetros tercero y cuarto contendrán las etiquetas HTML de apertura y cierre que desearía que rodearan a la frase. Ejemplo:

```
$string = "Este es un hermoso texto acerca de nada en particular.";  
  
$string = highlight_phrase($string, "hermoso texto",  
    '<span style="color:#990000">', '</span>');
```

El texto anterior devuelve:

Este es un **hermoso texto** acerca de nada en particular.

word_wrap()

Introduce saltos de línea automáticos al texto para la cantidad especificada de caracteres, mientras mantiene completas las palabras. Ejemplo:

```
$string = "Esta es una cadena de texto simple que nos ayudará a ejemplificar esta función.";  
  
echo word_wrap($string, 24);  
  
// Produciría:  
  
Esta es una cadena de  
texto simple que nos  
ayudará a ejemplificar  
esta función.
```

ellipsize()

Esta función quitará las etiquetas de la cadena, la dividirá en una longitud máxima definida y le insertará puntos suspensivos.

El primer parámetro es la cadena a la cual aplicar la función, el segundo parámetro es la cantidad de caracteres en la cadena final. El tercer parámetro indica donde en la cadena tienen que aparecer los puntos suspensivos, de 0 a 1, izquierda a derecha. Por ejemplo: un valor de 1 ubicará los puntos suspensivos a la derecha de la cadena, .5 en el medio y 0 a la izquierda.

Un parámetro opcional es el tipo de puntos suspensivos. Por defecto, se insertará puntos suspensivos (...).

```
$str = 'esta_cadena_es demasiado_larga_y_puede_romper_mi_diseño';  
  
echo ellipsize($str, 32, .5);  
  
// Produce:  
// esta_cadena_es_de...romper_mi_diseño
```

Helper Typography

El archivo del Helper Typography contiene funciones que lo ayudan a **formatear texto** de forma semántica.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

auto_typography()

Formatea texto para que sea HTML semántica y tipográficamente correcto. Para mayor información, vea la Clase Typography.

Ejemplo de Uso:

```
$string = auto_typography($string);
```

Nota: El formateo tipográfico puede usar intensamente al procesador, particularmente si tiene un montón de contenido para formatearse. Si elige usar esta función, debería considerar cachear sus páginas.

nl2br_except_pre()

Convierte caracteres de nueva línea a etiquetas **
** a menos que éstas aparezcan dentro de etiquetas **<pre>**. Esta función es idéntica a la **nl2br()** nativa de PHP, salvo que ignora las etiquetas **<pre>**.

Ejemplo de Uso:

```
$string = nl2br_except_pre($string);
```

Helper URL

The Helper URL file contiene funciones que lo ayudan a trabajar con URLs.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

site url()

Devuelve la URL del sitio, según se especifica en el archivo de configuración. El archivo **index.php** (o el que haya establecido como el **index_page** del sitio en el archivo de configuración) se agregará a la URL, así como cualquier segmento de URI que pase a la función.

Lo animamos a utilizar esta función en cualquier momento que usted necesite para generar una URL local, de forma que las páginas se vuelven más portables en caso de que su URL cambie.

Los **segmentos** se puede **pasar** opcionalmente a la **función** como una **cadena** o un **array**. Aquí hay un ejemplo de cadena:

```
echo site url("news/local/123");
```

El ejemplo anterior devolvería algo como: <http://example.com/index.php/news/local/123>.
site url

Aquí hay un ejemplo de segmentos pasados como un array:

```
$segments = array('news', 'local', '123');

echo site_url($segments);
```

base_url()

Devuelve la URI base del sitio, según se especifica en el archivo de configuración. Ejemplo:

```
echo base_url();
```

Devuelve la URL completa (segmentos incluidos) de la página que se está viendo en este momento.

uri_string()

Devuelve los segmentos URI de cualquier página que contiene esta función. Por ejemplo, si la URL fuera esta:

<http://some-site.com/blog/comments/123>

La función devolvería:

```
/blog/comments/123
```

index_page()

Devuelve la página "index" del sitio, según se especifica en el archivo de configuración. Ejemplo:

```
echo index_page();
```

anchor()

Crea un enlace ancla de HTML estándar basado en la URL de su sitio:

```
<a href="http://example.com">Clic aquí</a>
```

La etiqueta tiene tres parámetros opcionales:

```
anchor('segmentos uri', 'texto', 'atributos')
```

El primer parámetro puede contener cualquier segmento que deseé agregar a la URL. Como con la función `site_url()` anterior, los segmentos pueden estar en una cadena o un array.

Nota: Si está armando enlaces que son internos a la aplicación, no incluya la URL base (`http://...`). Esto se agregará automáticamente desde la información especificada en el archivo de configuración. Incluir solamente los segmentos de URI que deseé agregar a la URL.

El segundo segmento es el texto que querría que diga el enlace. Si lo deja en blanco se usará la misma URL.

El tercer parámetro puede contener una lista de atributos que quisiera agregar al enlace. Los atributos pueden ser una cadena simple o un array asociativo.

Aquí hay algunos ejemplos:

```
echo anchor('news/local/123', 'Mis noticias', 'title="Titulares"');
```

Produciría: `Mis noticias`

```
echo anchor('news/local/123', 'Mis noticias',
array('title' => 'Buenas noticias!'));
```

Produciría: `Mis noticias`

anchor_popup()

Casi idéntica a la función `anchor()` salvo que abre la URL en una ventana nueva. Puede especificar atributos de ventana JavaScript en el tercer parámetro para controlar cómo se abre la ventana. Si el tercer parámetro no está establecido, simplemente abrirá una nueva ventana con los valores del navegador. Aquí hay un ejemplo con atributos:

```
$atts = array(
    'width'      => '800',
    'height'     => '600',
    'scrollbars' => 'yes',
    'status'      => 'yes',
    'resizable'   => 'yes',
    'screenx'    => '0',
    'screeny'    => '0'
);

echo anchor_popup('news/local/123', 'Clic aquí!', $atts);
```

Nota: Los **atributos anteriores** son los **valores por defecto** de la **función**, por lo que solamente **necesita establecer** aquellos que **son distintos a los que necesita**. Si quiere que la **función** use **todos sus valores por defecto**, simplemente **pásele un array vacío** en el **tercer parámetro**:

```
echo anchor_popup('news/local/123', 'Clic aquí!', array());
```

mailto()

Crea un **enlace de email** estándar de **HTML**. Ejemplo de uso:

```
echo mailto('yo@misitio.com', 'Haga clic aquí para contactarme');
```

Como con la **función anchor()** anterior, puede establecer los **atributos** usando el **tercer parámetro**.

safe_mailto()

Es idéntica a la **función anterior**, salvo que **escribe una versión ofuscada de la etiqueta mailto** usando **números ordinales escritos con JavaScript** para ayudar a evitar que la **dirección de email** sea **recolectada por bots de spam**.

auto_link()

Convierte **automáticamente URLs** y **direcciones de email** contenidas en **cadenas en enlaces**. Ejemplo:

```
$string = auto_link($string);
```

El **segundo parámetro determina** si las **URLs o emails** se **convierten ambos, uno o el otro**. Por defecto el **parámetro es ambos (both)** si no se especifica nada. Los enlaces de email se codifican usando **safe_mailto()** como se muestra antes.

Convierte solo URLs:

```
$string = auto_link($string, 'url');
```

Convierte solo direcciones de email:

```
$string = auto_link($string, 'email');
```

El **tercer parámetro determina** si los **enlaces** se **muestran** en una **ventana nueva**. El **valor** puede **ser TRUE o FALSE** (booleano):

```
$string = auto_link($string, 'both', TRUE);
```

url_title()

Toma como entrada una cadena de caracteres y crea una cadena de URL humanamente legible. Esto es útil si, por ejemplo, si Ud tiene un blog en el que le gustaría usar el título de sus entradas como URL. Ejemplo:

```
$title = "Que está mal en el CSS?";  
  
$url_title = url_title($title);  
  
// Produce: Que-está-mal-en-el-CSS
```

El segundo parámetro determina el delimitador de palabra. Por defecto se usa el guion (**dash**). Las opciones son: **dash**, o **underscore**:

```
$title = "Que está mal en el CSS?";  
  
$url_title = url_title($title, 'underscore');  
  
// Produce: Que_está_mal_en_el_CSS
```

El tercer parámetro determina si se fuerza o no por los caracteres en minúsculas. Por defecto no. Las opciones son **TRUE/FALSE** booleanos:

```
$title = "Que está mal en el CSS?";  
  
$url_title = url_title($title, 'underscore', TRUE);  
  
// Produce: que_está_mal_en_el_css
```

prep_url()

Esta función agregará **http://** en caso que falte un esquema en la URL. Pase la cadena de URL a la función de este modo:

```
$url = "example.com";  
  
$url = prep_url($url);
```

redirect()

Hace una "redirección de encabezado" a la URI especificada. Si especifica la URL completa del sitio, se armará ese enlace. Pero para enlaces locales, simplemente proveer los segmentos URI al controlador que quiere direccionar para crear el enlace. La función armará la URL basándose en los valores del archivo de configuración.

El segundo parámetro opcional le permite elegir entre el método **location** (por defecto) o **refresh**. **Location** es más rápido, pero a veces puede causar problemas en servidores Windows. El tercer parámetro opcional le permite enviar un código de respuesta HTTP específico – esto se podría usar por ejemplo para crear redirecciones 301 para los motores de búsqueda. El código de respuesta por defecto es 302. El tercer parámetro solo está disponible para redirecciones **location** y no para **refresh**. Ejemplos:

```
if ($logged_in == FALSE)
{
    redirect('/login/form/', 'refresh');
}

// with 301 redirect
redirect('/article/13', 'location', 301);
```

Nota: Para que la función trabaje tiene que usarse antes que algo salga por el navegador, ya que utiliza encabezados de servidor.

Nota: Para un control muy fino sobre los encabezados, debería usar la función **set_header()** de la Clase Output.

Helper XML

El archivo del Helper XML file contiene funciones que lo ayudan a trabajar con datos XML.

Cargar este Helper

Este helper se carga usando el siguiente código:

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

Están disponibles las siguientes funciones:

xml_convert('string')

Toma una cadena como entrada y convierte los siguientes caracteres XML reservados a entidades:

Ampersands: &

Caracteres menor que y mayor que: < >

Comillas simples y dobles: ' "

Guiones: -

Esta función ignora los signos & si son parte de entidades de carácter existentes. Ejemplo:

```
$string = xml_convert($string);
```



Anexos

Anexo I: Actualizar desde una Versión Anterior

Por favor lea las notas de actualización correspondientes a la versión desde la que está actualizando.

Actualizando de 2.1.2 a 2.1.3

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplazar todos los archivos y directorios en su carpeta **system** y reemplazar su archivo **index.php**. Si realizó modificaciones en su archivo **index.php** tendrán que hacerse nuevamente en este.

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Actualizando de 2.1.1 a 2.1.2

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplazar todos los archivos y directorios en su carpeta **system** y reemplazar su archivo **index.php**. Si realizó modificaciones en su archivo **index.php** tendrán que hacerse nuevamente en este.

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Actualizando de 2.1.0 a 2.1.1

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplazar todos los archivos y directorios en su carpeta **system** y reemplazar su archivo **index.php**. Si realizó modificaciones en su archivo **index.php** tendrán que hacerse nuevamente en este.

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Step 2: Replace config/mimes.php

Se actualizó este archivo de configuración para que contenga más tipos mime del usuario. Por favor cópielo, a **application/config/mimes.php**.

Step 3: Actualizar sus tablas de direcciones IP

Esta actualización agrega soporte para direcciones IP IPv6. Para almacenarlas, se necesita agrandar las columnas **ip_address** a 45 caracteres. Por ejemplo, la tabla sesiones de CodeIgniter necesitará cambiar a:

```
ALTER TABLE ci_sessions CHANGE ip_address ip_address varchar(45) default '0' NOT NULL
```

Actualizar de 2.0.3 a 2.1.0

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplazar todos los archivos y directorios en su carpeta **system** y reemplazar su archivo **index.php**. Si realizó modificaciones en su archivo **index.php** tendrán que hacerse nuevamente en este.

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Reemplazar config/user_agents.php

Se actualizó este archivo de configuración para que contenga más tipos de agentes de usuario. Por favor cópielo a **application/config/user_agents.php**.

Actualizar de 2.0.2 a 2.0.3

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplazar todos los archivos y directorios en su carpeta **system** y reemplazar su archivo **index.php**. Si realizó modificaciones en su archivo **index.php** tendrán que hacerse nuevamente en este.

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar los archivos de CodeIgniter

Reemplazar todos los archivos y directorios en su carpeta **system** con las nuevas versiones.

Paso 3: Actualizar su archivo index.php principal

Si está ejecutando un archivo **index.php** original, simplemente reemplace su versión con la nueva.

Si su archivo **index.php** tiene modificaciones internas, por favor agregue sus modificaciones al nuevo archivo y úselo.

Paso 4: Reemplazar config/user_agents.php

Se actualizó este archivo de configuración para que contenga más tipos de agentes de usuario. Por favor cópielo a **application/config/user_agents.php**.

Paso 5: Cambiar las referencias de la constante EXT a ".php"

Nota: La constante EXT se marcó como obsoleta, pero no se eliminó de la aplicación. Le recomendamos que haga los cambios tan pronto como sea posible.

Paso 6: Elimine APPPATH.'third_party' de autoload.php

Abrir **application/autoload.php** y buscar lo siguiente:

```
$autoload['packages'] = array(APPPATH.'third_party');
```

Si no eligió cargar ningún paquete adicional, esa línea se puede cambiar a :

```
$autoload['packages'] = array();
```

Lo que debería proporcionar una ganancia en el rendimiento nominal si no hay carga automática de paquetes.

Actualizar las Tablas de Sesiones de la Base de Datos

Si está usando sesiones de base de datos con la Biblioteca Session de CodeIgniter, por favor actualice su tabla `ci_sessions` de la base de datos como se muestra a continuación:

```
CREATE INDEX last_activity_idx ON ci_sessions(last_activity);
ALTER TABLE ci_sessions MODIFY user_agent VARCHAR(120);
```

Actualizar de 2.0.1 a 2.0.2

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplazar todos los archivos y directorios en su carpeta **system** y reemplazar su archivo **index.php**. Si realizó modificaciones en su archivo **index.php** tendrán que hacerse nuevamente en este.

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Eliminar las llamadas de carga de la Biblioteca Security

Se movió Security al núcleo y ahora se carga automáticamente. Asegurarse de quitar cualquier llamada para cargar, ya que dará lugar a errores de PHP.

Paso 3: Mover MY_Security

Si está anulando o extendiendo la Biblioteca Security, necesitará moverla a **application/core**.

Actualizar de 2.0.0 a 2.0.1

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace todos los archivos y directorios en su carpeta **system** y reemplace su archivo **index.php**. Si se hicieron modificaciones a su **index.php**, ellas se deberán hacer otra vez en la nueva versión.

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Reemplazar config/mimes.php

Este archivo de configuración se actualizó para contener más tipos mime, por favor copiarlo a **application/config/mimes.php**.

Paso 3: Verificar si hay formularios que envían al controlador por defecto

El comportamiento por defecto para **form_open()** cuando se lo llamaba sin parámetros solía ser el envío al controlador por defecto, pero ahora dejará una acción vacía (`action=""`) significando que el formulario será enviado a la URL actual. Si el envío al controlador por defecto fue el comportamiento esperado, se necesitará cambiarlo de:

```
echo form_open(); //<form action="" method="post" accept-charset="utf-8">
```

a usar **base_url()**:

```
echo form_open('/'); //<form action="http://example.com/index.php/" method="post" accept-charset="utf-8">
echo form_open(base_url()); //<form action="http://example.com/" method="post" accept-charset="utf-8">
```

Actualizar de 1.7.2 a 2.0.0

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace todos los archivos y directorios en su carpeta **system excepto** su carpeta **application**.

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Ajustar get_dir_file_info() donde sea necesario

La versión 2.0.0 trae un cambio no compatible para **get_dir_file_info()** en el Helper File. Los cambios no compatibles son extremadamente raros en CodeIgniter, pero éste sentimos estaba justificado debido a lo fácil que era crear graves problemas de rendimiento del servidor. Si necesita recursividad donde está usando esta función helper, cambie tales instancias estableciendo el segundo parámetro, **\$top_level_only** a **FALSE**:

```
get_dir_file_info('/ruta/al/directorio', FALSE);
```

Paso 3: Convertir sus Plugins en Helpers

CodeIgniter 2.0.0 se deshace del sistema de "Plugin" ya que su funcionalidad es idéntica a la de los Helpers, pero no extensible. Necesitará renombrar su archivo de plugin de **filename_pi.php** a **filename_helper.php**,

moverlos a su carpeta **helpers** y cambiar todas las instancias de:

```
$this->load->plugin('foo');
```

a:

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

Paso 4: Actualizar datos encriptados almacenados

Nota: Si la aplicación no utiliza la biblioteca de cifrado, no almacena los datos encriptados de forma permanente, o está en un entorno que no admite Mcrypt, puede omitir este paso.

La biblioteca Encrypt tuvo una cantidad de mejoras, algunas para la fortaleza de la encriptación y otras para el rendimiento, lo que tiene la inevitable consecuencia de no permitir más decodificar datos encriptados con la versión original de esta biblioteca. Para ayudar con la transición, se agregó un nuevo método, **encode_from_legacy()** que decodificará los datos con el algoritmo original y devolverá una cadena recodificada usando los métodos mejorados. Esto le permitirá reemplazar fácilmente datos encriptados viejos con datos encriptados nuevos en su aplicación, sea tanto al vuelo como en masa.

Por favor leer cómo usar este método en la documentación de la biblioteca Encrypt.

Paso 5: Eliminar las llamadas de carga para el helper de compatibilidad

El helper de compatibilidad se eliminó del núcleo de CodeIgniter. Todos los métodos en él deberían estar nativamente disponibles en las versiones soportadas de PHP.

Paso 6: Actualizar la extensión de clases

Ahora todas las clases del núcleo están prefijas con **CI_**. Actualice los Modelos y Controladores para extender **CI_Model** y **CI_Controller**, respectivamente.

Paso 7: Actualizar las llamadas al Constructor Padre

Todas las clases nativas de CodeIgniter usan ahora la convención **__construct()** de PHP 5. Por favor actualice las bibliotecas extendidas para llamar a **parent::__construct()**.

Paso 8: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.7.1 a 1.7.2

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **system/codeigniter**
- **system/database**
- **system/helpers**

- **system/language**
- **system/libraries**
- **index.php**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Eliminar header() de la plantilla del error 404

Si está usando **header()** en su plantilla del error 404, tal como el caso con la plantilla **error_404.php** por defecto mostrada abajo, quite esa línea de código.

```
<?php header ("HTTP/1.1 404 Not Found"); ?>
```

Ahora los encabezados de estado 404 se manejan adecuadamente en el método **show_404()**.

Paso 3: Confirmar su system_path

Si actualizó el archivo **index.php**, confirme que la variable **\$system_path** está establecida a la carpeta del sistema de su aplicación.

Paso 4: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.7.0 a 1.7.1

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **system/codeigniter**
- **system/database**
- **system/helpers**
- **system/language**
- **system/libraries**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.6.3 a 1.7.0

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **system/codeigniter**
- **system/database**
- **system/helpers**
- **system/language**
- **system/libraries**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su Tabla de Sesiones

Si está usando la clase Session en su aplicación, Y si está almacenando datos de sesión en la base de datos, tiene que agregar una nueva columna llamada user_data a su tabla de sesiones. Aquí está un ejemplo de lo cómo se vería esta columna en MySQL:

```
user_data text NOT NULL
```

Para agregar esta columna, ejecutará una consulta similar a esta:

```
ALTER TABLE `ci_sessions` ADD `user_data` text NOT NULL
```

Encontrará más información acerca de la nueva funcionalidad de sesiones en la página de la clase Session.

Paso 3: Actualizar su Sintaxis de Validación

Este es un paso **opcional** pero recomendado para gente que usa actualmente la clase Validation. CI 1.7 introduce una nueva clase de Validación de Formularios, que reemplaza a la antigua biblioteca de Validación. Dejamos la versión vieja para que las aplicaciones existentes que la usan no se rompan, pero lo animamos a migrar a la nueva versión tan pronto como sea posible. Por favor, lea las instrucciones cuidadosamente ya que la nueva biblioteca funciona un poco diferente, y tiene varias características nuevas.

Paso 4: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.6.2 a 1.6.3

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **system/codeigniter**
- **system/database**
- **system/helpers**
- **system/language**
- **system/libraries**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.6.1 a 1.6.2

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **system/codeigniter**
- **system/database**
- **system/helpers**
- **system/language**
- **system/libraries**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Clave de Encriptación

Si está usando sesiones, abra **application/config/config.php** y verifique que estableció una clave de encriptación.

Paso 3: Archivo Constants

Copiar **/application/config/constants.php** a su instalación y modificarlo si es necesario.

Paso 4: Archivo Mimes

Reemplazar **/application/config/mimes.php** con la versión descargada. Si agregó tipos mime personalizados, deberá volver a agregarlos.

Paso 5: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.6.0 a 1.6.1

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **system/codeigniter**
- **system/database**
- **system/helpers**
- **system/language**
- **system/libraries**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.5.4 a 1.6.0

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **system/codeigniter**
- **system/database**
- **system/helpers**
- **system/libraries**
- **system/plugins**
- **system/language**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Agregar time_to_update a su config.php

Agregar lo siguiente a **application/config/config.php** con otras opciones de configuración de sesión:

```
$config['sess_time_to_update'] = 300;
```

Paso 3: Agregar \$autoload['model']

Agregar lo siguiente a **application/config/autoload.php**:

```
/*
| -----
| Auto-load Model files
| -----
| Prototype:
|
| $autoload['model'] = array('my_model');
|
*/
$autoload['model'] = array();
```

Paso 4: Agregar a su database.php

Hacer los siguientes cambios a su archivo **application/config/database.php**:

Agregar la siguiente variable antes de las opciones de configuración de la base de datos, con **\$active_group**

```
$active_record = TRUE;
```

Quitar lo siguiente de las opciones de configuración de la base de datos:

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

Agregar lo siguiente a las opciones de configuración de la base de datos:

```
$db['default']['char_set'] = "utf8";
$db['default']['dbcollat'] = "utf8_general_ci";
```

Paso 5: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.5.3 a 1.5.4

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **application/config/mimes.php**
- **system/codeigniter**
- **system/database**
- **system/helpers**
- **system/libraries**
- **system/plugins**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Agregar el conjunto de caracteres a su config.php

Agregar lo siguiente a **application/config/config.php**:

```
/*
| -----
| Default Character Set
| -----
|
| Esto determina que conjunto de caracteres se usa por defecto en varios
| métodos que necesitan que se provea un conjunto de caracteres.
|
*/
$config['charset'] = "UTF-8";
```

Paso 3: Carga automática de archivos de idiomas

Si quiere cargar cualquier archivo de idioma automáticamente, agregue esta línea a **application/config/autoload.php**:

```
$autoload['language'] = array();
```

Paso 4: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.5.2 a 1.5.3

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **system/database/drivers**
- **system/helpers**
- **system/libraries/Input.php**
- **system/libraries/Loader.php**
- **system/libraries/Profiler.php**
- **system/libraries/Table.php**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.5.0 a 1.5.2

Nota: Las instrucciones en esta página asumen que está ejecutando la versión 1.5.0 o 1.5.1. Si no actualizó a esa versión, hágalo primero.

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

system/helpers/download_helper.php
system/helpers/form_helper.php
system/libraries/Table.php
system/libraries/User_agent.php
system/libraries/Exceptions.php
system/libraries/Input.php
system/libraries/Router.php
system/libraries/Loader.php
system/libraries/Image_lib.php
system/language/english/unit_test_lang.php
system/database/DB_active_rec.php
system/database/drivers/mysql/mysqli_driver.php
codeigniter/

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.4.1 a 1.5.0

Nota: Las instrucciones en esta página asumen que Ud está ejecutando la versión 1.4.1. Si no actualizó a esta versión, hágalo primero.

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **application/config/user_agents.php** (archivo nuevo para 1.5)
- **application/config/smileys.php** (archivo nuevo para 1.5)
- **codeigniter/**
- **database/** (carpeta nueva para 1.5. Reemplaza a la carpeta "drivers")
- **helpers/**
- **language/**
- **libraries/**
- **scaffolding/**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su archivo database.php

Abra su archivo **application/config/database.php** y **agregue** estos ítems nuevos:

```
$db['default']['cache_on'] = FALSE;  
$db['default']['cachedir'] = '';
```

Paso 3: Actualizar su archivo config.php

Abra su archivo **application/config/config.php** y **agregue** estos nuevos ítems:

```
/*  
|-----  
| Class Extension Prefix  
|-----  
|  
| Este ítem le permite establecer el prefijo nombre de archivo/nombre de clase  
| al extender bibliotecas nativas. Para más información lea la guía del usuario:  
|  
| http://codeigniter.com/user_guide/general/core_classes.html  
| http://codeigniter.com/user_guide/general/creating_libraries.html  
|  
*/  
$config['subclass_prefix'] = 'MY_';
```

```
/*
| -----
| Rewrite PHP Short Tags
| -----
|
| Si su aplicación PHP no tiene habilitado el soporte para etiquetas cortas, CI
| puede sobrescribir las etiquetas al vuelo, permitiéndole usar la sintaxis
| en sus archivos de vista. Las opciones son TRUE o FALSE (booleanos)
|
*/
$config['rewrite_short_tags'] = FALSE;
```

En el mismo archivo **quite** este ítem:

```
/*
| -----
| Enable/Disable Error Logging
| -----
|
| Si le gustaría que los mensajes de depuración o error se registren, establezca
| esta variable a TRUE (booleano). Nota: Tiene que establecer los permisos de
| archivo en la carpeta "logs" de tal forma que se pueda escribir.
|
*/
$config['log_errors'] = FALSE;
```

El registro de errores ahora está deshabilitado simplemente fijando el umbral a cero.

Paso 4: Actualizar su archivo index.php principal

Si está ejecutando un archivo **index.php** original, simplemente reemplace su versión con la nueva.

Si su archivo **index.php** tiene modificaciones internas, por favor agregue sus modificaciones al nuevo archivo y úselo.

Paso 5: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.4.0 a 1.4.1

Nota: Las instrucciones en esta página asumen que Ud está ejecutando la versión 1.4.0. Si no actualizó a esta versión, hágalo primero.

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **codeigniter**
- **drivers**
- **helpers**

- **libraries**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su archivo config.php

Abra su archivo **application/config/config.php** y **agregue** estos nuevos ítems:

```
/*
| -----
| Output Compression
| -----
|
| Enables Gzip output compression for faster page loads. When enabled,
| the output class will test whether your server supports Gzip.
| Even if it does, however, not all browsers support compression
| so enable only if you are reasonably sure your visitors can handle it.
|
| VERY IMPORTANT: If you are getting a blank page when compression is enabled it
| means you are prematurely outputting something to your browser. It could
| even be a line of whitespace at the end of one of your scripts. For
| compression to work, nothing can be sent before the output buffer is called
| by the output class. Do not "echo" any values with compression enabled.
|
*/
$config['compress_output'] = FALSE;
```

Paso 3: Renombrar un Item de Carga Automática

Abra el siguiente archivo: **application/config/autoload.php**

Encuentre este ítem del array:

```
$autoload['core'] = array();
```

Y renómbrello a esto:

```
$autoload['libraries'] = array();
```

Estos cambios se hicieron para mejorar la claridad ya que algunos usuarios no estaban seguros que sus bibliotecas se pudieran cargar automáticamente.

Paso 4: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.3.3 a 1.4.0

Nota: Las instrucciones en esta página asumen que Ud está ejecutando la versión 1.3.3. Si no actualizó a esta versión, hágalo primero.

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **application/config/hooks.php**
- **application/config/mimes.php**
- **codeigniter**
- **drivers**
- **helpers**
- **init**
- **language**
- **libraries**
- **scaffolding**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su archivo config.php

Abra su archivo **application/config/config.php** y **agregue** estos nuevos ítems:

```
/*
| -----
| Enable/Disable System Hooks
| -----
|
| If you would like to use the "hooks" feature you must enable it by
| setting this variable to TRUE (boolean). See the user guide for details.
|
*/
$config['enable_hooks'] = FALSE;

/*
| -----
| Allowed URL Characters
| -----
|
| This lets you specify which characters are permitted within your URLs.
| When someone tries to submit a URL with disallowed characters they will
| get a warning message.
|
| As a security measure you are STRONGLY encouraged to restrict URLs to
| as few characters as possible. By default only these are allowed: a-z 0-9~%.:_-_
|
| Leave blank to allow all characters -- but only if you are insane.
|
| DO NOT CHANGE THIS UNLESS YOU FULLY UNDERSTAND THE REPERCUSSIONS!!
|
*/
$config['permitted_uri_chars'] = 'a-z 0-9~%.:_-';
```

Paso 3: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.3.2 a 1.3.3

Nota: Las instrucciones en esta página asumen que Ud está ejecutando la versión 1.3.2. Si no actualizó a esta versión, hágalo primero.

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **codeigniter**
- **drivers**
- **helpers**
- **init**
- **libraries**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar sus Modelos

Si **NO** está usando la funcionalidad de Modelos de CodeIgniter, haga caso omiso a este paso.

Desde la versión 1.3.3, CodeIgniter **no** conecta automáticamente a la base de datos cuando el modelo se carga. Esto le permite una mayor flexibilidad en determinar qué base de datos querría usar con sus modelos. Si su aplicación no está a su base de datos antes que a un modelo que se está cargando, Ud tendrá que actualizar su código. Hay varias opciones para conectar, como se describe aquí.

Paso 3: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.3.1 a 1.3.2

Nota: Las instrucciones en esta página asumen que Ud está ejecutando la versión 1.3.1. Si no actualizó a esta versión, hágalo primero.

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **drivers**
- **init**
- **libraries**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.3 a 1.3.1

Nota: Las instrucciones en esta página asumen que Ud está ejecutando la versión 1.3. Si no actualizó a esta versión, hágalo primero.

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **drivers**
- **init/init_unit_test.php** (nuevo en 1.3.1)
- **language/**
- **libraries**
- **scaffolding**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de 1.2 a 1.3

Nota: Las instrucciones en esta página asumen que Ud está ejecutando la versión 1.2. Si no actualizó a esta versión, hágalo primero.

Antes de realizar una actualización debería sacar de línea su sitio reemplazando el archivo **index.php** con uno estático.

Paso 1: Actualizar sus archivos CodeIgniter

Reemplace estos archivos y directorios de su carpeta **system** con las nuevas versiones:

- **application/models/** (nuevo en 1.3)
- **codeigniter** (nuevo en 1.3)
- **drivers**
- **helpers**
- **init**
- **language**
- **libraries**
- **plugins**
- **scaffolding**

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Paso 2: Actualizar sus archivos de error

La versión 1.3 contiene dos nuevas plantillas de error ubicadas en **application/errors**, y por consistencia de nombres las otras plantillas de error se renombraron.

Si **no** personalizó ninguna plantilla de error, simplemente reemplace esta carpeta:

- **application/errors/**

Si **personalizó** sus plantillas de error, renómbrelas así:

- **404.php = error_404.php**
- **error.php = error_general.php**
- **error_db.php** (nueva)
- **error_php.php** (nueva)

Paso 3: Actualizar su archivo index.php

Por favor, abra su archivo **index.php** principal (ubicado en su raíz). Al final del archivo, cambie esto:

```
require_once BASEPATH.'libraries/Front_controller'.EXT;
```

A esto:

```
require_once BASEPATH.'codeigniter/CodeIgniter'.EXT;
```

Paso 4: Actualizar su archivo config.php

Abra su archivo **application/config/config.php** y agregue estos nuevos elementos:

```
/*
| -----
| URL suffix
| -----
|
| This option allows you to add a suffix to all URLs.
| For example, if a URL is this:
|
| example.com/index.php/products/view/shoes
|
| You can optionally add a suffix, like ".html",
| making the page appear to be of a certain type:
|
| example.com/index.php/products/view/shoes.html
|
*/
$config['url_suffix'] = "";  
  
/*
| -----
| Enable Query Strings
| -----
|
| By default CodeIgniter uses search-engine and
| human-friendly segment based URLs:
```

```
| example.com/who/what/where/
|
| You can optionally enable standard query string
| based URLs:
|
| example.com?who=me&what=something&where=here
|
| Options are: TRUE or FALSE (boolean)
|
| The two other items let you set the query string "words"
| that will invoke your controllers and functions:
| example.com/index.php?c=controller&m=function
|
*/
$config['enable_query_strings'] = FALSE;
$config['controller_trigger'] = 'c';
$config['function_trigger'] = 'm';
```

Paso 5: Actualizar su archivo database.php

Abra su archivo **application/config/database.php** y agregue estos nuevos elementos:

```
$db['default']['dbprefix'] = "";
$db['default']['active_r'] = TRUE;
```

Paso 6: Actualizar su guía del usuario

Por favor reemplace su copia de la guía del usuario con la nueva versión, incluyendo los archivos de imágenes.

Actualizar de Beta 1.1 a Final 1.2

Nota: Si tiene algún archivo desarrollado a medida, por favor haga una copia primero.

Para actualizar a la Versión 1.2 por favor reemplace los siguiente directorios con las nuevas versiones:

- **drivers**
- **helpers**
- **init**
- **language**
- **libraries**
- **plugins**
- **scaffolding**

Por favor, también reemplace la copia local de la guía del usuario con la nueva versión.

Actualizar de Beta 1.0 a Beta 1.1

Para actualizar a Beta 1.1, por favor ejecute los siguientes pasos:

Paso 1: Reemplazar el archivo index

Reemplace su archivo **index.php** principal con el nuevo archivo **index.php**. Nota: Si renombró su carpeta **system** necesitará editar esta información en un nuevo archivo.

Paso 2: Reubicar la carpeta config

Esta versión de CodeIgniter ahora permite varios conjuntos de "aplicaciones" para compartir un conjunto completo de archivos de backend. Para habilitar cada aplicación para que tenga sus propios valores de configuración, el directorio **config** ahora tiene que estar dentro de su carpeta **application**, por lo que deberá moverla ahí.

Paso 3: Reemplazar directorios

Reemplace los siguientes directorios con sus nuevas versiones:

- **drivers**
- **helpers**
- **init**
- **libraries**
- **scaffolding**

Paso 4: Agregar el archivo de idioma del calendario

Hay un nuevo archivo de idioma correspondiente a la nueva clase de calendario que se tiene que agregar a su carpeta de idiomas. Agregue el siguiente elemento a su versión: **language/english/calendar_lang.php**.

Paso 5: Editar el archivo config

El archivo original **application/config/config.php** tiene un error de tipo. Abra el archivo y busque los ítems relacionados a las cookies:

```
$conf['cookie_prefix'] = "";
$conf['cookie_domain'] = "";
$conf['cookie_path'] = "/";
```

Cambie el nombre del array de **\$conf** a **\$config**, como esto:

```
$config['cookie_prefix'] = "";
$config['cookie_domain'] = "";
$config['cookie_path'] = "/";
```

Finalmente, agregue el siguiente nuevo ítem al archivo de configuración (y edite la opción si es necesario):

```
/*
| -----
| URI PROTOCOL
| -----
|
| This item determines which server global
| should be used to retrieve the URI string. The
| default setting of "auto" works for most servers.
| If your links do not seem to work, try one of
| the other delicious flavors:
|
| 'auto' Default - auto detects
| 'path_info' Uses the PATH_INFO
| 'query_string' Uses the QUERY_STRING
*/
$config['uri_protocol'] = "auto";
```

Anexo II: Registro de Cambios

Versión 2.1.3

Fecha de Liberación: 8 de octubre de 2012

Errores corregidos para 2.1.3

- Corregido el error (#1543) – El método `get_metadata()` de almacenamiento en caché basada en archivo usaba una clave de array inexistente para buscar el valor TTL.
- Corregido el error (#1314) – El método `sess_destroy()` de la **Biblioteca Session** no destruía el array userdata.
- Corregido el error donde la **Biblioteca Profiler** presentara el error `E_WARNING` si los datos del usuario de la sesión contenian objetos.
- Corregido el error (#1699) - La **Biblioteca Migration** ignoraba el valor de `$config['migration_path']`.
- Corregido el error (#227) - **Input Library** permitía la suplantación sin condiciones de direcciones IP de clientes de HTTP mediante el encabezado `HTTP_CLIENT_IP`.
- Corregido el error (#907) - La **Biblioteca Input** ignoraba los encabezados `HTTP_X_CLUSTER_CLIENT_IP` y `HTTP_X_CLIENT_IP` al verificar proxies.
- Corregido el error (#940) - `csrf_verify()` solía establecer la cookie CSRF al procesar una solicitud POST sin datos POST reales, lo que resultaba en la validación de una solicitud que debería considerarse inválida.
- Corregido el error en la **Biblioteca Security** donde una cookie CSRF se creaba aún cuando `$config['csrf_protection']` fuera `FALSE`.
- Corregido el error (#1715) – La **Biblioteca Input** disparaba `csrf_verify()` en solicitudes CLI.

Nota: Para ver los cambios de las versiones anteriores, lea el archivo [changelog.html](#). También puede consultar el mismo archivo en la carpeta [user_guide/](#) de su instalación.