

Giuseppe Bellisano

**Guida allo sviluppo moderno con**  
**CodeIgniter**

*versione 0.3.1*

Realizzato con **L<sup>A</sup>T<sub>E</sub>X**  
15 settembre 2014

Questo lavoro è stato realizzato con L<sup>A</sup>T<sub>E</sub>X su GNU/Linux usando ArsClassica, una rielaborazione dello stile ClassicThesis di André Miede ispirato a Gli elementi dello stile tipografico di Robert Bringhurst.

I nomi commerciali, i loghi e i marchi registrati menzionati nella guida appartengono ai rispettivi proprietari, i pacchetti e le relative documentazioni ai rispettivi autori.

CodeIgniter è un prodotto registrato da EllisLab: <http://ellislab.com>

Guida realizzata sulla base della documentazione ufficiale. Autore: Giuseppe Bellisano: [bellisano.wordpress.com](http://bellisano.wordpress.com)

Ultima versione: <https://www.dropbox.com/s/egsna0591sfizwq/book.pdf>

This work is free; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This work is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this work; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

# INDICE

1	IL FRAMEWORK	1
1.1	Un framework non serve	1
1.2	Design e architettura	6
1.3	Riepilogo	7
2	INSTALLARE CODEIGNITER	11
2.1	Server locale e remoto	11
2.2	Installare CodeIgniter	15
2.3	Configurazione	15
2.4	Primi accorgimenti per la sicurezza	17
2.5	Siamo pronti	18
2.6	Risoluzione dei problemi	18
3	I MODERNI PATTERN DI SVILUPPO	23
3.1	Separare presentazione e logica	23
3.2	Iniziare con CodeIgniter	24
3.3	Model-View-Controller	24
3.4	Architettura del progetto	26
3.5	Riepilogo	29
4	I FANTASTICI TRE	31
4.1	Il Controller	33
4.2	La View	39
4.3	Restituire una View come un dato	44
4.4	Personalizziamo l'header	45
4.5	Evoluzione del progetto	49
4.6	Il Model	52
4.7	Riepilogo	64
5	ARGOMENTO AVANZATO I	67
5.1	Gestione degli URL	67
5.2	Abilitare le Query String	68
5.3	Aggiungere un suffisso URL	68
5.4	Un file index di troppo	69
5.5	Helper e Plugin	70
5.6	Interfaccia e implementazione	70
5.7	Gli Helper nel dettaglio	71
5.8	Plugin ridefiniti	74
5.9	Riepilogo	75

6	ARGOMENTO AVANZATO II	77
6.1	Le Librerie	77
6.2	I Driver	83
6.3	Classe Core	86
6.4	Gli Hook	89
6.5	Web Page caching	94
6.6	Profiling	96
6.7	Gestire le applicazioni	99
6.8	Template Engine	101
6.9	Funzioni comuni	103
6.10	Fondamenti di Sicurezza	107
7	ARGOMENTO AVANZATO III	115
7.1	Stile e sintassi del PHP	115
7.2	Query SQL	129
7.3	URI routing e caching	130
7.4	Utilizzo di wildcard ed regex	134
7.5	Wildcard	134
7.6	Regular Expression	136
7.7	Rotte riservate	136
7.8	Nomi riservati	138
7.9	Scripting	140
8	ELENCO DELLE CLASSI	141
8.1	Classe Benchmarking	143
8.2	Classe/Driver Caching	146
8.3	Classe Calendar	150
8.4	Classe Shopping Cart	155
8.5	Classe Config	161
8.6	Classe Database	165
8.7	Classe Active Record	184
8.8	Classe Email	227
8.9	Classe Encryption	233
8.10	Classe Upload File	238
8.11	Classe Validazione dei Form	245
8.12	Classe FTP	269
8.13	Classe HTML table	274
8.14	Classe Image Manipulation	280
8.15	Classe Input	291
8.16	Classe Javascript	297
8.17	Classe Loader	303
8.18	Classe Language	308
8.19	Classe Migration	310
8.20	Classe Output	313
8.21	Classe Pagination	316
8.22	Classe Security	320
8.23	Classe Session	322
8.24	Classe Trackback	330

8.25	Classe Template Parser	336
8.26	Classe Typography	339
8.27	Classe Unit Testing	341
8.28	Classe User Agent	349
8.29	Classe XML-RPC e XML-RPC Server	353
8.30	Classe Zip Encoding	364
9	ELENCO DEGLI HELPER	369
9.1	Helper Array	370
9.2	Helper Captcha	373
9.3	Helper Cookie	376
9.4	Helper Date	377
9.5	Helper Directory	382
9.6	Helper Download	384
9.7	Helper Email	385
9.8	Helper File	386
9.9	Helper Form	389
9.10	Helper HTML	398
9.11	Helper Inflector	405
9.12	Helper Language	407
9.13	Helper Number	408
9.14	Helper Path	409
9.15	Helper Security	410
9.16	Helper Smiley	411
9.17	Helper String	415
9.18	Helper Text	418
9.19	Helper Typography	421
9.20	Helper URL	422
9.21	Helper XML	427
9.22	Alternative PHP Cache	428

ACRONIMI	431
----------	-----

## NOTE DI VERSIONE

La guida è in fase di sviluppo ed è da considerarsi provvisoria in ogni sua parte. Se avete idee su argomenti da inserire, togliere o modificare in questa guida, o se vi dovesse capitare di notare un errore, sia di battitura che di sostanza, mi fareste un favore comunicandomelo, così che possa apportare le opportune correzioni in versioni successive.

## PREFAZIONE

“Scrivere un libro è un’avventura.  
Per iniziare, è un gioco e un  
divertimento;  
poi diventa un padrone e infine un  
tiranno”.

---

Winston Churchill

Programmare oggi è indubbiamente più facile rispetto al passato. Grazie a programmi di sviluppo amichevoli e alla grande mole di documentazione reperibile in Rete, chiunque può prodigarsi nell’“arte della creazione digitale”, ma programmare con cura è però tutt’altro discorso. Aprire un semplice editor di testi e buttarsi a capofitto nello sviluppo è oggi ancora possibile, ma chiunque abbia scritto più di qualche centinaio di righe di codice, sa bene come questa pratica conduca ad un progetto confuso, difficile da comprendere (anche allo stesso sviluppatore), e in definitiva ingestibile.

Molti libri sulla programmazione si soffermano pedantemente su ogni istruzione di un linguaggio, evitando però di fornire soluzioni a problemi realistici. Ricordo ancora le tante lezioni universitarie: mi era stato insegnato come compilare la serie di fibonacci in ogni immaginabile linguaggio di programmazione, mai mai era stato fatto un cenno sui pattern di sviluppo, i paradigmi, le universalmente adottate. Tra i profani è abbastanza comune pensare: “Cosa ci vorrà a programmare un software?”, “Basta conoscere le basi di un linguaggio e scrivere, scrivere, scrivere”. Peccato che questa filosofia porti proprio ad una serie di problematiche che mina il buon esito di qualsiasi progetto, portando lo sviluppatore alla frustrazione e all’abbandono del proprio lavoro incompiuto. In tal senso non sono di aiuto neppure la maggior parte dei libri in commercio che spesso trattano la programmazione in maniera eccessivamente astratta e accademica: si concentrano sulla sintassi dei comandi, tralasciando importanti concetti sulla sicurezza, la memorizzazione e protezione dei dati sensibili, con le spiacevoli conseguenze a cui si assiste tutti i giorni nel web.

Con questa guida si è voluto accompagnare il lettore attraverso le metodologie che vengono effettivamente utilizzate nello sviluppo di software moderno. Si è cercato di andare oltre il solito elenco neutro di funzioni, e di fornire quando possibile, vere e proprie soluzioni ai problemi che si affrontano quotidianamente nello sviluppo.

Nel corso della guida gli argomenti vengono estesi progressivamente, andando oltre la documentazione ufficiale del framework CodeIgniter, su cui comunque questo lavoro si basa ampiamente. In particolare sono state trattate le metodologie di programmazione (pattern) che conducono ad uno sviluppo ordinato, modulare e riusabile del codice. Ci si è soffermati ovviamente sul cuore di CodeIgniter, ovvero il modello Model-View-Controll (MVC) che permette la separazione del codice dalla presentazione (ciò che si vede su schermo), senza trascurare di riportare le innumerevoli funzionalità, attraverso esempi di codice commentato.

Fin dalla prima pagina si vedrà il modo corretto di utilizzare CodeIgniter, sviluppando più aspetti di un realistico progetto web professionale. La prassi seguita è quella di estendere i temi affrontati in maniera amichevole e senza dare nulla per scontato al profano, a cui questa guida principalmente si rivolge. Ovviamente molti

aspetti importanti o capaci di suscitare curiosità sono stati trattati con meno dovizia per non appesantire l'apprendimento. In questi casi, quando possibile, si è cercato di indirizzare il lettore verso la documentazione ufficiale e risorse disponibili in Rete.

## A CHI È RIVOLTO QUESTO LIBRO

Anche se questa guida cerca di non dare alcun argomento per scontato, si presuppone che il lettore abbia acquisito qualche conoscenza con i linguaggi di markup come l'Hyper Text Mark-up Language ([HTML](#)) e i costrutti basilari dei linguaggi di programmazione al fine di richiamare correttamente i metodi ed impostare i loro argomenti. Non è necessario essere degli esperti di programmazione per utilizzare inizialmente un framework, ma per sfruttarne tutte le potenzialità e sviluppare un progetto professionale sarà necessaria costanza e una genuina curiosità.

## CONVENZIONI

Nella guida di stampo tecnico, sono presenti riferimenti a codice inline, directory, percorsi di sistema e variabili. Si è cercato di essere il più possibile chiari adottando le seguenti convenzioni basate su alcune regole:

1. Nonostante il termine “funzione” appartenga alla programmazione procedurale e il “metodo” a quella ad oggetti, in questo testo si utilizzano senza distinzione i due termini.
2. Diverse parole sono virgolettate, tra cui comandi, nomi di file, o percorsi di sistema. Se non diversamente specificato, i comandi “tra virgolette” devono essere eseguiti senza i relativi doppi apici.
3. Il riferimento a directory di sistema di CodeIgniter verranno evidenziate con un codice come **application/view/**. In alcuni casi il percorso indicato potrebbe contenere nella parte finale anche il file a cui si fa riferimento come in **application/view/index.php**.
4. I riferimenti a variabili, costanti e ai valori loro assegnati sono evidenziati da espressioni come **var = view()**.
5. Quando si vorrà mettere in risalto un nome proprio di un file questo sarà evidenziato come **index.php**.
6. I listati di codice sono stati messi in risalto all'interno di apposite tabelle in questo modo:



```
// caricamento dell'Helper per la gestione degli array
$this->load->helper('array');

// definizione di un array
$mioarray = array('nome' => 'Giuseppe',
                  'cognome' => 'Bellisano',
                  'citta' => 'Cagliari'
                );

// restituzione di un valore tramite la funzione fornita dall'Helper
echo element('nome', $mioarray);
```

## ARGOMENTI TRATTATI

L'esposizione del lavoro è articolata come segue:

- Nel primo capitolo viene offerta una visione d'insieme sui framework, sui vantaggi derivanti dalla loro adozione e vengono analizzate le peculiarità di CodeIgniter rispetto agli altri tool simili disponibili sul mercato.
- Nel secondo capitolo vengono spiegate le operazioni per installare CodeIgniter sul proprio sistema.
- Nel terzo capitolo vengono presentate le nozioni fondamentali alla base della moderna programmazione, con particolare attenzione al pattern [MVC](#).
- Nel quarto capitolo si esamina lo studio preliminare da condurre nello sviluppo di un progetto. Vengono descritti i vantaggi derivanti dall'individuazione dei requisiti funzionali e di quelli utente.
- Nel quinto capitolo viene introdotto il Controller, uno dei punti focali del framework demandato all'analisi e instradamento dei dati forniti dall'utente. Si incomincia a definire il progetto alla base dell'esercitazione.
- Nel sesto capitolo viene esaminato lo strumento dedito alla presentazione del nostro progetto: la View.
- Nel settimo capitolo ci si sofferma sul progetto sin qui sviluppato. Si analizzano i progressi compiuti e si evidenziano ulteriori punti meritevoli di essere perfezionati.
- Nell'ottavo capitolo viene introdotto il Model, parte essenziale del framework nelle iterazioni con la base di dati.
- Nel nono capitolo vengono introdotti gli Helper, significativi aiutanti nella realizzazione delle parti più ripetitive, ma per questo non meno importanti di un software.

- Nel decimo capitolo breve accenno ai Plugin che dalla versione 2.0 di CodeIgniter sono stati accorpati con gli Helper condividendone obiettivi e utilizzo.
- Nell'undicesimo capitolo vengono esaminate le Librerie, vere e proprie collezioni di classi utilizzabili per le funzionalità più disparate di un progetto.
- Nel capitolo si esaminano gli Hook, un comodo strumento dedicato a chi desidera personalizzare il core del framework CodeIgniter senza per questo precluderne la stabilità e il corretto funzionamento.
- Nel dedicesimo capitolo viene svolta un'analisi approfondita sugli URI e sul sistema di caching del framework.
- Nel tredicesimo capitolo infine, vengono descritte le pratiche volte a garantire la migliore sicurezza al prodotto sviluppato. Si analizzano, in tale contesto, gli strumenti messi a disposizione da CodeIgniter.

# 1 | IL FRAMEWORK

In questo capitolo vengono presentate le idee di fondo e le peculiarità che sono alla base di ogni framework, con particolare attenzione al tool CodeIgniter, che verrà utilizzato in tutti gli esempi di questa guida, e nella realizzazione dei progetti presentati.

## 1.1 UN FRAMEWORK NON SERVE

Questo titolo può apparire fuori luogo, ma anche doveroso: è necessario comprendere che ogni strumento offre dei vantaggi se usato nell'appropriato contesto. I tool di sviluppo sono spesso complessi, articolati in sottoprogrammi (plugin) e nonostante siano di aiuto in innumerevoli situazioni, risultano onerosi da apprendere e da impiegare nella realizzazione di un piccolo sito web. Certo, nessuno vieta di utilizzarlo comunque, così come nessuno ci proibisce di sollevare un barattolo di marmellata con un carrello elevatore.

Elenchiamo pertanto i motivi per cui un framework è "eccessivo":

- progetto composto da poche pagine
- mancanza di un database integrato
- sito statico con codice che non verrà utilizzato in seguito

Queste sono alcune linee di massima, e certamente ogni progetto è un caso a se stante su cui è sempre arduo generalizzare. In tutti i casi, padroneggiare un framework non risulterà mai un'esperienza negativa, anzi arricchirà le proprie conoscenze informatiche.

## PERCHÉ USARE UN FRAMEWORK

Originalità a parte nella scelta dei titoli, l'uso di un tool di sviluppo ha indubbi vantaggi come quello di potersi dedicare ad un progetto non banale, senza rischiare di affogare sotto una miriade di problematiche comuni: "come si chiama quella funzione che ho creato tempo addietro? Perché il metodo sviluppato ieri, ora non funziona più? Eppure non ho toccato nulla (o quasi)." Queste sono alcune delle frasi che, chi ha scritto anche poche righe di codice, si è ritrovato prima o poi a pronunciare, magari con gli occhi levati al cielo. L'utilizzo di un framework è di notevole aiuto in questi frangenti, poiché mette a disposizione dello sviluppatore una serie di moduli esterni già pronti da impiegare per la validazione dei dati, la creazione di form o la gestione di un database: insomma, si eviterà di reinventare la ruota ad ogni nuovo prodotto sviluppato. L'utilizzo di un framework non è

certamente la panacea di tutti i mali, ma una volta che si padroneggerà la filosofia che vi è alla base, unita alla necessaria pratica, sarà impossibile tornare indietro.

## UN'AMPIA SCELTA

Anche se in questa guida analizzeremo le funzionalità del prodotto di EllisLab, è doveroso tenere presente che non esiste un solo prodotto per lo sviluppo e ciò, in un sano mercato competitivo, è sempre un bene. Anche se le differenze tra i framework possono essere più o meno marcate, non si deve mai dimenticare lo scopo per cui questi vengono utilizzati: semplificare l'architettura e lo sviluppo di un progetto complesso. Detto questo, ogni framework adatto allo scopo è consigliato. Nel panorama attuale la scelta è molto vasta per funzionalità, complessità di apprendimento e performance. Ovviamente non esiste un prodotto migliore di un altro in senso assoluto, così come non esiste quello peggiore. È pressoché impossibile racchiudere in un solo prodotto tutti i pregi possibili: basta immaginare che quello più ricco di funzioni, è solitamente più complesso da apprendere, meno veloce e spesso sovrastimato alle proprie reali esigenze. Comunque non ci si spaventi: una volta appresa la filosofia alla base dello sviluppo moderno e fatta pratica con CodeIgniter, sarà quasi indolore utilizzare, se lo vorrete, un altro tool di sviluppo: molti aspetti sono infatti ampiamente condivisi tra i principali software:

- paradigma ad oggetti
- pattern architetturale [MVC](#)
- utilizzo dei sistemi di Object-relational mapping ([ORM](#))
- supporto all'i18n (ovvero multilingua)

Qui di seguito si illustreranno, senza la pretesa di essere esaustivi, le principali proposte sul mercato. Si rimarca nuovamente che la definizione di migliore o peggiore è fuorviante: esiste semplicemente il framework maggiormente adatto al proprio progetto.

- CodeIgniter v2. È il framework preso come riferimento in questa guida. Non è il più completo o il più utilizzato, ma nel suo arco ha molte frecce. Prima di tutto è leggero e veloce e, aspetto importante per chi muove i primi passi, è molto più amichevole rispetto ai suoi concorrenti.
- Symfony v2. È utilizzato per progetti di grandi dimensioni e risulta complesso da installare, configurare e apprendere nelle sue più piccole sfaccettature. Si tratta comunque di uno dei principali tool di sviluppo utilizzati nelle grandi aziende.
- Yii Framework v.1. La sua popolarità è in continuo crescendo per via di una curva di apprendimento meno ripida rispetto a Symfony e Zend. Incorpora il meglio dei punti chiave di Rails.

- Zend Framework v1. Può essere definito come il framework di sviluppo accademico. Insieme a Symfony risulta essere il tool più complesso da apprendere, ma anche quello più utilizzato in ambito commerciale. Richiede una conoscenza ottima del PHP: Hypertext Preprocessor ([PHP](#)) e Object Oriented Programming ([OOP](#)).
- CakePHP. Utilizzato in progetti dalle piccole-medie dimensioni è attualmente uno dei framework più apprezzati dagli sviluppatori.

## CODEIGNITER: L'OBIETTIVO

CodeIgniter è un Application Development Framework ovvero un toolkit per “persone che costruiscono siti web utilizzando [PHP](#)”. Il suo scopo è consentire lo sviluppo di progetti più velocemente rispetto a quello che si potrebbe fare scrivendo il codice da zero. Questo grazie ad una ricca dotazione di librerie già pronte per gli usi più comuni, tra l'altro facilmente accessibili grazie ad una semplice interfaccia e ad una struttura logica che permette l'accesso a queste librerie. CodeIgniter consente di concentrarsi sul proprio progetto, minimizzando il codice necessario al suo sviluppo ed evitando di reinventare “la ruota”.

## A CHI SI RIVOLGE

CodeIgniter è consigliato a quelle persone che:

- desiderano un framework dalle piccole dimensioni
- necessitano di performance eccezionali
- hanno bisogno di compatibilità con gli account di host che utilizzano varie configurazioni di [PHP](#)
- desiderano utilizzare un framework che richiede una configurazione minima
- hanno bisogno di un tool che non preveda l'uso di istruzioni impartiti da linea di comando
- vogliono utilizzare un framework che non utilizza codice dalle regole restrittive
- non desiderano imparare necessariamente un linguaggio di template, anche se uno di questi è utilizzabile opzionalmente
- desiderano evitare soluzioni complesse a favore di quelle più immediate
- necessitano di una documentazione chiara ed esaustiva

## I PUNTI DI FORZA

Le caratteristiche salienti di CodeIgniter sono anche i suoi punti di forza che lo distinguono dalle altre proposte sul mercato.

- CodeIgniter è gratuito
- CodeIgniter possiede una licenza Apache/BSD-style open source che lo rende fruibile da chiunque. Per maggiori informazioni si veda il contratto di licenza
- CodeIgniter è leggero. Il suo core system richiede solo poche e piccole librerie. Ciò è in netto contrasto con molti altri framework che richiedono molte più risorse. Librerie aggiuntive possono essere caricate dinamicamente, su richiesta, in base alle proprie esigenze
- CodeIgniter è veloce. Probabilmente in questo campo non ha rivali
- CodeIgniter usa **MVC**. Questo pattern architetturale consente la separazione tra logica e presentazione, aspetto fondamentale per i progetti in cui più persone dalle differenti competenze lavorano sulle stesse porzioni di codice
- CodeIgniter genera Uniform Resource Locator (**URL**) puliti e rivolti alla facile indicizzazione da parte dei motori di ricerca. Invece di utilizzare il metodo standard "query string" per gli **URL** che è sinonimo di sistemi dinamici, CodeIgniter utilizza un approccio basato sui segmenti come nell'indirizzo qui riportato (per maggiori informazioni si veda il capitolo [7.3 a pagina 130](#)):

`esempio.com/index.php/news/articolo/345`

Nota: Per impostazione predefinita, la stringa **index.php** è sempre presente nell'**URL**. Questa può comunque essere rimossa agendo semplicemente sul file **.htaccess** (vedi capitolo [5.3 a pagina 69](#)).

- CodeIgniter è ricco di librerie. Sono fornite e a corredo una gamma completa di strumenti che consentono le attività di sviluppo web più comuni, come l'accesso a un database, l'invio di email, la convalida dei dati di un form, la gestione delle sessioni, la manipolazione delle immagini, l'utilizzo dei dati eXtensible Markup Language-Remote Procedure Call (**XML-RPC**) e molto altro. Conseguenza di ciò, CodeIgniter evita la necessità di dover utilizzare librerie esterne per l'integrazione di funzioni aggiuntive come per esempio PHP Extension and Application Repository (**PEAR**)
- CodeIgniter è estendibile. Il sistema può essere facilmente esteso mediante l'uso delle proprie librerie, helper, o attraverso le estensioni di classe o gli Hook
- CodeIgniter non richiede un engine di template. Sebbene un apposito engine sia presente, quest'ultimo può essere utilizzato opzionalmente (si veda il capitolo [7.9 a pagina 140](#))

- CodeIgniter è ben documentato. I programmatori amano scrivere il codice ma odiano preparare la documentazione. CodeIgniter è caratterizzato da un codice sorgente estremamente pulito e ben commentato oltre che da una documentazione chiara e agevole
- CodeIgniter ha una comunità amichevole. Il forum presente all'indirizzo <https://ellislab.com/forums> costituisce il punto di ritrovo di appassionati che amano confrontarsi e aiutarsi nella risoluzione delle problematiche più comuni

## PECULIARITÀ TECNICHE

L'elenco delle caratteristiche di un software è ritenuto da alcuni come un modo molto superficiale per giudicare un'applicazione, in quanto non fornisce alcuna informazione sull'esperienza d'uso, sull'intuitività con cui è stata progettata oppure sulla qualità del codice, le performance, l'attenzione per i dettagli e le metodologie utilizzate per la sicurezza. Probabilmente l'unica strada per giudicare realmente una applicazione è utilizzarla. Installare CodeIgniter è un gioco da ragazzi, quindi si incoraggia a fare proprio questo. Per i più impazienti, ecco un elenco delle caratteristiche principali del framework:

- Sistema basato sul pattern [MVC](#)
- Estremamente leggero
- Ricco di classi di database con il supporto per diverse piattaforme
- Supporto per gli Active Record
- Validazione dei form e dei dati
- Filtri per la sicurezza e Cross-site scripting ([XSS](#))
- Gestione delle sessioni
- Classe per l'invio delle email. Supporto per gli allegati, email [HTML](#)/testuali, protocolli multipli (sendmail, Simple Mail Transfer Protocol ([SMTP](#)), Mail) e altro
- Librerie per la manipolazione delle immagini (taglio, ridimensionamento, rotazione, ecc) Supporto per GD Graphics Library ([GD](#)), ImageMagick e NetPBM
- Classe per l'upload dei file
- Classe per l'File Transport Protocol ([FTP](#))
- Localizzazione
- Paginazione
- Crittografia dei dati

- Benchmarking
- Full page caching
- Registrazione degli errori
- Profilazione
- Classe Calendario
- Classe User Agent
- Classe Zip Encoding
- Classe Engine dei template
- Classe Trackback
- Libreria [XML-RPC](#)
- Classe Unit Testing
- [URL](#) amichevoli per i motori di ricerca
- Supporto flessibile per il routing Uniform Resource Identifier ([URI](#))
- Supporto per gli Hook e le Estensioni delle classi
- Vasta libreria di funzioni Helper

## 1.2 DESIGN E ARCHITETTURA

L'obiettivo di CodeIgniter è quello di massimizzare le performance, la qualità e la flessibilità, pur impegnando il minor numero di risorse possibile lato server. Per raggiungere questo obiettivo ci si è impegnati nel benchmark, re-factoring e nella semplificazione di ogni fase del processo di sviluppo, tralasciando ogni aspetto non utile a tale scopo. Dal punto di vista tecnico, CodeIgniter è stato sviluppato con i seguenti obiettivi:

- Istanze dinamiche. I componenti sono caricati e le routine vengono eseguite solo quando richieste, piuttosto che globalmente. Nessuna risorsa viene allocata in anticipo, se non quando effettivamente necessaria. Per questo motivo, il sistema è molto leggero. Gli eventi sono innescati dalle richieste Hypertext Transfer Protocol ([HTTP](#)) mentre i Controller e le Viste determineranno ciò che verrà prodotto e visualizzato
- Loose Coupling. Questo termine, che si può tradurre in “legami deboli”, viene usato per indicare il grado con cui i componenti di un sistema sono legati l'uno all'altro. Minore è il numero di componenti che dipendono da altri, maggiore sarà la riusabilità e la flessibilità del sistema. L'obiettivo è quello di avere un sistema con un grado di Coupling minimo



- **Component Singularity.** La Singularity è il grado con cui i componenti assumono uno scopo ben preciso. In CodeIgniter ogni classe e ogni sua funzione sono altamente indipendenti per consentire la massima utilità

In definitiva si può affermare che CodeIgniter è un sistema istanziato dinamicamente, con un grado di coupling basso e un alto grado di specificità. Esso si sforza di essere semplice, flessibile e potente in un package dalle minime dimensioni.

## LICENZA

CodeIgniter è gratuito, o meglio open source, ciò significa che potrà essere utilizzato e modificato liberamente sulla base delle proprie esigenze; i soli vincoli da rispettare sono quelli relativi alla *licenza Apache/BSD-style* che protegge il prodotto. Maggiori informazioni sono reperibili su [https://it.wikipedia.org/wiki/Licenze\\_BSD](https://it.wikipedia.org/wiki/Licenze_BSD) e [https://it.wikipedia.org/wiki/Licenza\\_Apache](https://it.wikipedia.org/wiki/Licenza_Apache).

## Autori

CodeIgniter è stato sviluppato originariamente da *Rick Ellis* (CEO di EllisLab, Inc) ed una prima versione è stata resa disponibile nel febbraio del 2006. Il framework è stato scritto per ottenere le migliori prestazioni con diverse classi di librerie, helper, e sottosistemi presi in prestito dal codice base di ExpressionEngine. Il suo attuale sviluppo e mantenimento avviene grazie al team di *ExpressionEngine* e a tutti i volontari selezionati con cura dal *Team Reactor*, che contribuiscono direttamente alla sua evoluzione. Gli autori ci tengono a ringraziare *Ruby on Rails* per averli ispirati nella realizzazione di un tool di sviluppo [PHP](#), e aver contribuito a diffondere nella comunità web la coscienza del framework. Chi volesse apportare dei cambiamenti al codice sorgente e contribuire al miglioramento di CodeIgniter troverà nel repository GitHub (<https://github.com/EllisLab/CodeIgniter/>) tutto il necessario.

### Definizione 1: Ruby on Rails

<b>Ruby on Rails</b>	Spesso chiamato RoR o semplicemente Rails, è un framework open source per applicazioni web scritto in Ruby da David Heinemeier Hansson per conto della 37signals la cui architettura è ispirata al paradigma Model-View-Controller. Sito ufficiale: <a href="http://rubyonrails.org/">http://rubyonrails.org/</a>
----------------------	---

## 1.3 RIEPILOGO

È stata fornita una panoramica sulle peculiarità di un framework e sui vantaggi che la sua adozione fornisce. Per progetti che non prevedono connessioni ad una base

di dati e che sono costituiti da poche pagine, magari statiche, l'utilizzo di CodeIgniter, ma anche di un qualsiasi altro framework, risulta inutilmente dispendioso. In ogni caso, padroneggiare un tool di sviluppo espande notevolmente le proprie conoscenze informatiche, incrementa l'efficienza, la sicurezza e la solidità del proprio lavoro.

+



## 2 | INSTALLARE CODEIGNITER

Questo capitolo spiega in maniera concisa e si spera chiara, come installare CodeIgniter a prescindere dal sistema operativo utilizzato. La guida è stata realizzata basandosi sulla versione 2.2.0 disponibile sul sito ufficiale: <http://ellislab.com/codeigniter>. Il framework per poter funzionare correttamente deve essere utilizzato su una macchina con installato [PHP](#) e un server web come Apache.

### 2.1 SERVER LOCALE E REMOTO

Esistono due soluzioni per creare un ambiente idoneo allo sviluppo: realizzare un server web locale o uno remoto. Le due soluzioni mettono a disposizione ambedue un sistema caratterizzato dagli stessi software di base come Apache, Oracle MySQL ([MySQL](#)) e [PHP](#), ma differiscono per la dislocazione delle macchine. Un server locale è una macchina che fisicamente è situata nelle vicinanze dello sviluppatore e che sovente si identifica con il sistema utilizzato quotidianamente. Diverse soluzioni software permettono infatti di realizzare un sistema casalingo per lo sviluppo e la fruizione dei contenuti web. Un server remoto, come si evince dalla parola stessa, è una macchina non disponibile fisicamente. Solitamente essa viene messa a disposizione da apposite aziende commerciali sotto forma di hosting o housing con pacchetti che variano dalle poche decine di euro alle migliaia di euro l'anno. L'oscillazione dipende dalle funzionalità del server, dalla sua potenza e dall'assistenza fornita dall'azienda. L'utilizzo di un server remoto non è però strettamente necessario per realizzare un ambiente votato allo sviluppo, pertanto si esamineranno le soluzioni più immediate per installare e utilizzare un server locale.

### REQUISITI MINIMI E INSTALLAZIONE

Le richieste di CodeIgniter sono tutto sommato modeste per quanto riguarda l'hardware, mentre lato software, la compatibilità con [PHP](#) si ha dalla versione 5.1.6 o più recente. Un database è richiesto per lo sviluppo della maggior parte delle applicazioni web, mentre le basi di dati attualmente supportate sono: MySQL (4.1+), MySQLi, MS SQL, Postgres, Oracle, SQLite, e ODBC.

#### 2.1.1 Installazione per Windows

Sotto il sistema Microsoft esistono numerose soluzioni denominate Windows-Apache-MySQL,P (per PHP, PERL, Python) ([WAMP](#)): vere e proprie piattaforme software di sviluppo web/database basate su Apache, MySQL e PHP, Perl, Python o altri linguaggi di scripting. Una soluzione pronta all'uso è *Wamp Stack* della Bitnami

che nel sito ufficiale <https://bitnami.com/stack/wamp> mette a disposizione un pacchetto autoinstallante e assolutamente amichevole. Oltre a comprendere tutto l'occorrente per avere una macchina locale dedicata allo sviluppo con i principali framework, comprende altri componenti come phpMyAdmin (per la gestione del database), OpenSSL, PEAR, PECL e altri importanti software sul cui scopo, per ora, si potrà sorvolare.

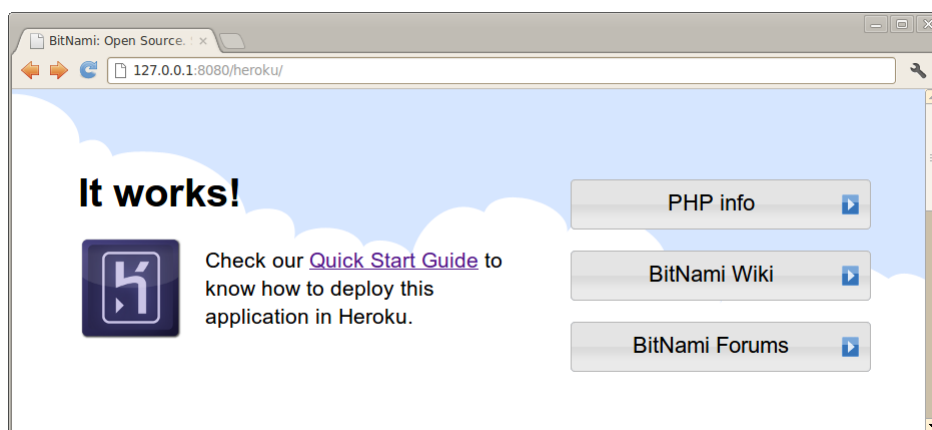


Fig. 1: Bitnami: it's work!

Un'altra soluzione molto utilizzata è *EasyPHP* <http://www.easyphp.org/> per creare in pochi passi un ambiente di sviluppo ottimizzato per la programmazione e il debug delle applicazioni. Un aspetto che potrebbe essere rilevante per alcuni utenti, è che EasyPHP è portatile: basta una chiavetta USB o un hard disk portatile per trasformare ogni macchina, all'occorrenza, in un server web pronto all'uso. Chi fosse particolarmente curioso può trovare altre soluzioni, semplicemente effettuando una ricerca in Rete con il termine **WAMP**.



Fig. 2: EasyPHP

## 2.1.2 Installazione per Mac OS

Molte sono anche le soluzioni “pronte all’uso” per Mac OS X (ma anche versioni precedenti del sistema operativo Apple), definite semplicemente MacOS-Apache-MySQL,P (per PHP, PERL, Python) ([MAMP](#)). Tra i tanti software citiamo la proposta dell’azienda *appsolute* che fornisce un pacchetto liberamente utilizzabile ed una versione PRO a pagamento dotata di maggiori funzionalità. Per le nostre esigenze, la versione gratuita sarà più che sufficiente: <http://www.mamp.info/en/>

## 2.1.3 installazione per GNU/Linux

Questo è il sistema di riferimento per la produzione. Nonostante non tutti abbiano le conoscenze e la pazienza per padroneggiare un sistema operativo differente da quello solitamente utilizzato, vale la pena prima o poi muovere i passi nell’affascinante mondo del software libero. La procedura di installazione varia a seconda della distribuzione utilizzata: qui prenderemo in esame quella per il sistema Ubuntu, una delle maggiori distruzioni per diffusione tra utenti consumer. Esistono diverse modalità per l’installazione di Linux/GNU-Apache-MySQL,P (per PHP, PERL, Python) ([LAMP](#)) e per tutte è obbligatorio avere i permessi di amministratore Super User ([su](#)). Esamineremo alcune procedure:

**MEDIANTE TASKSEL** Probabilmente la soluzione più veloce. Aprendo il terminale bash, si esegua il comando per installare “tasksel”:

```
sudo apt-get install tasksel
```

Successivamente si esegua l’istruzione:

```
sudo tasksel
```

Dal menu proposto, si selezionerà l’installazione “Lamp server”. Successivamente si potranno installare altri utili moduli come “PhpMyAdmin” con:

```
sudo apt-get install phpmyadmin
```

**MEDIANTE SYNAPTIC** É un ottimo software center per gestire le applicazioni di sistema: lo si trova su Sistema > Amministrazione > Synaptic Package Manager; se non presente nella distruzione GNU/Linux, lo si può installare con il comando da terminale:

```
sudo apt-get install synaptic
```

Successivamente si apra il programma dotato di interfaccia grafica e si selezioni dal menu “Modifica” la voce “Marca i pacchetti per attività”. Si aprirà una finestra in cui si dovrà selezionare l’opzione “LAMP server”; infine, si confermi l’operazione premendo su “OK” (si veda la figura a pagina 21).

#### 2.1.4 Configurare e verificare il server

In fase di installazione si dovranno apportare alcune modifiche e verificare alla fine il corretto funzionamento del server. Esaminiamole nel dettaglio:

1. durante l’installazione verrà chiesta la password per MySQL (si veda l’immagine a pagina 22). Si inserisca una password (non dimenticatela!), e si tenga presente che il nome utente/login è “root”, infine si concluda l’installazione.
2. a fine installazione si verifichi la corretta presenza sul proprio sistema di Apache scrivendo sul terminale bash:

```
sudo /etc/init.d/apache2 restart
```

Se apro un browser qualsiasi e digitando nella barra degli indirizzi:

```
http://localhost
```

compare una schermata di benvenuto o un messaggio “it Works!”, significa che l’installazione di Apache è andata a buon fine (altrimenti si veda a pagina 18).

3. Per testare l’installazione del [PHP](#) si crei un file di nome **info.php** con il seguente codice:

```
<?
    phpinfo();
?>
```

Si salvi il file e lo si copi nella directory del server web locale (che su sistemi GNU/Linux si trova generalmente nel percorso **/var/www/**).

Nella barra degli indirizzi del browser si digiti:

```
http://localhost/info.php
```

Se anche [PHP](#) è presente nel vostro sistema, apparirà una schermata con il riepilogo dei parametri principali (si veda l’immagine a pagina 22).



Tabella 2: Database supportati da CodeIgniter

Database	Versione
MySQL	4.1
MySQLi	
MS SQL	
Postgres	
Oracle	
SQLite	
ODBC	

## 2.2 INSTALLARE CODEIGNITER

Una volta scaricato il framework sotto forma di file compresso lo si decompila nella directory del proprio server web. Fatto questo, nella cartella sarà presente il file **index.php** che, insieme alle directory **System** e **Application**, costituiranno l'ossatura di CodeIgniter. Per verificare il corretto funzionamento del framework (se lo avete installato in una cartella dal nome **codeigniter**), digitate:

```
http://localhost/index.php/codeigniter/
```

Bene, ora si è “quasi” pronti ad assaporare le potenzialità di questo potente tool di sviluppo.

## 2.3 CONFIGURAZIONE

Gli utenti esperti potranno configurare alcuni aspetti di CodeIgniter aprendo con un editor di testo i seguenti due file:

- **/application/config/config.php**. Qui si dovrà settare il nostro [URL](#) di base ed eventualmente impostare altri parametri come la chiave di criptazione (encryption key), il formato della data, il linguaggio o i parametri [URI](#).
- **/application/config/database.php**. In questo file si inseriranno i dati di accesso al database, se si intende utilizzarne uno, ovviamente.

Una opzione degna di nota nel **config.php** è quella che imposta l'[URL](#) di base del proprio server remoto. Se l'indirizzo di primo livello è per esempio <http://esempio.com> e CodeIgniter è installato in una sottodirectory chiamata “mioProgetto”, è possibile definire la radice dell'intero progetto con il seguente parametro:

```
$config['base_url'] = "http://esempio.com/mioProgetto";
```

Se invece si lavora in un ambiente locale il parametro dovrà tenere conto dell'URL di base (localhost oppure 127.0.0.1):

```
$config['base_url'] = "http://localhost/mioProgetto";
```

oppure

```
$config['base_url'] = "http://127.0.0.1/mioProgetto";
```

Nota: i termini localhost e 127.0.0.1 sono equivalenti e richiamano ambedue il server locale utilizzato in fase di sviluppo.

Comunque chi non avesse esperienza nella configurazione di un server web, può tranquillamente evitare per il momento di mettere mano ai preziosi file di configurazione.

Se la nostra applicazione dovrà collegarsi ad una base di dati sarà necessario configurare anche il file **database.php** localizzato al percorso **/application/config/database.php**

Qui sarà possibile impostare le seguenti voci:

```
['hostname']: il nome di host del database server;  
['username']: lo username per la connessione al database;  
['password']: la password per la connessione al database;  
['database']: il nome del database che si desidera selezionare;  
['dbdriver']: il tipo di database che si desidera utilizzare;  
['dbprefix']: il prefisso da associare opzionalmente al nome delle tabelle;  
['pconnect']: valori TRUE o FALSE per connessioni persistenti o meno;  
['db_debug']: valori TRUE o FALSE per visualizzare gli errori del database;  
['cache_on']: valori TRUE o FALSE per abilitare o meno la cache delle query;  
['cachedir']: consente di definire il percorso alla cache delle query;  
['char_set']: definisce il set di caratteri per le comunicazioni al database;  
['dbcollat']: definisce la collation di caratteri per le comunicazioni al DB.
```

Se per esempio si dispone di un database di nome **test** a cui è possibile accedere con i dati di login **mioNome** e password **1234**, i dati nel file **database.php** dovranno essere così impostati:

```
$db['default']['hostname'] = 'localhost';
$db['default']['username'] = 'mioNome';
$db['default']['password'] = '1234';
$db['default']['database'] = 'test';
$db['default']['dbdriver'] = 'mysql';
```

L'ultima riga indica che stiamo utilizzando un database di tipo MySQL.

## 2.4 PRIMI ACCORGIMENTI PER LA SICUREZZA

Se si desidera incrementare la sicurezza nascondendo il percorso dei propri file di CodeIgniter, è possibile rinominare le directory **System** e **Application** per renderle meno evidenti; successivamente si apra il file **index.php** e si configurino opportunamente le variabili `$system_path` e `$application_folder` inserendo i nomi delle directory precedentemente modificati.

Se si è particolarmente diffidenti (e questo è un bene) è consigliabile rendere inaccessibili sia la directory **System** che quella **Application** ad un browser web. Normalmente i file **.htaccess** sono inclusi in ogni directory per prevenire l'accesso diretto, ma è consigliato rimuoverli dall'accesso pubblico quando non siano strettamente necessari o la configurazione del server web cambia.

Una ulteriore misura da adottare in fase di produzione è quella di disabilitare gli avvisi che mostrano gli errori di **PHP**, e qualsiasi altro messaggio che riporti informazioni non rilevanti per l'utente. In CodeIgniter questo può essere fatto settando opportunamente le costanti **ENVIRONMENT** sempre all'interno di **index.php**. I valori selezionabili sono: `development`, `testing` e `production`.

Per il momento, non ci si preoccupi troppo poiché le misure da adottare per la sicurezza del proprio progetto saranno affrontate successivamente nel capitolo [6.10 a pagina 107](#).

### Definizione 3: Gli insiemi di caratteri

<b>character set</b>	Sono i diversi sistemi con cui i caratteri alfanumerici, i segni di punteggiatura e qualsiasi simboli visualizzabile su un computer vengono memorizzati in un valore binario.
<b>collation</b>	Ad ogni character set è associata una o più collation, che rappresentano i modi possibili di confrontare le stringhe di caratteri di uno stesso insieme di caratteri.

## 2.5 SIAMO PRONTI

A questo punto siete pronti per visualizzare la nostra prima schermata di benvenuto. Se tutto è filato liscio, inserendo nel browser il seguente indirizzo:

```
http://127.0.0.1/index.php/codeigniter/
```

dove **codeigniter** è il nome dato alla directory in cui si trova il framework installato nel server web, comparirà la schermata di benvenuto mostrata a pagina 22.

CodeIgniter ci saluta, invitandoci a modificare la pagina introduttiva **welcome.php** che si trova al percorso `application/views/welcome_message.php` e il corrispondente Controller situato su `application/controllers/welcome.php`. Al completare il messaggio vi è un rimando alla pratica guida, che si può consultare anche in locale nella directory `user_guide/`.

Per il momento, evitiamo di seguire l'invito di CodeIgniter e di apportare qualsiasi modifica: dobbiamo ancora impratichirci con i concetti base.

## 2.6 RISOLUZIONE DEI PROBLEMI

È assai improbabile che la fase di installazione non porti a qualche inconveniente, soprattutto se non si ha molta dimestichezza con la configurazione di un server. Qui di seguito sono passati in rassegna alcuni dei problemi più comuni nell'installazione e configurazione di CodeIgniter.

**PAGINE NON CARICATE** se si scopre che nonostante ciò che inseriamo nel nostro [URL](#) del browser, solo la pagina di default viene caricata, questo potrebbe significare che il server non supporta la variabile `PATH_INFO` necessaria per mostrare [URL](#) amichevoli ai motori di ricerca. Come primo passo si apra il file **config.php** al percorso `/application/config/` quindi si cerchino le informazioni sul protocollo [URI](#). È consigliabile provare un paio di impostazioni alternative. Se ancora non funziona dopo aver provato questo metodo, si dovrà forzare CodeIgniter per aggiungere un punto interrogativo agli [URL](#). Per fare questo, si apra il **config.php** al percorso `/application/config/`, e si modifichi:

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

con:

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

**PERMESSI** non si riesce a copiare CodeIgniter nella root del proprio server web? Potrebbe essere un problema legato ai permessi. Solitamente solo l'ammini-

stratore può scrivere/modificare i file in questa cartella, quindi una soluzione veloce è quella di copiare i file con i permessi di super user, facendo precedere ogni vostro comando dalla stringa `su`. Per esempio, una volta scaricato e decompresso il file zip che contiene CodeIgniter, questo può essere copiato nella directory del server con:

```
su cp codeigniter /var/www/MioProgetto/
```

Con il comando `su` si acquisiranno per breve tempo i “poteri” di amministratore, ma ovviamente si dovrà conoscere l’apposita password. Un’altra soluzione, meno consigliata sotto il profilo della sicurezza, è quella di modificare i permessi di scrittura/lettura della directory del server web. In questo caso, il comando “`su chmod -R 777`”, eseguito nel percorso `/var/www/` permetterà a chiunque di scrivere e modificare i file del vostro progetto: comodo ma rischioso se il computer è accessibile ad altri utenti.

**ATTENZIONE AI DETTAGLI** verificate il percorso inserito nel browser. A volte basta un carattere in più (o in meno), una maiuscola nell’indirizzo per visualizzare un errore “404 not found” e farci sprofondare nella disperazione più totale.

**SERVIZI WEB** se digitando `http://127.0.0.1` la pagina che vi si presenta è un desolante “404 not found” è probabile che non siano stati avviati i servizi web per far funzionare correttamente il vostro server locale. Provate a digitare:

```
sudo service mysqld start
```

e successivamente:

```
sudo service httpd start
```

Collegatevi nuovamente all’indirizzo `http://127.0.0.1` e incrociate le dita.

**TROVARE LA SOLUZIONE DA SOLI** leggete con attenzione i messaggi di errore. Se legati a CodeIgniter sono spesso accompagnati da informazioni che ne permettono l’individuazione e la risoluzione: basterà una ricerca in rete, magari nel forum della community per venirne a capo facilmente.

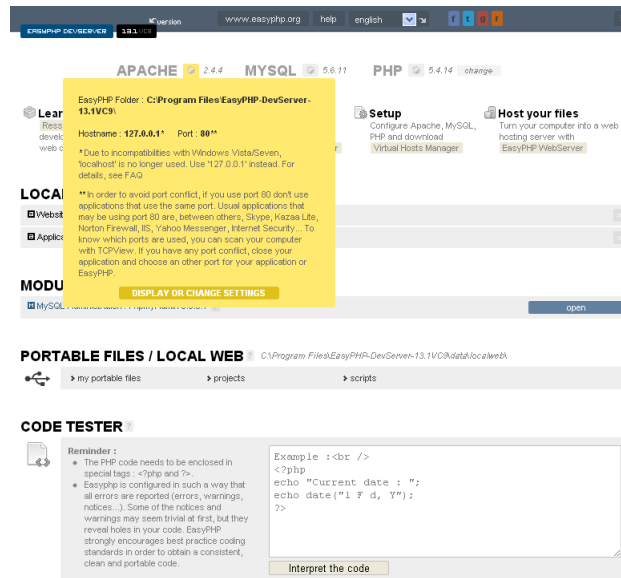


Fig. 3: EasyPHP: code tester

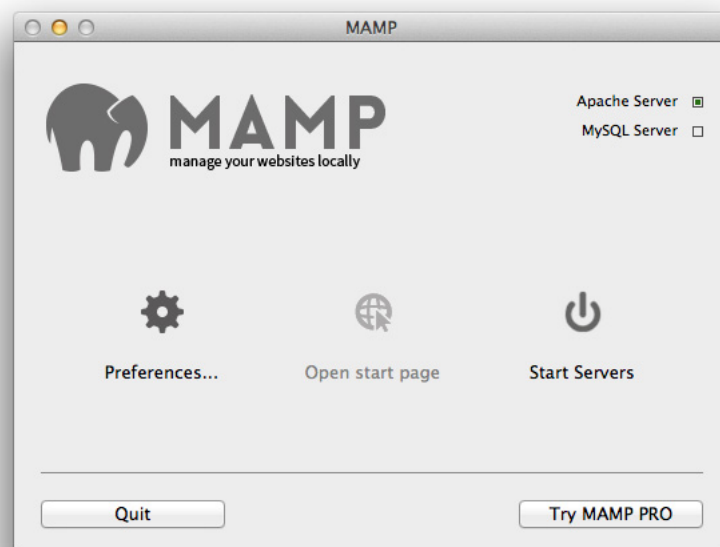


Fig. 4: MAMP - schermata iniziale

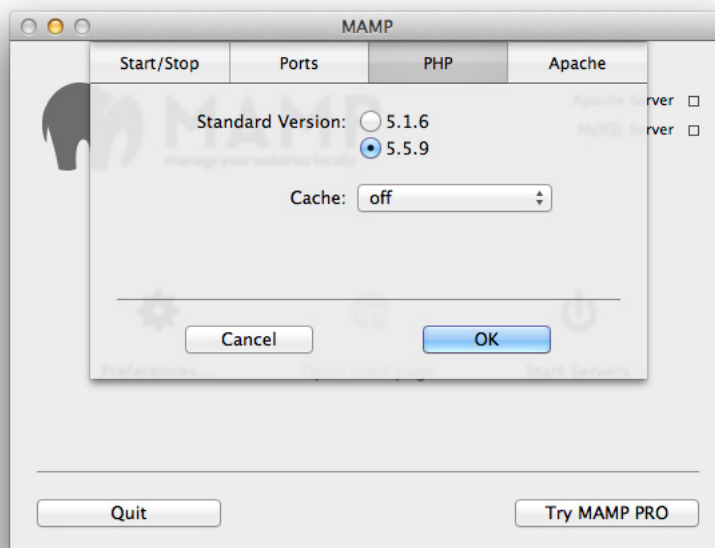


Fig. 5: MAMP - PHP

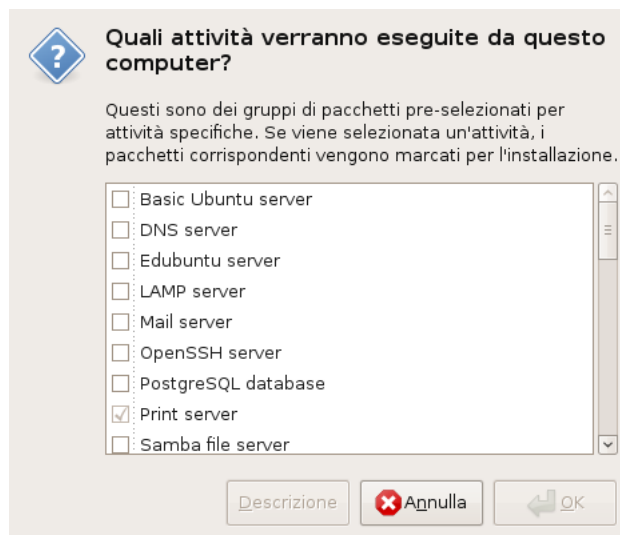


Fig. 6: Synaptic - installazione LAMP

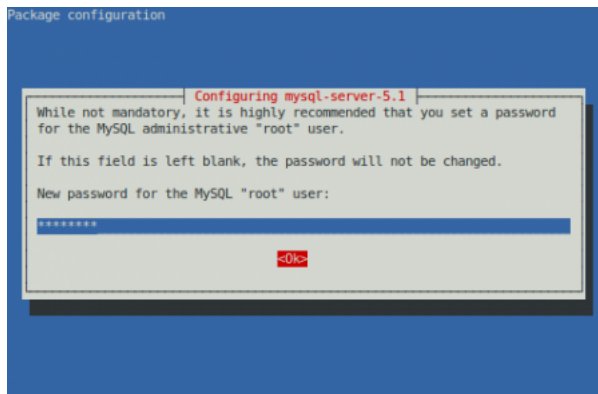


Fig. 7: MySQL ha bisogno di una password



Fig. 8: PHP: info

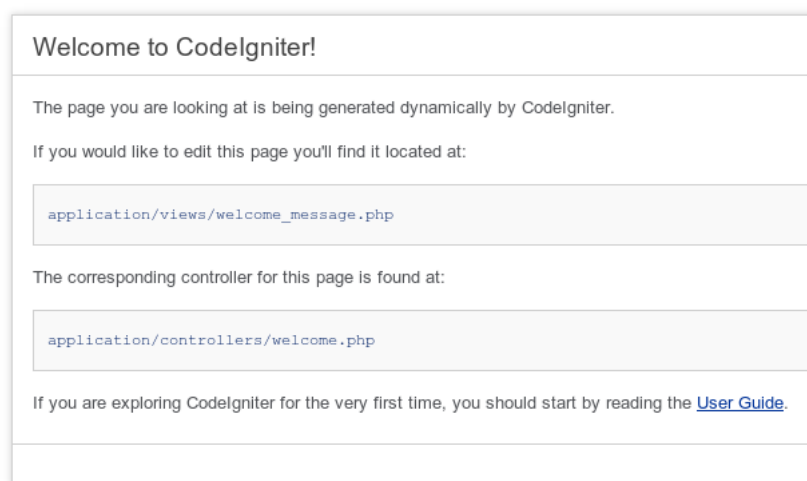


Fig. 9: Il benvenuto di CodeIgniter



# 3 | I MODERNI PATTERN DI SVILUPPO

Lo scopo di questo capitolo è quello di introdurre ai principi base del framework CodeIgniter e all'architettura [MVC](#). Verrà illustrato come una applicazione viene costruita mattone dopo mattone con questo potente e flessibile strumento. Non ci si lasci scoraggiare dalle difficoltà iniziali perché man mano che si procederà nella lettura, molti termini sconosciuti diverranno sempre più familiari. Ad ogni modo, le caratteristiche salienti di CodeIgniter saranno introdotte gradualmente e con profusione di spiegazioni e dettagli al fine di facilitare l'apprendimento. I prossimi capitoli saranno dedicati all'implementazione di una semplice applicazione per visualizzare un elenco di notizie: inizialmente ci si concentrerà su una piattaforma software elementare con semplici pagine statiche e poche informazioni preconfezionate. Successivamente, man mano che si prenderà familiarità con il framework, verranno aggiunte nuove funzionalità come il caricamento dinamico delle informazioni tramite un database e la loro memorizzazione mediante un pratico form. Verranno esaminati i seguenti argomenti:

- le basi Model-View-Controller
- le basi del Routing
- validazione dei form
- esecuzione delle query di database usando "Active Record"

L'intero tutorial è diviso in più capitoli, ognuno dei quali illustrerà distinte parti delle funzionalità del framework CodeIgniter e del pattern [MVC](#).

## 3.1 SEPARARE PRESENTAZIONE E LOGICA

Questo è un aspetto cardine della moderna programmazione che vogliamo introdurre sin da subito. Converrete che l'ordine, in qualsiasi contesto, è un aspetto positivo, no? In passato questo non era facilmente raggiungibile, in quanto sia il codice logico (adibito alle funzionalità del software) che quello di presentazione (per la rappresentazione grafica) venivano inseriti nello stesso file, con la generazione così di codice caotico, spesso prodotto da più programmatori. Un grafico, per esempio, deve barcamenarsi con porzioni di codice a lui sconosciute: basta un attimo di disattenzione come cancellare inavvertitamente un punto e virgola per introdurre degli errori logici nel codice. E questi, per quanto banali, non sempre risultano facili da scovare. Il programmatore dal suo canto deve gestire un codice in cui "altri" mettono le mani (aspetto che già di suo alimenta le sue già notevoli paranoie).

La pratica di inserire nello stesso file codici dai diversi scopi (di solito presentazione e logica) viene definita come "inline code". Un esempio pratico è dato da

quei file di codice in cui è presente sia il **PHP** (per interfacciarsi con il database) che l'**HTML** (adibito alla visualizzazione delle pagine web). Oggi per fortuna la situazione sta cambiando velocemente e forse l'esempio più conosciuto di separazione tra codice e presentazione è dato proprio dall'uso congiunto di **HTML** e Cascading Style Sheets (**CSS**). Il primo viene adibito alla definizione delle informazioni e al loro valore semantic, mentre il secondo definisce la presentazione che verrà mostrata all'utilizzatore finale. Sì, come è lecito supporre da quanto appena detto, il codice **HTML** e quello **CSS** si trovano in due file distinti. Vediamo quali sono i vantaggi di questo approccio.

**RIUSABILITÀ** È possibile usare nuovamente il codice di presentazione in cui sono definiti, per esempio, i colori, il tipo di carattere e la grandezza del font. Basta copiare il file in un nuovo progetto e potrete dire al vostro capo che state sudando "nuovamente" le proverbiali sette camicie, mentre impunemente sorseggiate un aperitivo sulla spiaggia.

**ORDINE** File diversi per scopi diversi, rendono la vita migliore di ognuno di noi. Se volete introdurre una nuova funzionalità nel progetto, non avrete la necessità di toccare i template grafici, giusto?

**MENO LITIGI** Non si dovranno più guardare in cagnesco i colleghi che hanno accesso al nostro prezioso codice. Ognuno lavorerà sul proprio file (o quasi) riducendo drasticamente la possibilità di errori difficili da scovare.

**RAPIDITÀ** Le fasi di sviluppo e soprattutto di debug diverranno più rapide.

Separando la presentazione, ovvero la grafica di un software dalla logica che ne espleta le funzionalità, si sono già avuti dei grandi vantaggi, non credete?

## 3.2 INIZIARE CON CODEIGNITER

Ogni software ha bisogno di un certo impegno per essere padroneggiato. Questa guida vuole minimizzare la curva di apprendimento in modo da rendere il processo di apprendimento di CodeIgniter il più amichevole possibile.

Il primo passo è installare CodeIgniter e seguire i consigli di questa guida (si veda il capitolo [2.1 a pagina 11](#)). Una grande risorsa di informazioni e richieste di aiuto è data dal Forum della Comunità a cui sottoporre domande o problemi <http://ellislab.com/codeigniter/>.

## 3.3 MODEL-VIEW-CONTROLLER

Questo capitolo introduce il paradigma più importante alla base dei framework come CodeIgniter e della sviluppo di software professionale. Non si tratta di concetti difficili da apprendere, ma è bene prestare molta attenzione. Ricordate quando si è parlato di separare la presentazione dalla logica? Bene il discorso del pattern **MVC** ne è solo una rivisitazione ampliata. Lo scopo ultimo è sempre lo stesso, separare i diversi aspetti di un progetto in modo da facilitarne la riusabilità e la manutenzione

del codice e, aspetto da non sottovalutare, la collaborazione tra più sviluppatori. Esaminiamo prima di tutte queste tre paroline magiche.

**MODEL** Il Modello rappresenta la struttura dei dati. Solitamente in esso sono presenti le classi con le funzioni per recuperare, inserire e aggiornare le informazioni sul proprio database.

**VIEW** La Vista, come è intuibile dal nome, è dedicata alla rappresentazione grafica delle informazioni. Si desidera formattare il testo con un determinato colore? Ingrandirlo? Bene, il codice che si occuperà dell'aspetto puramente grafico verrà inserito in questa sezione.

**CONTROLLER** Il Controller (no, ci spiace, non lo tradurremo in controllore) è un nuovo elemento, che ancora non è stato introdotto nel nostro discorso. Esso si occupa, al pari di un router, di instradare una richiesta [HTTP](#) verso una determinata funzione. Un esempio renderà più chiaro il concetto: ogni volta che si digita un indirizzo nel proprio browser si effettua una richiesta del tipo: "caro browser, caricami la pagina [www.miourl.com](#)". Il browser analizza la richiesta e se ammissibile (se il sito esiste per esempio), visualizza la pagina desiderata. Questo è il compito di un Controller: analizzare la richiesta e instradarla verso il percorso stabilito.

Questi tre elementi, trattati per ora molto superficialmente, sono peculiari a quasi tutti i moderni e più evoluti framework web. Padroneggiarne la filosofia è prioritario, ma non ci si preoccupi: con i prossimi capitoli ci si ritroverà ad utilizzarli senza quasi accorgersene.

### 3.4 ARCHITETTURA DEL PROGETTO

Con questo capitolo ci prenderemo una veloce pausa dalle (poche) nozioni sin qui apprese su CodeIgniter. Più precisamente ci si concentrerà sulle regole per realizzare un buon progetto, anche se la definizione “regole” non è forse il termine più corretto, e suona come qualcosa di terribilmente severo e rigido. In informatica si preferisce parlare di “best practice”, ovvero una raccolta di esperienze significative, raccolte in criteri a cui è bene attenersi per sviluppare un prefissato progetto. Ma perché abbiamo introdotto questo argomento proprio ora che il progetto incominciava ad entrare nel vivo? In realtà, la domanda corretta sarebbe dovuta essere “perché non si è parlato di “architettura” software ben prima!”. Non ci si spaventi comunque, perché molte delle best practice comunemente adottate sono decisamente intuitive da padroneggiare.

È importante ora porre l’attenzione sull’analisi di un progetto a prescindere da quale esso sia: un software web o un ponte sopra un fiume, questo non ha alcuna importanza. Uno degli aspetti più sottovalutati è lo studio a monte di un progetto, quello che in ingegneria del software viene comunemente chiamato come “analisi dei requisiti” e che costituisce una materia di studio a se stante, rilevante quanto la stessa fase di sviluppo.

Incominciare ad abbozzare un progetto scrivendo direttamente il codice è uno dei peggiori errori che si possa infatti commettere. Soventemente questo entusiasmo si pagherà dovendo rimettere mano al progetto, modificando piccole e continue caratteristiche, se non, nel peggiore dei casi, ricominciando il suo sviluppo da zero. Quindi non ci senta poco professionali se ci si arma di una penna e un block notes per definire con dovizia gli aspetti peculiari del proprio progetto progetto.

#### PER CHI STIAMO SVILUPPANDO?

Domanda banale, ma che influenza direttamente la fase di progettazione. Stiamo sviluppando un software per passione e per mettere alla prova le nostre capacità? Bene, in questo caso, si può affrontare meno rigidamente la fase di progettazione che comunque riveste un ruolo sempre di primo piano: spendete, qualche ora (o giorni per i progetti più complessi) e verrete ripagati da una fase di sviluppo con pochi intoppi, agile e appagante. In caso contrario, non dite che non eravate stati avvisati.

Diversa è la definizione dei requisiti software quando si vuole realizzare un prodotto commerciale commissionato da una azienda. In questo caso sono molti i parametri da tenere in considerazione, e non sempre vale l’equazione che accomuna il “prodotto con più funzioni” a “quanto richiesto dall’azienda”. Vi sembra strano? Non proprio se ci si riflette bene. Un prodotto con molte funzioni è anche più complesso da apprendere per chi ci lavora e questo può portare a dover spendere risorse per addestrare il personale con dispendio di tempo e denaro. Ottimizzare le performance di un software web quando questo non sarà mai posto sotto carico eccessivo, è un altro dispendio inutile di risorse. Insomma, comprendere cosa desidera l’azienda che ci commissiona un lavoro è fondamentale quanto sviluppare un buon codice.

Questo porta ad introdurre il concetto di “utile” che talvolta può risultare soggettivo. Il giudizio su alcuni parametri, come la presentazione, l’efficienza o l’usabilità

dell'interfaccia, varia a seconda del punto vista degli attori coinvolti (chi utilizza il prodotto). In generale si può affermare che più funzionalità si introducono nel software, maggiore risulterà la sua utilità. È però opportuno considerare che una piattaforma di sviluppo e gestione dei contenuti sarà appesantita da funzionalità non strettamente necessarie, andando ad inficiare sulle prestazioni, sull'usabilità dell'interfaccia e sulla riusabilità del codice.

## ANALISI DEI REQUISITI

La definizione degli obiettivi deve essere raggiunta insieme a chi commissiona il lavoro. Esistono molti metodi per individuare i "requisiti utente", ovvero le caratteristiche imprescindibili del software rivolte a chi lo utilizzerà. Il metodo più semplice per definirli è l'uso del *linguaggio naturale* con l'end-user, ovvero il cliente finale: insomma, una chiacchierata amichevole, ma puntigliosa permetterà di comprendere chiaramente le specifiche esigenze e le priorità nello sviluppo. Perciò chiedere quali aspetti si dovrebbero introdurre o migliorare, può essere veloce ed efficiente quanto la stesura di diagrammi funzionali tanto comuni nell'ingegneria del software.

Al termine di questa fase, sarà più facile definire i requisiti funzionali, dove con questo termine si vuole intendere una descrizione accurata delle funzionalità del software, in termini di servizi offerti, di come il sistema reagisce a specifici tipi di input e di come si comporta in situazioni particolari.

Qui viene mostrato un semplice esempio di requisiti funzionali espressi da una ipotetica azienda che si suppone desideri un tool per la catalogazione delle immagini da affiancare alla propria piattaforma editoriale di contenuti web.

1. il prodotto sviluppato deve risultare indipendente da software proprietari e dalla piattaforma editoriale utilizzata, sia per quanto riguarda la presentazione che le funzionalità annesse. La sua implementazione deve poter essere, in ogni momento esclusa, con impatti nulli sulle funzionalità del software che gestisce i contenuti editoriali;
2. il software deve permettere all'utilizzatore di "ricercare, catalogare e modificare" le informazioni sensibili riguardanti le immagini classificate in categorie;
3. requisito fondamentale è l'interoperabilità con la piattaforma editoriale, su cui il progetto deve appoggiarsi soprattutto per la struttura logica con cui vengono gestite le informazioni memorizzate nella base di dati del sito aziendale;
4. le informazioni sensibili riguardanti le categorie delle immagini devono risultare aggiornabili;
5. l'accesso agli strumenti di editing deve essere concesso ad un utente dotato di relativi permessi (utente administrator);
6. è richiesto il miglioramento della granularità delle operazioni già presenti nella piattaforma editoriale;

Questo è solo un esempio, che non ha la pretesa di essere esaustivo. Infatti molti punti meriterebbero un approfondimento per definire ulteriori dettagli, come il numero delle categorie, la loro definizione, l'interfaccia utente e molto altro. Definire comunque un primo elenco di requisiti macroscopici per poi realizzarne di nuovi, più specifici e dettagliati è una prassi accettabile.

## PROGRESSIVE ENHANCEMENT

Una volta che si avrà un quadro preciso di cosa si deve sviluppare e in quale modo, saranno chiari anche i tempi e i costi di sviluppo. Nulla vieterà di apportare dei cambiamenti al prodotto (in comune accordo con l'azienda che l'ha commissionato ovviamente), ma qualsiasi intoppo nascerà in seguito, verrà senz'altro mitigato dallo studio condotto nella fase preliminare.

Nonostante tutti gli studi condotti e le buone intenzioni, sarà comunque arduo non commettere qualche errore di progettazione a cui dover poi porre rimedio. Per limitare le correzioni nella fase di sviluppo, ma anche successivamente, si è diffuso in tempi recenti la "best practice" definita come "progressive enhancement". Secondo questa metodologia di sviluppo, comune alla realizzazione dei software più complessi così come ad una semplice pagina web, si intende comunemente uno sviluppo incrementale del progetto sviluppato, definendo inizialmente le funzioni basilari per poi potenziarle ed estenderle. Questa metodologia non prevede però banalmente solo una progettazione secondo il modello "top-down" (dall'alto verso il basso) con cui si formula una visione generale del sistema, senza scendere nello specifico dettaglio delle sue parti. Il progressive enhancement, che comunque molto condivide con il modello top-down, mette in risalto maggiormente la separazione tra l'informazione, di cui fruirà l'utente e la logica (il codice) che è alla base del funzionamento del software. Con questo semplice paradigma è possibile sviluppare un prodotto concentrandosi sulle informazioni prodotte e non sui metodi con cui ottenerle. Sarà sempre possibile in seguito affinare le funzioni del software per renderlo più efficiente e completo.

## DEFINIRE GLI OBIETTIVI

È giunto il momento di definire i requisiti funzionali del nostro progetto di "notizie" che ci vedrà coinvolti. Cosa sappiamo al momento? Poco effettivamente, se non che visualizzeremo una pagina di informazioni prelevate da un database e che potremo definirle e memorizzarle noi stessi tramite una apposita interfaccia gestita da un form. Stendiamo una prima lista:

- realizzare un sito web che sia composto da più sezioni, che in futuro si possano ampliare
- le sezioni inizialmente previste sono: una pagina principale in cui visualizzare le notizie, una in cui realizzare un form per memorizzarle ed una pagina per essere contattati

- il sito, inizialmente strutturato in pagine statiche, deve potersi evolvere adottando contenuti dinamici
- la visualizzazione delle notizie deve essere realizzata mediante una sola pagina principale
- ogni notizia sarà composta da un titolo, il relativo testo e la data di inserimento.
- la pagina per l'inserimento sarà composta da appositi moduli per inserire il titolo e il testo dell'informazione. Per il campo data invece verrà automaticamente memorizzata quella di inserimento della notizia, senza alcun intervento da parte dell'utente

Definendo una serie di punti che si potranno (e dovranno) sviluppare successivamente con maggiore dovizia di particolari, si rende più chiara l'architettura del progetto con positive ripercussioni sui tempi e sul piacere di svilupparlo. Non c'è niente di più appagante che realizzare un buon codice senza il fastidio di doverci rimettere le mani a più riprese per apportare modifiche non preventivate.

### 3.5 RIEPILOGO

Queste regole o accorgimenti risultano probabilmente tediosi e possono essere tralasciati per gettarsi a capofitto nella programmazione. Se il progetto è di piccole dimensioni, come quello di cui ci occuperemo, si può anche correre questo rischio. È bene comunque abituarsi a condurre uno studio preventivo ogni volta che ci si appresta a realizzare un software di qualsiasi tipo e dimensione.





## 4 | I FANTASTICI TRE

Gli elementi alla base dei moderni software di sviluppo si basano su tre punti cardine che definiscono per l'appunto l'architectural pattern Model-View-Controller. Originariamente introdotto nel linguaggio Smalltalk, il pattern è stato esplicitamente o implicitamente sposato da numerose tecnologie moderne, come framework basati su PHP (Symfony, Laravel, Zend Framework, CakePHP, Yii framework, CodeIgniter), su Ruby (Ruby on Rails), su Python (Django, TurboGears, Pylons, Web2py, Zope), su Java (Swing, JSF e Struts), su Objective C o su .NET.

Il pattern MVC è in grado di separare la logica di presentazione dei dati dalla logica di business rendendo lo sviluppo modulare e decisamente più ordinato. In Rete è possibile reperire molta documentazione sull'argomento; tra la tanta, segnaliamo le slide universitarie del Prof. Maurizio Pizzonia su [http://www.dia.uniroma3.it/~pizzonia/swe/slides/12\\_MVC.pdf](http://www.dia.uniroma3.it/~pizzonia/swe/slides/12_MVC.pdf)

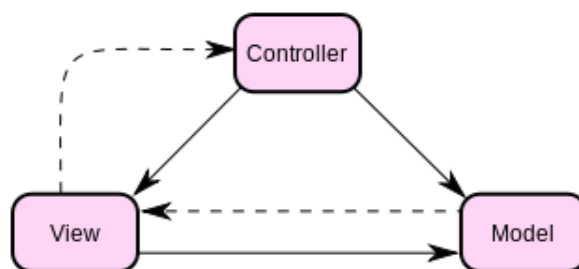


Fig. 10: Il pattern MVC

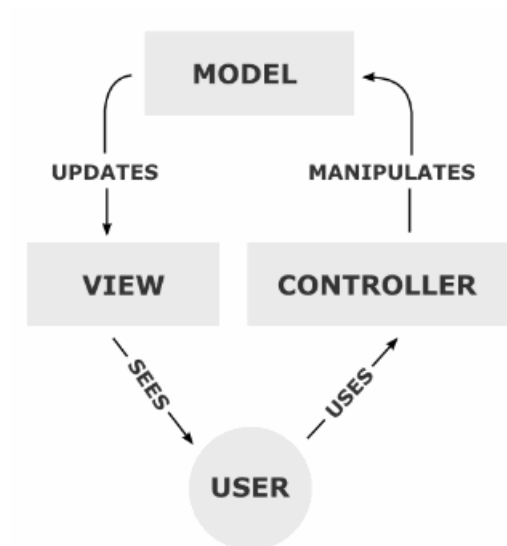


Fig. 11: I processi

## 4.1 IL CONTROLLER

Ora si entrerà nel vivo del nostro programma. Introduciamo il concetto del Controller che ci renderà più chiaro il pattern [MVC](#) e lo stesso CodeIgniter. Con poche righe di codice genereremo una pagina con un elenco di notizie prelevate da un database e saremo anche in grado, tramite un form, di crearne facilmente di nuove e di memorizzarle automaticamente in una base di dati persistente. Se l'installazione di CodeIgniter e la configurazione del server web saranno stati portati a termine correttamente si sarà pronti per creare il primo Controller. In caso contrario, ahimè, vi rimandiamo al paragrafo [2.5 a pagina 18](#) per la risoluzione dei problemi più comuni.

La nostra pagina delle notizie avrà un indirizzo presumibilmente come:

```
http://127.0.0.1/codeigniter/index.php/pages/news/
```

Che si può scomporre nel seguente schema:

```
http://127.0.0.1/[codeigniter/index.php]/[controller]/[metodo]/[argomenti]
```

**127.0.0.1** è l'indirizzo della nostra macchina locale o del sito online se questi si trova su un server esterno

**CODEIGNITER/INDEX.PHP** è la directory in cui è installato il framework che sarà presente in tutti gli [URI](#) della nostra guida

**CONTROLLER** è la classe principale che viene invocata per instradare la nostra richiesta. Nel nostro esempio si chiama **pages**. Questa classe racchiude uno o più metodi dalle precise funzionalità

**METODO** si chiama **news** ed è il metodo del nostro Controller **pages**. È questo il nome con cui sono chiamate le funzioni nei linguaggi orientati agli oggetti come lo è il [PHP](#). Ogni metodo è in pratica una funzione specifica del nostro programma. In questo esempio, il metodo **news** si occuperà della pubblicazione delle "notizie"

**ARGOMENTI** (facoltativi) si riferiscono alle variabili e ai relativi valori che vengono passati al metodo **news**. Attualmente non abbiamo previsto alcun argomento ma in seguito li introdurremo, non preoccupatevi

## IL NOSTRO PRIMO CONTROLLER

Con un editor di testo creiamo un file di nome **pages.php** al seguente percorso **application/controllers/** con il codice qui riportato:

```
<?php
class Pages extends CI_Controller {
    public function view()
    {

    }
}
```

Il codice [PHP](#) è molto semplice: viene definita una classe **Pages** (il nostro Controller) e al suo interno il metodo **view()**, al momento non ancora definito. Quest'ultimo, una volta completo, si occuperà di visualizzare un elenco di notizie.

I Controller, lo ribadiamo, accettano in ingresso una richiesta attraverso il protocollo [HTTP](#) e associano ad essa una funzionalità del nostro programma. Come viene definita la richiesta in un Controller? Semplicemente definendo un [URI](#), ovvero una stringa che identifica in modo univoco una risorsa del progetto. Riassumiamo i concetti esposti con un esempio pratico che tenga conto del nostro progetto:

```
http://127.0.0.1/codeigniter/index.php/pages/view
```

In questo indirizzo, subito dopo **index.php** compare il nome del nostro Controller **Pages** e il metodo **view()**. La nostra richiesta si può riassumere così: *“per piacere, con questo indirizzo ti chiedo di caricare il Controller Pages e di eseguire la funzione View”*.

Tra l'[URL](#) di base (127.0.0.1) e il nome del controller **/pages/** è presente la stringa **/index.php/**: per impostazione predefinita essa è obbligatoria, ma volendo si può configurare CodeIgniter per nascerla e “accorciare” l'indirizzo. Non preoccupiamoci troppo neppure del metodo **view()**: lo completeremo molto presto.

A questo punto siamo pronti per completare il nostro Controller. Ricordate che il metodo **view()** era stato lasciato vuoto? Così com'era non poteva svolgere alcun compito, quindi vediamo di porvi rimedio.

```
<?php
class Pages extends CI_Controller {
    public function view()
    {
        echo "Ciao! Ti trovi nella sezione principale del sito";
    }
}
```

Si tratta di poche righe, che risulteranno familiari a chi possiede anche una minima conoscenza del [PHP](#).

- `[class Pages extends CI_Controller]` definisce la classe **Pages** che è una estensione della classe-Controller principale di CodeIgniter, `CI_Controller`. Quando definiremo un Controller abituiamoci a questa sintassi

- function `view()` è il nostro metodo che si occuperà, in questo caso specifico, di visualizzare un testo

Il metodo **`view()`** scritto in codice [PHP](#) utilizza il comando **`echo`** per visualizzare la stringa contenuta tra virgolette. Ovviamente nessuno vi vieta di personalizzare il messaggio se lo desiderate. Per vedere il risultato del primo Controller/metodo, carichiamo la pagina come precedentemente indicato:

```
http://127.0.0.1/codeigniter/index.php/pages/view
```

## FACCIAMO PRATICA

Proviamo a comprendere meglio la funzione del Controller realizzando un secondo esempio. Se nel nostro sito volessimo inserire anche una pagina dedicata ai “contatti”, basterà definire un nuovo Controller. Si crei un nuovo file **`contatti.php`** in **`application/controllers/`** con il codice:

```
<?php
class Contatti extends CI_Controller {
    public function index()
    {
        echo "Per contattarci inviaci una email o telefonaci";
    }
}
```

Le differenze rispetto al precedente Controller sono minime, eppure significative. Qui di seguito un elenco chiarificatore:

**IL NOME** il Controller ora si chiama **`Contatti`**: questo è esplicitato sia nel codice (il nome della classe) che nel nome del file (**`contatti.php`**).

**METODO** il metodo **`view()`** viene sostituito da **`index()`**, che è definito come il “metodo base” di ogni Controller. Se non viene specificato alcun metodo, nella richiesta [HTTP](#) il Controller esegue automaticamente quello predefinito, appunto **`index()`**.

Osserviamo il frutto del nostro lavoro digitando:

```
http://127.0.0.1/codeigniter/index.php/contatti/
```

Ricordate quanto detto prima, analizzando la sintassi degli [URI](#) a pagina [33](#)? Subito dopo la stringa **`index.php`** compare **`contatti`**: questo è il nuovo Controller. Inoltre, non specificando alcun metodo subito dopo **`/contatti/`**, verrà eseguito **`index()`**, quello predefinito di ogni Controller. Si vuole rimarcare che un Controller non

“genera” alcunché. Il suo compito non è produrre, o visualizzare, ma esaminare una richiesta [HTTP](#), ovvero un indirizzo ed eseguire la funzione definita al suo interno. In modo analogo è possibile sbizzarrirsi creando svariati Controller, ognuno dei quali corrisponde ad un nuovo file all’interno del percorso **application/controllers/**:

- <http://127.0.0.1/codeigniter/index.php/about/> (instrada verso la pagina dei contatti)
- <http://127.0.0.1/codeigniter/index.php/login/> (instrada verso la pagina per autenticarsi)
- <http://127.0.0.1/codeigniter/index.php/top/> (instrada verso una classifica generica)

Nota: Vi è una aspetto sui nomi dei file e delle classi a cui è bene prestare attenzione. Il file del Controller (**pages.php**) assume il nome della classe definita al suo interno (negli esempi **Pages** e **Contatti**). Il nome del file deve essere scritto in minuscolo, mentre la classe definita all’interno del codice deve presentare il primo carattere (l’iniziale) in maiuscolo.

## UN METODO AVANZATO

Riprendiamo il metodo **view()** per introdurre il concetto di “passaggio di variabili”. In precedenza si era accennato al fatto che un metodo può contenere degli argomenti opzionali. Esaminiamo il nostro Controller **Pages** opportunamente modificato.

```
<?php
class Pages extends CI_Controller {
    public function View($page = 'home')
    {
        echo "Ciao! Ti trovi nella sezione $page";
    }
}
```

Il codice è quasi immutato tranne che per l’introduzione di due elementi.

- `$page = 'home'` all’interno del metodo **view()** si trovano, specificati tra parentesi, gli (opzionali) argomenti. Nel nostro caso viene definita la variabile `$page` che assume il valore “home”
- il messaggio che viene stampato su schermo con **echo**, ora contiene un riferimento alla variabile appena introdotta: la nostra scritta sarà personalizzata in base al valore che l’argomento `$page` assumerà

Se ora si digita: **<http://127.0.0.1/index.php/pages/view>** comparirà il messaggio “Ciao! Ti trovi nella sezione home”.

La variabile `$page` del metodo **view()** in mancanza di un argomento specifico, assume il valore predefinito “home” e lo riporta nel messaggio. Proviamo ora ad introdurre un argomento. Digitiamo l’indirizzo seguente e leggiamo il messaggio.

`http://127.0.0.1/codeigniter/index.php/pages/view/secreta`

Ebbene sì, attraverso questo [URI](#) definiamo in sequenza il Controller **Pages**, il metodo **view()** e il relativo argomento **secreta** che personalizza il messaggio su schermo. Se l’argomento non è del tutto chiaro, non preoccupatevi, verrà ripreso nuovamente in seguito con altri esempi (si veda a pagina [43](#)).

## IMPORTANZA DEL CONTROLLER

Ora dovrebbe essere maggiormente chiaro il compito svolto dal Controller all’interno del pattern [MVC](#). Ma perché è così importante? Chi ha qualche nozione di programmazione potrebbe supporre che questo modo di ragionare sia troppo farraginoso e che un controller possa essere banalmente sostituito da una libreria di funzioni che svolga compiti analoghi. Ma questo non è propriamente corretto, perché lo scopo ultimo di un Controller è instradare, non generare.

Si ipotizzi, in un futuro prossimo, di voler modificare la struttura del proprio sito, magari per realizzare una versione in lingua inglese da affiancare a quella già esistente in italiano. In questo caso, per accedere anche ad una ipotetica pagina inglese dei “contatti” utilizzando il pattern [MVC](#), si implementeranno due controller:

- <http://mioSito/index.php/contatti> (per la versione in italiano)
- <http://mioSito/index.php/contact> (per la versione in inglese)

È stata variata solo una parola (il Controller), ma questa modifica è solo in parvenza banale. Infatti se il progetto non avesse adottato il pattern [MVC](#) si sarebbe potuti incorrere in problemi molto fastidiosi, come per esempio la ridefinizione di tutti i link nei menu di navigazione, e probabilmente delle funzioni richiamate dagli script. Apportare queste correzioni si rivela sempre impegnativo, noioso, senza tener conto degli errori che si introducono ogni volta che si modifica un vecchio codice.

La definizione di un Controller risolve agevolmente tutti questi problemi. Nel nostro caso specifico, basterà creare un nuovo Controller **contact.php** con all’interno un codice identico a quello di **contatti.php**, avendo ovviamente l’accortezza di rinominare la classe al suo interno in **Contact** (mi raccomando solo l’iniziale maiuscola!). In questo modo un utente potrà digitare ambedue gli indirizzi sopra esposti ed essere instradato verso la medesima pagina. Anche se, come è lecito immaginare, si dovrebbero personalizzare i due Controller per caricare due pagine con informazioni differenti: una in italiano e l’altra in inglese. Ma questo, al momento, non è una priorità.

Il Controller è in definitiva questo: analizza una richiesta esplicitata nell’[URI](#) presente nell’indirizzo e instrada verso il metodo appropriato. Altre sue importanti funzionalità hanno ripercussioni positive sulla sicurezza del progetto web tanto per

citare un esempio. Prima che il Controller sia caricato e i suoi metodi eseguiti, la “richiesta http” e tutti i dati inviati dall’utente vengono filtrati per la sicurezza. Il Controller carica il Modello, le Librerie, gli Helper e ogni altra risorsa necessaria ad eseguire la richiesta specifica ed inoltre fa da intermediario tra il Modello, la Vista e ogni altra risorsa necessaria alle richieste [HTTP](#) e alla generazione di pagine web. CodeIgniter è abbastanza permissivo sull’approccio al pattern [MVC](#), in quanto si può anche escludere la parte riguardante il Modello: se non si utilizza un database, si può semplicemente ignorare questo aspetto e costruire la propria applicazione utilizzando il Controller e la Vista. CodeIgniter permette anche di incorporare i propri script o addirittura sviluppare le librerie di base, per lavorare in maniera più produttiva.

## RIEPILOGO

Probabilmente se si è arrivati a questo punto alcuni concetti saranno maggiormente chiari, altri continueranno a non esserlo, ma non preoccupatevi troppo: siete solo all’inizio. CodeIgniter è un framework che si differenzia dalla concorrenza per la sua facilità d’uso, leggerezza e prestazioni. Il suo scopo è sviluppare progetti software (generalmente web) di media complessità, utilizzando l’approccio [MVC](#) per separare la logica (il codice) dalla presentazione (l’aspetto grafico del nostro software). Per ottenere questo scopo vengono utilizzati il Controller, il Modello e la Vista. Il primo accetta una richiesta, la esamina ed esegue una operazione tra quelle previste (per esempio potrebbe restituire una pagina, fornire un elenco di dati prelevati da un database, ecc). Il Controller si comporta come un vigile in mezzo al traffico: esamina la situazione e in base alle direttive (fornite dallo sviluppatore) dirige il flusso verso una determinata strada. Il Modello (raramente opzionale) contiene le query per interfacciare il Controller con la nostra base di dati. La Vista infine si occupa dell’aspetto visuale delle pagine. In questo capitolo si è fatta dimestichezza con i Controller, compiendo un primo passo nella realizzazione di un progetto per la visualizzazione delle notizie collegate ad una base di dati.



## 4.2 LA VIEW

La View, o comunemente chiamata Vista, costituisce l'informazione finale che viene mostrata all'utente. Essa solitamente si identifica con una pagina web, ma il concetto è decisamente più ampio: in CodeIgniter una Vista può anche essere solo un frammento di una pagina come un header o un footer. Anche i formati possono essere i più disparati, [HTML](#), [PHP](#), Really Simple Syndication ([RSS](#)) o ogni altro formato di pagina.

### INTRODUZIONE

La Vista costituisce il tassello “visuale” del pattern [MVC](#). La sua funzione è quella di rappresentare il nostro progetto in un formato testuale come [HTML](#) o un altro linguaggio di markup. Ovviamente tra i vantaggi nell'utilizzo della Vista, valgono le considerazioni esposte in precedenza come la “separazione” della logica dalla presentazione. Questo permette di massimizzare l'efficienza nello sviluppo di un progetto web, minimizzando gli inconvenienti dovuti ad una sovrapposizione delle competenze tra programmatori e designer. L'utilizzo del pattern [MVC](#), e nel caso specifico delle View fa sì che i designer possano concentrarsi su specifici file su cui sviluppare le proprie competenze.

La creazione delle View è estremamente semplice, modulare e flessibile. É importate però tenere a mente alcuni concetti:

- una View non può essere chiamata direttamente tramite un [URL](#): la loro gestione è demandata ai Controller
- la View si trova nel percorso `/application/views/`
- è possibile prevedere un ordinamento gerarchico articolato con View che richiamano altre View
- molti framework consentono l'utilizzo di ulteriori linguaggi di scripting per ridurre l'impatto dell'eventuale codice di programmazione

### SEMPLIFICARE TRAMITE SCRIPTING

I template, anche quando sono prodotti tramite una View, contengono spesso all'interno codice in [PHP](#), o altri linguaggi, che rendono meno netta la separazione tra presentazione e logica. Proprio per questo è possibile adottare degli ulteriori linguaggi che interpretano, per esempio, il [PHP](#) e producono un codice dello stesso valore semantico ma più chiaro e conciso: in questo modo si realizza il sogno del designer che si ritroverà l'eventuale codice [PHP](#), Javascript, ecc, circoscritto in blocchi a se stanti, incapaci di generare confusione.

Ci sono però anche risvolti negativi. É necessario apprendere un nuovo linguaggio di scripting, che per quanto chiaro e semplice, va comunque padroneggiato. Ogni framework ha un diverso approccio allo scripting delle View e questo perché manca

uno standard comune. Inoltre le performance del progetto risentono della natura interpretata di questo codice. Insomma, l'adozione o meno di un codice di scripting ha numerosi vantaggi, ma anche degli aspetti da ponderare. Per fortuna il suo utilizzo è opzionale, per questo lo introdurremo in uno dei capitoli successivi (si veda la sezione [7.9 a pagina 140](#)).

La prima View verrà realizzata creando nel percorso **/application/views/** il file **lista.php**. Il codice utilizzato è un elementare [HTML](#) e dovrebbe risultare facilmente comprensibile:

```
<html>
<head>
<title>Primo progetto con CodeIgniter</title>
</head>
<body>
<h1>Lista delle notizie principali</h1>
realizzata da TizioCaio
</body>
</html>
```

Questa Vista, quando richiamata da un Controller, produrrà una pagina dal titolo “Primo progetto con CodeIgniter” e un testo principale “Lista delle notizie principali”.

## SIAMO PRONTI A VISUALIZZARE

La View, come precedentemente detto, non può essere processata autonomamente, ma deve essere richiamata da uno specifico Controller (se volete sincerarvene voi stessi, provate a caricare dalla barra degli indirizzi la Vista **lista.php**). Si apra nuovamente il file **pages.php** in **/application/controller** creato precedentemente e si aggiunga il codice per richiamare questa View:

```
$this->load->view('lista'); // senza estensione .php
```

Riportiamo il codice di tutto il Controller **pages.php** precedentemente sviluppato con la necessaria modifica.

```
<?php
class Pages extends CI_Controller {
    public function index()
    {
        $this->load->view('lista');
    }

    public function view($page = 'home')
    {
        echo "Ciao! Ti trovi nella sezione $page";
    }
}
```

L'istruzione che abbiamo introdotto carica (**load**) la Vista (**view**) denominata "lista". A questo file comprende anche l'estensione ".php", ma nel codice questa si omette.

## LA POTENZA DELLA VIEW

La nostra classe **Pages** ora è costituita da due metodi. Il primo (index) è quello che viene richiamato automaticamente, se non si specifica alcuna funzione: carica la Vista **lista** che si trova in un file separato. Il secondo metodo, invece, è l'esempio riportato precedentemente a pagina 36 che visualizza sempre un testo, ma senza alcun rimando ad una Vista, e quindi ad un file esterno. Anche se questa pratica è consentita e spesso utilizzata, si consiglia di abituarsi all'utilizzo delle View che permettono di sfruttare appieno le potenzialità del pattern **MVC**.

Le pagine di un sito sono suddivise generalmente in poche componenti principali, che sovente si ripetono. Sarebbe quindi molto comodo, poter "riciclare" parte del codice, velocizzando lo sviluppo. Le parti di un sito sono generalmente:

- header. La testata di un sito, che comprende il logo aziendale e il richiamo a librerie javascript, file **CSS**, ecc. Generalmente rimane immutata in tutto il sito, ed è quindi un elemento che sarebbe consigliato riutilizzare
- menu. Il menu di navigazione. Anche questo template rimane inalterato in tutto il sito
- main. È la parte principale che contiene le informazioni da proporre al visitatore. Questa pagina risulterà, per ovvie ragioni, sempre differente a seconda della sezione visitata
- footer. La parte finale di ogni pagina, ovvero quella sezione che contiene il copyright, la data, ed eventuali informazioni sull'indirizzo dell'azienda o i contatti. Anche questo template non è soggetto a variazioni ed è altamente riciclabile

## SUDDIVIDIAMO IL TEMPLATE

Ora che abbiamo dimestichezza con i Controller e le View, suddividiamo la nostra pagina precedentemente preparata (vedi a pagina 40) in elementi distinti e riutilizzabili. Realizziamo quindi le View nel percorso `/application/view/` con il seguente codice:

**Header.** Il codice `HTML` della nostra testata.

```
<html>
<head>
<title>Primo progetto con CodeIgniter</title>
</head>
<body>
```

**Menu.** Un menu contestuale per accedere agevolmente alle sezioni del progetto. Al momento ne abbiamo previsto tre: la prima visualizzerà tutte le notizie, la seconda permetterà di inserire nuove informazioni e l'ultima pagina visualizzerà i contatti aziendali.

```
<nav>
  <a href="#">Notizie</a>
  <a href="#">Inserimento</a>
  <a href="#">Contattaci</a>
</nav>
```

**Main.** Il piccolo testo della nostra pagina. Non preoccupatevi, con il tempo si amplierà.

```
<h1>Lista delle notizie principali</h1>
```

**Footer.** L'ultima sezione della nostra pagina.

```
realizzata da TizioCaio
</body>
</html>
```

Nota. Il codice riferito alle sezioni, come precedentemente detto, dovrà essere inserito in file differenti dotati di estensione `".php"`, mentre nel codice del Controller ogni riferimento a questa estensione verrà omissso. Ma dedichiamoci subito ad esso, aprendo nuovamente il file **pages.php**.

```

<?php
class Pages extends CI_Controller {
    public function index()
    {
        $this->load->view('header'); // carica l'header
        $this->load->view('menu'); // carica il menu
        $this->load->view('main'); // carica la sezione centrale
        $this->load->view('footer'); // carica la parte finale
    }

    public function view($page = 'home')
    {
        echo "Ciao! Ti trovi nella sezione $page";
    }
}

```

All'interno del metodo **index()** abbiamo inserito quattro istruzioni per caricare le relative Viste. In questo modo è possibile sviluppare agevolmente ogni pagina del nostro sito senza dover riscrivere per esempio l'header e il footer che rimarranno identici. Questo metodo non solo apporta vantaggi per quanto riguarda la rapidità di sviluppo, ma facilita anche la manutenibilità del codice. Se un domani si volesse modificare per esempio il footer, inserendo anche l'indirizzo dell'azienda, basterà aprire la View relativa al **footer.php** e apportare la modifica per vederla propagata sull'intero sito. Molto comodo non trovate?

## INTRODUZIONE ALLE VARIABILI

Si vuole ora riprendere un discorso introdotto con i metodi del Controller. Le variabili, come dice il nome stesso, sono degli elementi "non immutabili" che possono assumere valori differenti a seconda del contesto grazie a delle istruzioni definite "condizionali". Nel linguaggio umano, quanto detto si potrebbe esplicitare con un esempio del tipo "*se piove* (istruzione condizionale) *io* (variabile) *prenderò un ombrello* (valore assumibile), *altrimenti* (istruzione condizionale) *no* (altro valore assumibile)".

Non staremo qui a teorizzare sull'importanza rivestita dalle variabili nella programmazione: la loro utilità è disarmante in ogni ambito. Qui ci occuperemo del loro ruolo nella realizzazione di un sito dinamico.

### Definizione 4: Dinamico

<b>Dinamico</b>	In contrapposizione con statico, si indicano tutti quei siti in cui le informazioni proposte al visitatore tengono conto della sua identità, delle sue scelte o di particolari contesti
-----------------	---

Abbiamo già incontrato alcune variabili nel codice del nostro Controller: è stato

assegnato loro un valore ed è stato anche effettuato un “passaggio di variabili”. Per i meno attenti, riproponiamo il secondo metodo del Controller **Pages** (si veda la sezione [4.1 a pagina 36](#)).

```
public function view($page = 'home') {  
    echo "Ciao! Ti trovi nella sezione $page";  
}
```

Ricordate? Il metodo **view()** è composto dalla variabile `$page` racchiusa tra parentesi, la cui definizione più corretta è “argomento” del metodo. `$page` assume il valore preimpostato “home”, ma nulla ci vieta di modificarla a piacimento.

Digitiamo:

```
http://127.0.0.1/codeigniter/index.php/pages/view/segretissima
```

Bene, ora il testo è cambiato tenendo conto dell’argomento **segretissima** fornito nell’**URI**. Quindi il metodo **view()** visualizza un semplice testo su schermo e se si scrive semplicemente:

```
http://127.0.0.1/codeigniter/index.php/pages/view/
```

non effettuiamo alcun passaggio di valori, e la variabile **pages** assume il valore di default “home”. In caso contrario, verrà valorizzata con l’argomento fornito, che modificherà il testo che comparirà sullo schermo. Si digitino altri **URL** per apprezzarne la flessibilità:

- `http://127.0.0.1/codeigniter/index.php/pages/view/listati`
- `http://127.0.0.1/codeigniter/index.php/pages/view/contattaci`
- `http://127.0.0.1/codeigniter/index.php/pages/view/immagini`

Questo esempio scalfisce in minima parte le potenzialità del passaggio delle variabili, ma ritorniamo al nostro progetto e vediamo di mettere a frutto ciò che abbiamo appena appreso.

### 4.3 RESTITUIRE UNA VIEW COME UN DATO

É possibile fare in modo che il metodo restituisca i dati sotto forma di una stringa piuttosto che inviarli direttamente al browser: questo può tornare utile se si desidera elaborare i dati in qualche altro modo.

Per modificare il comportamento della View, introduciamo un terzo parametro opzionale che accetta il valore booleano “true”. Se lo si imposta, ci si ricordi di assegnare il metodo View ad una variabile (nel nostro caso `$string`) per memorizzare i

dati restituiti come una stringa. Il comportamento predefinito assunto da CodeIgniter è “false”. Ecco un esempio di quanto appena detto:

```
$string = $this->load->view('miofile', '', true);
```

## 4.4 PERSONALIZZIAMO L'HEADER

Utilizziamo quanto appreso per personalizzare le Viste create nel nostro progetto. Lo scopo sarà quello di modificare il testo visualizzato per renderlo dinamico. Si apra la View **header.php** e si modifichi il codice per far sì che il titolo della pagina non sia più statico.

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

Si è utilizzata la funzione **echo** del **PHP** che abbiamo già incontrato: essa stamperà la variabile `$title`. Bene, abbiamo la nostra variabile pronta per essere rappresentata su schermo: ora assegniamogli un valore.

Modifichiamo il Controller **Pages** nell'omonimo file **pages.php**:

```
<?php
class Pages extends CI_Controller {

    public function index()
    { $data['title'] = "sito creato con CodeIgniter";
      $this->load->view('header', $data);
      $this->load->view('main');
      $this->load->view('footer');
    }

    public function view($page = 'home')
    {
        echo "Ciao! Ti trovi nella sezione $page";
    }
}
```

É stata introdotto l'array `$data` che contiene al suo interno l'indice “title” valorizzato con una stringa di testo. La riga successiva caricherà la Vista attraverso il metodo **load->view**. Si faccia attenzione all'argomento inserito tra parentesi: non solo viene specificato di caricare la Vista **header**, ma viene passato anche l'array `$data` con tutti i suoi indici.

## MODIFICHIAMO LE ALTRE VISTE

Seguendo quanto appreso sinora, modifichiamo anche la Vista **main.php**:

```
<body>
<h1><?php echo "$text"; ?></h1>
```

e il **footer.php**:

```
<?php echo "realizzato da $azienda"; ?>
<?php echo "copyright " .date("Y"); ?>
</body>
</html>
```

Per vedere gli effetti, modifichiamo nuovamente il Controller ampliando il nostro array di dati.

```
<?php
class Pages extends CI_Controller {

    public function index()
    { $data['title'] = "sito creato con CodeIgniter";
      $data['text'] = "Lista delle notizie principali";
      $data['azienda'] = "realizzata da TizioCaio ";

      $this->load->view('header', $data);
      $this->load->view('main', $data);
      $this->load->view('footer', $data);
    }

    public function view($page = 'home')
    {
        echo "Ciao! Ti trovi nella sezione $page";
    }
}
```

Nota: è possibile definire il nostro array anche con una sintassi differente che comunque non apporta alcuna modifica al risultato prodotto. Qui di seguito lo riportiamo per completezza, ridefinendo il solo metodo **index()**:



```
public function index()
{
    $data = array(
        'title' = "sito creato con CodeIgniter",
        'text' = "Lista delle notizie principali",
        'azienda' = "realizzata da TizioCaio"
    );

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

Se tutto funziona correttamente verrà visualizzata una pagina identica a quella statica realizzata inizialmente (si veda la sezione [4.2 a pagina 40](#)). A questo punto ci si potrebbe chiedere perché complicarsi la vita per ottenere un risultato simile a quello originale. I vantaggi di un framework, lo ripetiamo, non si valutano nella realizzazione di queste poche pagine, ma nella creazione e gestione di un medio-grosso progetto software (si veda la sezione [1.1 a pagina 1](#)).

Esaminiamo quanto prodotto sinora per rendercene conto.

**CODICE MEGLIO ORGANIZZATO** La nostra iniziale, unica pagina in [HTML](#) è stata scomposta in più parti distinte. Questo modo di operare permette la realizzazione delle pagine del nostro progetto, intervenendo solo sulla Vista che contiene le informazioni da visualizzare, ovvero **main.php**. Le altre Viste generalmente rimangono uguali in tutto il sito, e questo si traduce in un codice snello, facile da gestire e riutilizzare.

**TESTO DINAMICO** Le informazioni rappresentate su schermo non sono più statiche, ma cambiano a seconda del valore assegnato alle variabili all'interno del Controller. Certo, al momento si obietterà che la sostanza non è cambiata, ma nei prossimi capitoli si comprenderanno maggiormente i vantaggi della soluzione adottata.

**INFORMAZIONI ADATTIVE** Quello che appare su schermo si adatta automaticamente al contesto. Un esempio è dato dal codice del **footer.php**: nonostante non sia stato inserito esplicitamente l'anno corrente, questo viene correttamente visualizzato. Questo piccolo "prodigio", è reso grazie ad una apposita funzione del [PHP](#) che esamina la data del sistema e la riporta su schermo. Il prossimo anno non dovrete toccare alcuna pagina, perché la data del copyright si aggiornerà automaticamente.

In questo capitolo abbiamo finalmente aperto il nostro editor per scrivere un po' di codice. Abbiamo migliorato il nostro primo Controller per visualizzare, prima un testo interno e successivamente uno richiamato tramite le View. Questo ci ha fatto apprezzare alcuni degli aspetti del pattern [MVC](#): la velocità di sviluppo, il riutilizzo del codice, e la sua manutenibilità elevata.

La View è quella componente del framework demandata alla visualizzazione delle informazioni. Essa non può essere richiamata direttamente tramite il browser,

ma dipende dal Controller. L'utilizzo di questo aspetto cardine del pattern **MVC** fa sì che si realizzi un codice dove la presentazione è separata dalla logica.

È stato introdotto anche il passaggio di variabili, ma l'importanza che queste rivestono in un sito dinamico è stata appena messa in evidenza. Chi ha nozioni di programmazione si troverà a suo agio con questi concetti; in fondo, il Controller è una classe che contiene a sua volta uno, o più metodi, mentre il loro argomento rappresenta l'insieme di variabili che andremo ad utilizzare. Il nostro progetto è solo agli inizi, ma già mostra un template suddiviso in header, menu, main e footer, che possono essere modificati indipendentemente. Impareremo nei prossimi capitoli a migliorare quanto abbiamo sviluppato e a introdurre nuovi concetti.

## 4.5 EVOLUZIONE DEL PROGETTO

In questo capitolo miglioreremo alcuni aspetti del nostro software basandoci su quanto già appreso. Svilupperemo nuovi metodi e renderemo i Controller più articolati e flessibili.

### METODI MULTIPLI IN UN CONTROLLER

Il nostro Controller **Pages** è stato definito, inizialmente con un solo metodo **view()** a cui si è aggiunto quello di default **index()**. Questo ci fa comprendere che è possibile, anzi generalmente è la prassi, definire più metodi. Esaminiamo lo schema generale di un Controller:

```
// Schema generale di un Controller
class Sito extends CI_Controller {
    // definizione della funzione predefinita
    function index() {
        // output della funzione di default
        echo 'Messaggio stampato dalla funzione predefinita';
    }
    // definizione di una seconda funzione di classe
    function secondafunzione() {
        // output della seconda funzione
        echo 'Secondo messaggio';
    }
    // definizione di una terza funzione di classe
    function terzafunzione() {
        // output della terza funzione
        echo 'Terzo messaggio';
    }
}
?>
```

In questo Controller **Sito** vengono definiti tre metodi, ma nessuno ci vieta di implementarne un numero maggiore. Il primo metodo è **index()**: questo è solitamente il nome che viene dato al primo metodo di default. il suo scopo è quello di stampare la stringa *Messaggio stampato dalla funzione predefinita*. Di seguito, gli altri due metodi si occupano di stampare le relative stringhe.

Ora creiamo qualcosa che abbia un fine se non nobile, almeno utile per il nostro progetto: realizzare un metodo per visualizzare i contatti del nostro sito. A questo punto si hanno tutte le conoscenze necessarie per realizzare una pagina statica senza alcun tipo di aiuto: provate a scriverla da soli come pratico esercizio cercando di non sbirciare la prossima sezione.

## LA PAGINA DEI CONTATTI

Per visualizzare i contatti si è utilizzata una istruzione condizionale *if/else* all'interno del metodo **view()**. Il funzionamento è abbastanza intuitivo: se l'argomento passato al metodo è diverso dalla stringa *contatti*, verrà caricata una pagina generica. In caso contrario (ramo *else*), l'argomento passato è proprio la stringa desiderata, nel qual caso viene caricata la pagina con i riferimenti per contattare gli autori.

```
<?php

class Pages extends CI_Controller {

    public function index()
    { $data['title'] = "sito creato con CodeIgniter";
      $data['text'] = "Lista delle notizie principali";
      $data['azienda'] = "realizzata da TizioCaio ";

      $this->load->view('header', $data);
      $this->load->view('main', $data);
      $this->load->view('footer', $data);
    }

    public function view($page)
    { if ($page != 'contatti')
      { echo "Ciao! Ti trovi nella sezione $page";
      }

      else // (se e' uguale a contattaci)
      {
        $data['title'] = "sito creato con CodeIgniter";
        $data['text'] = "Pagina dei contatti";
        $data['azienda'] = "realizzata da TizioCaio ";

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

## AGGIUNGERE LA LOGICA AL CONTROLLER

É possibile aggiungere facilmente una istruzione condizionale anche per verificare se una Vista, che intendiamo caricare, esiste o meno. Se la Vista in questione è presente e funzionante, verrà visualizzata, altrimenti comparirà un messaggio di “avvertimento”.

```
public function view($page)
{
    if ( ! file_exists('application/views/'.$page.'.php'))
    {
        // Ehm, Questa pagina non esiste
        show_404();
    }

    $data['title'] = ucfirst($page); // La prima lettera è resa maiuscola

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

Il codice si riferisce al Controller **Pages** e al metodo **view()** che è stato modificato per effettuare un controllo sul caricamento delle Viste.

1. `if ( ! file_exists())` È una istruzione condizionale del **PHP** che restituisce un valore booleano. Il punto esclamativo trasforma la frase in “*se non esiste* l’argomento interno
2. `show_404()` Se la View non esiste, questo metodo viene eseguito visualizzando un messaggio di errore (404 Page not found)
3. `ucfirst()` è un metodo che trasforma la prima lettera del valore dato a `$page` (ovvero il titolo) in maiuscolo e lo assegna a `$data['title']`
4. Gli ultimi tre metodi si occupano di caricare le Viste predefinite

In questo modo si sono raggiunti alcuni importanti obiettivi. Si è appreso come verificare l’esistenza o meno di una Vista (e di qualsiasi file), e come visualizzare un messaggio di errore automatizzato. Inoltre è lecito aspettarsi che il nome della Vista, sia così esplicativo da poter essere utilizzato anche come titolo della pagina: per questo lo abbiamo assegnato all’indice **title** dell’array **data**, dopo aver reso maiuscola l’iniziale. Le precedenti Viste create si chiamano **contatti.php**, **main.php** e quindi si prestano bene ad essere utilizzate come titolo.

Con questi piccoli accorgimenti la realizzazione delle prossime pagine sarà ancora più rapida.

## 4.6 IL MODEL

Questo capitolo è dedicato al Modello, l'elemento del pattern MVC demandato ad interfacciare quanto sviluppato con una base di dati. Ora, non solo si farà pratica con tale prezioso strumento, ma si esamineranno anche le funzioni messe a disposizione per l'aggiornamento, inserimento, cancellazione e selezione dei dati tramite query.

### INTRODUZIONE ALLA BASE DI DATI

[...]

### SERVER LOCALE

Daremo per scontata una certa dimestichezza con i database di tipo MySQL: la loro installazione e configurazione è ormai alla portata dei neofiti grazie a progetti come LAMP per Linux e per i maggiori sistemi operativi. Abbiamo già esaminato un tutorial per la loro installazione e la configurazione (si veda la sezione 2.1 a pagina 11): per maggiori informazioni si rimanda anche alla vasta documentazione reperibile in rete. Si consiglia anche l'utilizzo del tool "phpMyAdmin" <http://www.phpmyadmin.net/> con cui è resa più agevole la gestione della propria base di dati grazie ad una intuitiva interfaccia grafica (vedi immagini in questa pagina e a pagina 65).

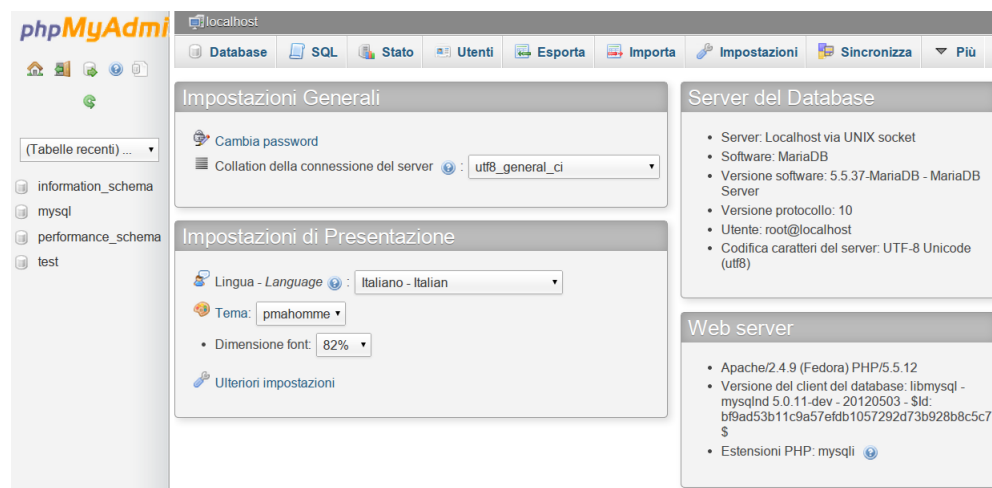


Fig. 12: phpMyAdmin

## CONFIGURAZIONE DELLA BASE DI DATI

Si crei un database che al momento chiameremo semplicemente “test”: tramite il menu in alto di phpMyadmin si selezioni la voce *Database* e si inserisca il nome desiderato, nel nostro caso “test” (vedi immagine a pagina 65). al suo interno definiamo una tabella “news” e quindi i suoi attributi (si veda a pagina 65). Come si ricorderà, nel capitolo 3.4 a pagina 28 si è già introdotto lo schema generico della nostra tabella: una serie di record con un *titolo*, un *testo* e la relativa *data di inserimento*. Realizziamo quanto appena esposto attraverso una serie di istruzioni MySQL, che possono essere eseguite attraverso l’interprete SQL installato o tramite l’interfaccia di phpMyAdmin.

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

L’attributo **slug** è un indice con cui “marcheremo” ogni notizia pubblicata per poi richiamarla facilmente in seguito: una sorta di segnalibro, più o meno. Ora si inseriscano due notizie per popolare la tabella. Qui viene riportato il codice che può essere processato velocemente selezionando la voce SQL dal menu principale (vedi immagine a pagina 66).

```
INSERT INTO `test`.`news` (`id`, `titolo`, `slug`, `testo`)  
VALUES (NULL, 'Oggi piove', 'n1', 'o almeno questo dicono le previsioni'),  
      (NULL, 'Domani splende il sole', 'n2', 'quindi andremo al mare');
```

## ANATOMIA DEL MODELLO

CodeIgniter, come visto per il Controller e le View, prevede una cartella apposita dove memorizzare i file Model. Come è oramai intuibile, questa si trova al percorso **/application/models/**.

Il framework richiamerà da questa directory i Model necessari alla nostra applicazione; è comunque possibile organizzare la directory in più sottocartelle in modo da realizzare una struttura del progetto strutturata e ordinata.

Creiamo quindi un nuovo file denominato `news_model.php` nel percorso relativo ai Modelli con il seguente codice:

```

<?php
class News_model extends CI_Model {

    public function __construct()
    {
        $this->load->database();
    }

    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();
    }
}

```

Il Model verrà richiamato tramite il Controller attraverso l'istruzione ormai facilmente intuibile:

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

Dove `nome_model` è il nome scelto per il proprio Model. Se questo si trova in una sottodirectory di `sysapplication/models/` basterà specificare il percorso prima del nome:

```
$this->load->model('percorso/nome_model');
```

La struttura di un Model è sintatticamente molto semplice e rispecchia le convenzioni già viste nel Controller e nella View. Viene creato un nuovo Modello di nome `News_model` che è una estensione del `CI_Model`, il modello di base di Codeigniter.

La nuova istruzione introdotta è `$this->load->database()` che si occupa di caricare il database i cui parametri sono memorizzati nel file **database.php** nel percorso **/application/config/** (si veda la sezione [4.6 a pagina 63](#)).

La seconda parte del codice si occupa di recuperare le notizie dal database “test”: d'altra parte è questo il compito del Model. All'interno del metodo `get_news()` sono presenti due differenti query: una per ottenere tutte le notizie e la seconda per recuperarne una ben precisa, marcata dall'identificatore **slug**.

Apriamo una piccola parentesi sulla sicurezza. Nella tradizionale programmazione ogni variabile usata come argomento per interfacciarsi con un database deve essere prima esaminata e “messa in sicurezza”, come nel caso di **slug**. Utilizzando



CodeIgniter non ci si dovrà preoccupare di questi dettagli perché sarà lui stesso a curare questo aspetto per noi.

Nota: una normale prassi nella programmazione ad oggetti e quella di chiamare i metodi che recuperano i dati con **get()**, mentre quelli che li modificano con **set()**. Il nostro metodo infatti si chiama `get_news()` e rende subito chiara la sua funzione. Ci si abitui a questa convenzione perché è universalmente adottata.

## ALIAS E AUTO-LOADING

È possibile specificare un nome che funga da alias per l'accesso alle funzioni del Model. Per esempio, le righe seguenti fanno sì che venga definito un alias di nome **pippo** che sostituisce il vero nome del Model quando se ne richiamano i metodi. Qui si fornisce un esempio della definizione dell'alias **pippo** e l'istruzione per richiamarne un metodo.

```
$this->load->model('nome_model', 'pippo');

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

Altra possibilità offerta da CodeIgnition è il caricamento automatico di un determinato Model durante l'inizializzazione del sistema. È possibile definire il Model in questione configurando l'array **autoload** nel file `/application/config/autoload.php`.

## VISUALIZZARE LE NOTIZIE

Ora che le query sono state realizzate, il Model deve essere collegato alle Viste: ci si prepari quindi a rappresentare su schermo le due notizie che sono state inserite nella tabella. Si dovrà fare affidamento ad un Controller, ad una o più Viste e al layer di astrazione *Active Record* incluso in CodeIgniter. Quest'ultimo è un potente strumento che interpreta le istruzioni per il database rendendole indipendenti dalla soluzione adottata per il progetto. Ritorneremo in seguito sull'argomento, ma per il momento ci si accontenti di sapere che Active Record permette di passare da un database MySQL ad uno completamente differente (che utilizza un altro set di istruzioni), senza che si verifichino incompatibilità o problemi di alcun tipo.

### Definizione 5: Istruzioni SQL

---

<b>Istruzioni SQL</b>	Le query sono istruzioni per interfacciarsi con il database, per cancellare, memorizzare o richiedere i record.
-----------------------	---

---

Ritorniamo allo sviluppo del nostro progetto e precisamente alla definizione del Controller **News**:

1. Creiamo il nuovo file **news.php**

2. Il file in questione dovrà trovarsi nel percorso `/application/controllers/`
3. Inseriamo il codice seguente:

```
<?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);
    }
}
```

Il Controller non si limita a caricare “solo” il Modello con l’istruzione `$this->load->model('news_model')` ma in esso vengono definiti anche due importanti metodi: **index()** e **view()**. Esaminiamoli.

1. **index()** è il metodo predefinito che caricherà tutte le notizie (al momento non è completo)
2. **view()** ha un compito preciso: caricare solo una notizia tra tutte quelle memorizzate nel database. Questa sarà individuata dalla variabile `$slug` che è l’indice con cui si è marcata ogni notizia
3. I più attenti avranno notato che i due metodi descritti sono preceduti dal metodo costruttore `__construct()`: il suo compito è richiamare il costruttore della classe padre `CI_Controller` e caricare il modello, che potrà essere utilizzato per tutti i metodi definiti all’interno del Controller

Dopo aver definito lo “scheletro” della classe **News**, analizziamo meglio il metodo **index()** e definiamo la sua implementazione.

```
public function index()
{
    $data['news'] = $this->news_model->get_news();
    $data['titolo'] = 'Lista delle notizie';

    $this->load->view('header', $data);
    $this->load->view('menu', $data);
    $this->load->view('main', $data);
    $this->load->view('footer');
}
```

Il metodo contiene un array **data** con due indici **news** e **titolo**. Il primo assume tutti i valori delle notizie attraverso l'istruzione `news_model->get_news()` di cui si occupa il Model, come si è già visto. Invece il **titolo** del medesimo array assume il valore di una stringa prefissata, ovvero il titolo della nostra pagina principale per cui abbiamo scelto il poco originale "Lista delle notizie". Le istruzioni successive caricano le Viste che compongono il template completo (in alcune di esse viene passato anche l'array `$data` per utilizzarne i dati memorizzati).

Ora che abbiamo definito il Controller e il Modello `News_model`, dobbiamo solo ridefinire le Viste per apprezzare il risultato di tutto questo duro lavoro. Si apra il file della Vista creata in precedenza **main.php** quindi si inserisca il codice:

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

    <h2><?php echo $news_item['titolo'] ?></h2>
    <div id="main">
        <?php echo $news_item['testo'] ?>
    </div>
    <p><a href="news/<?php echo $news_item['slug'] ?>">
        Visualizza la notizia</a></p>

<?php endforeach ?>
```

Un primo risultato incoraggiante è visibile ricaricando la pagina:

```
http://127.0.0.1/codeignition/index.php/news/
```

Il Controller caricherà il metodo predefinito e visualizzerà tutta la lista delle notizie memorizzate nel database.

## VISUALIZZARE LA NOTIZIA MARCATA

Per visualizzare solo una certa notizia è stato fissato un indice definito dall'argomento `$slug`. Terminiamo il lavoro completando l'implementazione della funzione `function view($slug)` nel Controller.

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

    if (empty($data['news_item']))
    {
        show_404();
    }

    $data['titolo'] = $data['news_item']['titolo'];

    $this->load->view('header', $data);
    $this->load->view('menu', $data);
    $this->load->view('unanews', $data);
    $this->load->view('footer');
}
```

Alcuni punti sono inediti, vediamoli:

1. Viene effettuato un controllo preventivo. Se non esiste alcuna notizia con l'argomento passato **slug**, l'array `$data['news_item']` sarà nullo: la funzione **empty()**, nella riga successiva, verifica proprio questa situazione. Se l'istruzione condizione *if* è true, ovvero se non esiste alcun indice slug corrispondente ad una notizia, verrà visualizzato un messaggio di errore tramite `show_404()`
2. `$data['titolo'] = $data['news_item']['titolo']` questa istruzione definisce un array bidimensionale che memorizza al suo interno il titolo della notizia identificata dall'indice `news_item`

Inoltre nella parte riguardante le Viste, si può osservare come ora venga richiamato un altro template, **unanews**. Creiamo quindi anche questa nuova Vista **unanews.php** e inseriamo le istruzioni per caricare la notizia identificata dal marcatore.

```
<?php
echo '<h2>'.$news_item['titolo'].'</h2>';
echo $news_item['testo'];
```

In sintesi, se verrà fornito un argomento corrispondente ad un valore `$slug` esistente, la notizia relativa verrà caricata. Al contrario se non viene fornito alcun argomento, verranno visualizzate tutte le notizie.

## ROUTING

Per visualizzare anche le singole notizie è necessario apportare alcune modifiche al routing, creando un “route” extra per visualizzare il controller appena creato. Si apra il file **route.php** nel percorso **/application/config/** e nella sezione apposita si sostituiscano (o meglio si commentino) le righe esistenti con le seguenti:

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

### Definizione 6: Commentare il codice

<b>Commentare il codice</b>	Quando si deve modificare del codice di cui non si è sicuri se potrà essere utile o meno in futuro, è consigliabile commentare le righe che lo contengono, piuttosto che cancellarle. In questo modo sarà sempre possibile ripristinare la precedente situazione semplicemente eliminando i simboli di commento che in <a href="#">PHP</a> sono rappresentati da “//”
-----------------------------	---

Si scriva nel browser l’indirizzo del controller e si ammiri il risultato ottenuto:

```
http://127.0.0.1/codeigniter/index.php/news
```

Il codice si avvale delle espressioni regolari che costituisce un argomento dell’informatica a se stante, che consigliamo di apprendere soprattutto se si è interessati a definire pattern di ricerca.

## INSERIRE UNA NOTIZIA

Precedentemente le notizie sono state inserite tramite una istruzione SQL e anche se l’interfaccia di phpMyAdmin è amichevole, esistono altri sistemi più intuitivi e sicuri per memorizzare informazioni nel database. Uno di questi prevede l’uso di form, ovvero pagine con semplici interfacce per l’inserimento di input. Quindi se nella sezione precedente il nostro obiettivo è stato quello di “leggere” i record dalla base di dati, ora ci occuperemo di “memorizzare” le notizie.

Innanzitutto si definisca una nuova Vista con il codice per visualizzare il form, al percorso **/application/views/** e caratterizzata dal nome **creanews.php**. Il form sarà composto da due campi: uno per l’inserimento del titolo ed uno per quello del testo.

Rispetto a quanto visto prima, non si dovrà inserire il marcatore slug, in quanto si svilupperà un sistema per la creazione automatica di un marcatore univoco.

Ecco il codice [PHP](#) e [HTML](#):

```
<h2>Inserisci una notizia</h2>

<?php echo validation_errors(); ?>

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

    <label for="titolo">Titolo</label>
    <input type="input" name="titolo" /><br />

    <label for="testo">Testo</label>
    <textarea name="testo"></textarea><br />

    <input type="submit" name="invia" value="Crea una news" />

</form>
```

Probabilmente sono due gli elementi che appaiono poco familiari. Il metodo `form_open()` e `validation_errors()`. Per il momento sappiate che il primo è un Helper che visualizza i campi del form e aggiunge funzionalità come un “CSRF prevention field nascosto”. Il secondo è utilizzato per visualizzare eventuali errori nell’inserimento di input anomali.

#### Definizione 7: Validazione

<b>Validazione</b>	La validazione dei form è generalmente basato sulle Regex. Il loro scopo è comprendere e se i dati inseriti sono formalmente corretti. Per esempio un form genererà un errore se l’utente inserisce una email priva del caratteristico e assolutamente immancabile simbolo @
--------------------	--

Torniamo indietro al Controller **news.php** e aggiungiamo il codice necessario a caricare gli Helper relativi al form (`$this->load->helper('form');`) e la Libreria per la validazione dei form (`$this->load->library('form_validation')`).

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

    $data['titolo'] = 'Inserisci una news';

    $this->form_validation->set_rules('titolo', 'Titolo', 'required');
    $this->form_validation->set_rules('testo', 'testo', 'required');

    if ($this->form_validation->run() === FALSE)
    {
        $this->load->view('header', $data);
        $this->load->view('creanews');
        $this->load->view('footer');
    }
    else
    {
        $this->news_model->set_news();
        $this->load->view('success');
    }
}
```

Le regole per la validazione del form vengono esplicitate tramite `set_rules()` che prevede tre argomenti: il nome del campo dell'input, il nome da utilizzare nel messaggio di errore e la regola (nel nostro caso "required" significa "obbligatorio").

È importante soffermarsi sull'istruzione condizionale:

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

Che si può interpretare come: "se la condizione per la validazione dell'input non verrà superata, il programma caricherà nuovamente la pagina" (mediante le Viste che seguono). In caso contrario (istruzione condizionale **else**) verrà caricato il Modello e sarà memorizzata la notizia (`set_news()`) nel database. Infine una nuova pagina confermerà il buon esito dell'operazione **view('success')**.

Questo è solo un esempio della potenza messa a disposizione da CodeIgniter tramite la libreria di validazione. Maggiori informazioni sono disponibili al capitolo che tratta in dettaglio l'argomento a pagina [245](#).

Definiamo ora una nuova Vista **successo.php** nel percorso `/application/view/-news/` e scriviamo un confortante messaggio con il seguente codice:

```
<h2><?php echo "Notizia creata correttamente" ?></h2>
```

## INFINE IL MODELLO

Ormai siamo ad un passo dal raggiungere l'obiettivo prefissato: l'unica cosa che manca è il metodo che memorizza i dati inseriti nel database. Useremo la classe Active Record di CodeIgniter per semplificare il procedimento. Si apra il Modello creato precedentemente e si aggiunga:

```
public function set_news()
{
    $this->load->helper('url');

    $slug = url_title($this->input->post('titolo'), 'dash', TRUE);

    $data = array(
        'titolo' => $this->input->post('titolo'),
        'slug' => $slug,
        'testo' => $this->input->post('testo')
    );

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

Questo nuovo metodo provvede a memorizzare correttamente i dati nel database: analizziamolo nel dettaglio. La terza linea contiene una nuova funzione `url_title()` fornita dall'Helper URL (si veda a pagina 422) che sostituisce tutti gli spazi della stringa inserita come argomento con dei simboli “dash” (-) e nel contempo, rende minuscolo ogni carattere della stringa. Questa è una funzione utilissima per creare degli URI corretti partendo da una stringa. Il risultato di questa funzione sarà assegnato alla variabile `$slug`: ecco così creato un indice automatizzato. Per evitare ambiguità ci si aspetta però che i titoli delle notizie siano tutti differenti.

Attraverso l'istruzione `$this->input->post` il titolo e il testo della notizia vengono memorizzati preventivamente nell'array `$data`. Il metodo `post()` è fornito dalla Libreria input che svolge anche una funzione importante: rende sicuri i dati evitando che codice malevole possa essere memorizzato e richiamato come input di dati. La libreria input è, proprio per il suo importante compito, caricata automaticamente dal framework. Alla fine tutti i campi dell'array `$data` saranno inseriti correttamente nel database con l'istruzione `db->insert()`;

## NUOVE MODIFICHE AL ROUTING

Prima di procedere, aggiungendo le notizie al database, è necessario modificare nuovamente il file `routes.php` per prevedere appunto la rotta per la “creazione di notizia” (`creanews`). Riportiamo tutte le regole route per completezza:



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

Ora si punti il browser all'indirizzo:

```
http://127.0.0.1/codeigniter/index.php/news/creanews
```

## CONNESSIONE AL DATABASE

Quando un Modello viene caricato, questo non permette una connessione automatica al proprio database, come già rimarcato in precedenza. Esistono tre possibili opzioni, che per completezza elenchiamo:

1. É possibile collegarsi usando i metodi standard forniti dalla classe del Controller o da quella del Modello (si veda la sezione a pagina [172](#))
2. Si può richiamare il metodo che carica automaticamente il modello con un argomento booleano TRUE. In questo modo, si recuperanno i dati di accesso al database da `/application/config/config.php`

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

3. É possibile passare manualmente i dati per l'accesso al database attraverso un terzo parametro, solitamente un array che contiene tutti i dati necessari:

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

## 4.7 RIEPILOGO

Congratulazioni, è stata realizzata la prima applicazione con CodeIgniter sfruttando il patter [MVC](#), utilizzando il Controller, la View e interfacciandosi con un database grazie al Model. Nonostante l'utilizzo di un database sia opzionale, la semplicità con cui può essere implementato e gli innumerevoli vantaggi che ne derivano dall'adozione lo rendono una scelta presente nella maggior parte dei progetti. Facilità di accesso ai dati, gestione semplificata e soprattutto la loro riusabilità in diversi contesti, fanno sì che l'accoppiata Model/database vengano utilizzati anche in semplici progetti. Si è potuto dare un primo sguardo agli Helper e alle Librerie apprezzando la loro utilità e soprattutto la loro potenza in fatto di sicurezza e validazione dei dati. Anche la sintassi degli [URI](#) utilizzata nel framework dovrebbe essere ormai più familiare.

Sono state scalfite solo alcune potenzialità di CodeIgniter e non ci si disperi se alcuni aspetti sono ancora poco chiari. Nei prossimi capitoli ritorneremo con maggiori dettagli su molte nozioni sin qui introdotte.

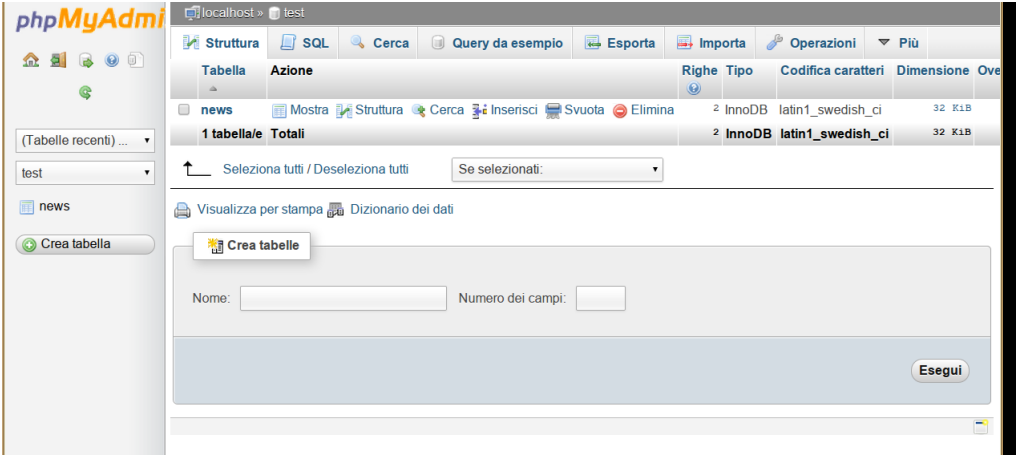


Fig. 13: Tabella test

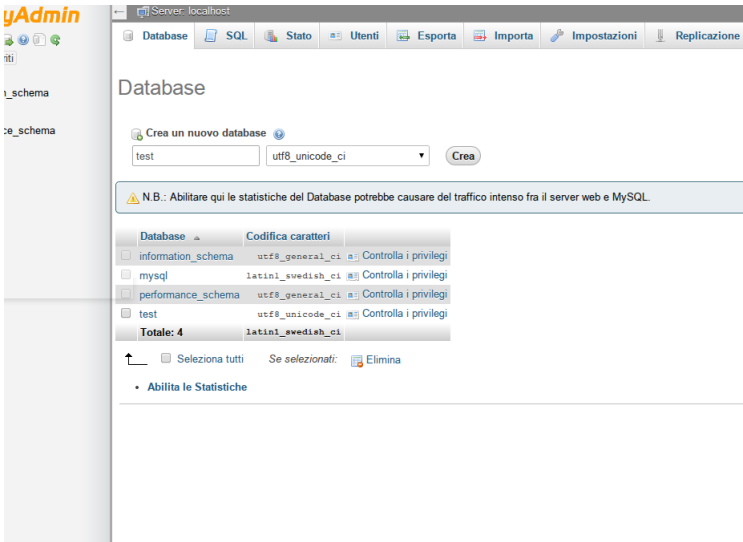


Fig. 14: Creazione di un nuovo database



Fig. 15: Inserimento della query

# 5

## ARGOMENTO AVANZATO I

Nel capitolo in cui si è introdotto per la prima volta il Controller (si veda la sezione [4.1 a pagina 33](#)) si è accennato alla gestione degli [URL](#) in CodeIgniter e a come essi vengono rappresentati. Nello schema adottato da CodeIgniter possono essere individuati i seguenti elementi:

```
miosito.com/[controller]/[metodo]/[argomenti]
```

**MIOSITO.COM** è l'indirizzo della nostra macchina remota o server locale

**CONTROLLER** è la classe principale che viene invocata per instradare la nostra richiesta

**METODO** rappresenta la funzione svolta dal Controller

**ARGOMENTI** (facoltativi) sono le variabili che vengono passati al metodo

### 5.1 GESTIONE DEGLI URL

Un aspetto peculiare dei moderni framework, a cui CodeIgniter non fa eccezione, è la gestione semplificata degli [URL](#), ovvero degli indirizzi che caratterizzano il progetto e che vengono visualizzati nella barra del browser web. Chi ha un po' di dimestichezza con la programmazione [PHP](#) e il passaggio dei parametri, sa bene che molti indirizzi verranno visualizzati nel formato:

```
http://www.miosito.com/cartella/val1=pippo&val2=pluto
```

Dove "val1" e "val2" rappresentano due variabili dal valore, rispettivamente di "pippo" e "pluto". Indirizzi di questo tipo, sono consuetudine nel web, soprattutto nei siti commerciali o nei motori di ricerca (si osservi a tal proposito la barra del browser dopo aver effettuato una ricerca su Amazon o Google). Gli indirizzi di questo tipo, definiti "query string" sono però suscettibili di qualche critica.

- non sono mnemonici: questo è un dato di fatto
- vengono visualizzati i nomi delle variabili e i loro valori: questo rende il sito meno impenetrabile. Anche se le comuni regole di sicurezza risolvono questo genere di lacune, sul web è sempre meglio precludere ai malintenzionati il maggior numero di informazioni sensibili

Codice 5.1: Esempio di URL basato su query string

```
index.php?c=prodottis&m=view&id=345
```

In virtù di quanto esposto sopra, CodeIgniter utilizza un sistema di gestione degli [URL](#) che è nel contempo sia *search-engine friendly* (ottimizzato per l'indicizzazione e il posizionamento delle pagine nei motori di ricerca), sia *human friendly* (semplice da ricordare e digitare per gli utenti). Come funziona? Semplice, gli [URL](#) basati sul sistema query string vengono riscritti basandosi sulla "segmentazione". In parole povere, le variabili e i loro parametri diventano meno espliciti all'utente, e ogni porzione dell'[URL](#) è suddiviso in "/" (slash). Questa prassi produce indirizzo dalla lunghezza più contenuta, facili da ricordare e soprattutto renderà meno gravoso il lavoro SEO per l'indicizzazione del sito nei principali motori di ricerca.

## 5.2 ABILITARE LE QUERY STRING

Se quanto detto sinora non vi convincesse, potete continuare ad usare il sistema query string, disabilitando quello che CodeIgniter vi propone di default.

È sufficiente aprire il file **config.php** e modificare alcune variabili all'interno:

```
$config['enable_query_strings'] = TRUE; // per abilitare le query string
$config['controller_trigger'] = 'c';
$config['function_trigger'] = 'm';
```

Se si sceglie di impostare su TRUE la variabile `enable_query_strings` i controller e i metodi saranno accessibili con le parole *trigger* specificate nel file **config.php**. Di default esse sono rispettivamente **c** e **m**. Esempio:

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

Nota: utilizzando gli url sotto forma di query string si dovrà prestare attenzione all'utilizzo degli Helper per la generazione degli [URI](#), che tengono conto della soluzione adottata.

## 5.3 AGGIUNGERE UN SUFFISSO URL

È possibile aggiungere un suffisso alla parte terminale degli [URL](#) generati, modificando la sezione **URL suffix** del file **config.php** che, si ricorda si trova al percorso `/application/config/`.

La stringa vuota `$config['url_suffix'] = ''`; può essere valorizzata, per esempio, con “.html” in modo da cambiare un indirizzo da:

```
http://mioSito.com/index.php/pages/news/1
```

in:

```
http://mioSito.com/index.php/pages/news/1.html
```

## 5.4 UN FILE INDEX DI TROPPO

Trattando un altro aspetto degli indirizzi, è possibile notare come CodeIgniter inserisca all'interno del [URL](#), la stringa **index.php**. Nel caso si desideri eliminare questa informazione dal percorso, si dovrà modificare il file **.htaccess**.

Nell'esempio seguente si utilizza un metodo “negativo” con cui ogni elemento viene reindirizzato eccetto gli elementi specificati: index. php, images e robot. txt.

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

## 5.5 HELPER E PLUGIN

Chi mastica un po' di inglese avrà già compreso la funzione di questi preziosi strumenti a corredo di CodeIgniter. Gli Helper sono degli "aiutanti", ovvero delle librerie di funzioni che rendono agevoli le operazioni più ripetitive nello sviluppo. Divisi in categorie, gli Helper sono ideati per la gestione degli array, dei file, delle date, delle sessioni, dei form, della posta elettronica e molto altro. Nonostante sia necessario un po' di tempo per apprendere le loro funzionalità (niente che con la pratica e la curiosità non si possa risolvere), la loro adozione presenta indiscutibili vantaggi, anche perché lo sviluppatore deve concentrarsi sull'interfaccia degli Helper e non sulla loro implementazione.

## 5.6 INTERFACCIA E IMPLEMENTAZIONE

Questi due concetti sono spesso presenti nella programmazione, soprattutto quando si parla di "quella orientata agli oggetti". Senza approfondire troppo l'argomento, riteniamo utile presentarli. I termini "interfaccia e implementazione" non sono comuni solo alla programmazione di qualche oscuro codice, ma sono degli aspetti che caratterizzano l'interazione di ognuno di noi con il mondo esterno. Immaginatevi al volante della vostra automobile che guidate beatamente in mezzo al traffico: sapete che ruotando lo sterzo a destra o a sinistra, la macchina si sposta nella relativa direzione. E conoscete anche la funzione dei pedali, il freno a mano e dei vari pulsanti presenti sul cruscotto. Bene, ma avete mai smontato il volante per comprendere i meccanismi o esaminato l'elettronica che rende possibili tutte queste funzionalità? Immagino che la risposta più gettonata sia: "No! E non ci penso neppure!". Comunque non preoccupatevi, non intendiamo aprire un corso sul come assemblare da soli la propria fuoriserie in comode uscite settimanali: i risultati sarebbe nefasti, soprattutto per la vostra automobile.

Questa digressione ci è servita per porre l'accento sulle differenze tra l'interfaccia e l'implementazione. La prima, in una macchina, è costituita dal volante, dai pedali e da tutti i pulsanti che la caratterizzano. Non dobbiamo essere degli ingegneri meccanici per guidare un'auto, basta aver studiato il codice della strada e aver acquisito un po' di dimestichezza con il veicolo. Insomma, se per noi l'automobile è un mezzo per andare dal punto A a quello B, concorderemo che sia meglio che di meccanica e di elettronica siano altri ad occuparsene. Tutti quegli oscuri meccanismi che si trovano "all'interno" dell'auto e che permettono di sterzare quando si gira il volante, o di arrestarsi se si preme l'apposito pedale costituisce l'implementazione. Riassumendo: possiamo guidare un'auto conoscendo la sua interfaccia, pur rimanendo all'oscuro dei dettagli interni, che rappresentano l'implementazione.

Questo è un discorso aiuta a comprendere meglio l'utilità degli Helper, perché questi aiutanti forniscono numerose funzioni già pronte, ma non impongono a chi le utilizza di conoscere il codice interno. Insomma, anche chi non padroneggia a menadito il [PHP](#), [Structured Query Language \(SQL\)](#), o altri linguaggi, potrà sfruttarne le potenzialità. La separazione tra interfaccia e implementazione presenta anche altri vantaggi:

**MINORI CONOSCENZE TECNICHE** È faticoso conoscere tanti linguaggi di programma-



zione, senza tener conto che ad ogni loro nuova release molte funzioni vengono introdotte o deprecate (sconsigliate)

**REINVENTARE LA RUOTA** Anche se siamo degli ottimi programmatori e potremmo voler sviluppare da soli le funzioni, e ogni più piccolo aspetto del progetto, si tenga conto che questo porterà via molto tempo. Non c'è niente di più dispendioso che reinventare la ruota ad ogni nuovo progetto

**IL CODICE ALTRUI É SEMPRE PIÙ VERDE** Il codice degli Helper viene esaminato, sviluppato e corretto dal team di CodeIgniter e da tanti volenterosi collaboratori: con tutta probabilità realizzeranno un codice migliore del nostro

## 5.7 GLI HELPER NEL DETTAGLIO

CodeIgniter a differenza di altri framework non realizza l'implementazione dei propri Helper sulla base del paradigma OOP (Object Oriented Programming o Programmazione Orientata agli Oggetti) ma secondo l'approccio procedurale, ovvero come comuni funzioni. Se questo è un bene o un male, sta a voi deciderlo. Inoltre, ogni aiutante costituisce una funzione a se stante, senza dipendenze che renda obbligatorio l'uso di altri Helper: questo, unito all'utilizzo di un codice procedurale, le rende estremamente comprensibili.

Un Helper può essere caricato da un Controller, ma anche da una View (anche se questa pratica è sconsigliata dagli stessi sviluppatori di CodeIgniter). Inoltre, è possibile caricare gli aiutanti nel costruttore del Controller, in modo che diventino automaticamente disponibili in qualsiasi metodo, oppure definirli in maniera che vengano eseguiti manualmente solo quando necessario. Questo aspetto è molto importante perché non precaricando automaticamente gli Helper, CodeIgniter fornisce un sistema decisamente più snello e meno avaro di risorse. Sarà lo sviluppatore che, se necessario, invocherà l'Helper desiderato. In tutti i casi, una volta che questo verrà caricato, sarà disponibile per qualsiasi Controller e Vista del progetto.

Esistono due percorsi in cui sono presenti gli Helper: **application/helpers** (il percorso predefinito in cui inizialmente viene ricercato) e **system/helpers**.

Un Helper viene richiamato attraverso una sintassi che dovrebbe essere ormai familiare:

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

Nota: l'Helper non restituisce alcun valore, quindi non è mai possibile assegnarlo ad una variabile.

Per esempio, se si desidera caricare l'Helper per la gestione degli array, il comando appropriato sarà:

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

Se si ha l'esigenza di caricare più Helper con un solo comando, basta specificarne i nomi separandoli con una virgola. Con questa istruzione abbiamo caricato gli aiutanti per array, le date e i file.

```
$this->load->helper('array', 'date', 'file');
```

## UTILIZZO

Abbiamo appreso i vantaggi di un Helper, sappiamo come caricarli, ma come si gestiscono esattamente? Ogni aiutante possiede una specifica documentazione, che illustra la sua funzione e come utilizzarlo nel dettaglio. Esaminiamo questo esempio:

```
// caricamento dell'Helper per la gestione degli array
$this->load->helper('array');

// definizione di un array
$mioarray = array('nome' => 'giuseppe',
    'cognome' => 'bellisano',
    'citta' => 'cagliari'
);

// restituzione di un valore tramite funzione messa a disposizione dall'Helper
echo element('nome', $mioarray);
```

Analizziamo il codice suddividendolo per punti:

- Viene caricato l'Helper demandato alla gestione degli array
- L'array è composto da tre indici (nome, cognome, città) a cui rispettivamente si assegnano tre valori (giuseppe, bellisano, cagliari) secondo la sintassi propria del linguaggio [PHP](#)
- la funzione `element()` svolge il compito di visualizzare un elemento dell'array. Nel nostro caso il primo argomento è l'indice, mentre il secondo è l'array da prendere in considerazione). Il risultato sarà "giuseppe".

Un altro esempio è dato dall'Helper **anchor()** riportato qui di seguito, che utilizzato in una Vista permette di creare facilmente un link. Il nome del collegamento è definito dalla stringa **Clicca qui**, mentre **blog/commenti** costituisce l'[URI](#) del controller/metodo che intendiamo linkare.

```
<?php echo anchor('blog/commenti', 'Clicca qui');?>
```

## AUTOLOAD

Anche se non consigliato, gli Helper possono essere caricati in anticipo in modo da fruire delle loro caratteristiche. Si ricorda che l'autoload degli Helper influisce negativamente sulle prestazioni dell'applicazione in fase di produzione. Questa funzionalità può essere definita specificando il nome dell'aiutante desiderato nel file **autoload.php** che si trova nel percorso **/application/config/**. Editando il file **autoload.php** con il seguente codice, si effettuerà per esempio il precaricamento degli Helper che gestiscono gli array e le date:

```
$autoload['helper'] = array('array', 'date');
```

## ESTENDERE GLI HELPER

Gli aiutanti predefiniti in CodeIgniter possono essere estesi creando un file in **/application/helpers/** con il medesimo nome, ma con il prefisso **MY\_**. L'Helper così creato potrà contenere nuove funzioni o sovrascrivere quelle esistenti con quelle definite nel proprio codice (pratica caldamente sconsigliata). Si presti attenzione che il termine *estendere* è proprio della programmazione **OOP** mentre gli Helper sono sviluppati secondo un approccio procedurale: riscrivere una funzione esistente, in questo caso, la sovrascriverebbe, eliminando di fatto l'implementazione nativa.

Come esempio citiamo l'Helper **Array** le cui funzioni potranno essere estese o sovrascritte creando **MY\_array\_helper.php** al percorso **/application/helpers/**

```
// any_in_array() è una nuova funzione da noi realizzata. Non è presente nell'
// Helper Array, quindi non è sovrascritta alcuna funzione nativa
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() è inclusa nell'Helper originale
// quindi la nostra funzione è sovrascritta quella nativa di CodeIgniter
function random_element($array)
{
    shuffle($array);
    return array_pop($array);
}
```

## UN PREFISSO PERSONALIZZATO

Se non vi aggrada, è possibile personalizzare il prefisso MY\_ utilizzato per estendere gli Helper. Il sistema illustrato si applica anche alle estensioni delle Librerie e delle Classi core. Si apra quindi il file `/application/config/config.php` e si cerchi la seguente riga:

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

Ora è possibile sostituire il prefisso con tutti quelli che vi passano per la testa. Quasi tutti, perché il prefisso CI\_ è riservato al core di CodeIgniter. Non usatelo!

## 5.8 PLUGIN RIDEFINITI

Per molti aspetti i Plugin possono essere identificati con gli Helper esaminati nel precedente capitolo. Infatti condividono molti elementi in comune, come il fatto di essere delle funzioni che aiutano lo sviluppatore in compiti precisi, senza per questo essere costretti a conoscere la loro implementazione. Ma se sono identici,

perché chiamarli in modo differente? Mentre gli Helper sono integrati nel core del framework e realizzati direttamente dal team di CodeIgniter, i Plugin appaiono più come moduli esterni, evoluti, solitamente prodotti da sviluppatori esterni, indicati con il termine di “vendor”. I Plugin rispetto agli Helper, inoltre non mettono a disposizione una serie di tool per svolgere i più disparati compiti, ma si concentrano unicamente su una funzione specialistica. Un esempio è dato dalla sistema per la gestione del Completely Automated Public Turing test to tell Computers and Humans Apart ([CAPTCHA](#)), che implementa l’odiato sistema per scovare i bot attraverso il riconoscimento difficoltoso di caratteri presentati sullo schermo. Quante volte si sarà esclamato: “Cosa è quello strano geroglifico?!? Sì forse ho capito, ma dove si trova nella tastiera il tasto “zampa di coniglio”?).

Nonostante queste premesse, la loro distinzione non è parsa così netta neppure al team di CodeIgniter, che a partire dalla versione 2.0.0 ha optato per la loro rimozione, accorpandoli con gli Helper. Se quindi utilizzate una versione precedente del framework, troverete i plugin al percorso **/application/plugins**.

È consigliabile utilizzare comunque l’ultima versione del framework, reperibile sul sito ufficiale: si beneficerà sempre di maggiori funzionalità, oltre che di un codice maggiormente ottimizzato ed esente da bug. Le funzionalità dei Plugin non sono comunque andate perse: semplicemente ora si trovano (compreso quello per il CAPTCHA) nella cartella degli Helper **/application/helpers**.

## 5.9 RIEPILOGO

Ora abbiamo imparato a gestire un nuovo tassello del framework. Gli Helper sono delle funzionalità interne a CodeIgniter che permettono di gestire facilmente molti aspetti del nostro progetto conoscendo semplicemente la loro interfaccia, senza preoccuparci del codice all’interno (implementazione). Si tratta di un grosso vantaggio per snellire le parti più ripetitive e noiose di un progetto in via di sviluppo.

CodeIgniter mette a disposizione anche delle funzioni estremamente specifiche che svolgono importanti compiti per la sicurezza o l’autenticazione dell’utente nel nostro progetto. Il termine Plugin, a partire dalla versione 2.0 del framework, scompare a favore di una loro integrazione con gli Helper.



# 6

## ARGOMENTO AVANZATO II

### 6.1 LE LIBRERIE

Per gli anglofili le Libraries sono, come si evince dal nome stesso, delle “raccolte” di classi suddivise in categorie omogenee. Ogni Libreria viene in aiuto dello sviluppatore per lo svolgimento dei compiti più ripetitivi e meritevoli di grande attenzione come le classi per la gestione della posta elettronica l’upload e il download dei file, la criptazione o la validazione degli input nei form. Anche in questo caso, il vantaggio per lo sviluppatore, è che non si rende necessario conoscere l’implementazione delle classi, ma semplicemente la loro interfaccia pubblica.

Le Librerie predefinite di CodeIgniter si trovano nel percorso `/system/libraries/` mentre per caricarle nel progetto sarà necessaria una funzione con la sintassi:

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

La seguente istruzione per esempio carica la libreria adibita alla gestione della posta elettronica:

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

Per caricare più raccolte di classi, si utilizza un array in cui si specificano le Librerie desiderate. Nell’esempio, verranno caricate nello stesso tempo, quelle dedicate alle **email** e alle **table** (tabelle).

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

Una volta caricata la Libreria si potranno utilizzare le classi definite al suo interno per i più svariati compiti, secondo una sintassi comune del tipo:

```
$this->nome_libreria->nome_funzione('argomento');
```

Quindi se si volesse inviare una email ad un preciso destinatario, si scriverà:

```
$this->email->to('info@tuosito.com');
```

- email. È il nome della Libreria precedentemente caricata
- to. È il nome della classe demandata ad individuare il destinatario dell'email
- info@tuosito.com. Ovviamente è l'argomento della nostra classe, ovvero colui che riceverà la missiva

Qui di seguito alcune delle classi più comuni della Libreria **email**, precedute dall'istruzione che, si tenga sempre a mente, deve precaricare tale Libreria.

```
// Invio di una email con una Libreria di CodeIgniter

// caricamento della Libreria per la gestione delle email
$this->load->library('email');

// mittente dell'email
$this->email->from('mail@miosito.com', 'Giuseppe');

// destinatario dell'email
$this->email->to('info@tuosito.com');

// indirizzi e-mail del destinatario in copia conoscenza
$this->email->cc('posta@lorosito.com');

// indirizzo e-mail del destinatario in copia conoscenza nascosta
$this->email->bcc('max@boss.com');

// oggetto dell'email
$this->email->subject('Invio email con CodeIgniter');

// testo del messaggio
$this->email->message('Email inviata con la Library Email.');
```

```
// invio del messaggio
$this->email->send();
```

## CREARE UNA LIBRERIA

Il programmatore esperto, nonostante le numerose Librerie già presenti in CodeIgniter, potrebbe avere la necessità di crearne di nuove, di estendere o sovrascrivere quelle esistenti. Chi lo desiderasse può sviluppare le proprie classi e riutilizzarle in futuri progetti. Per separare le Librerie del framework dalle proprie, si utilizza il percorso **/application/libraries/**, in cui si inseriscono quelle realizzate dall'utente. Esistono alcune convenzioni che è bene tenere presente nella sintassi delle Librerie:

- nomi dei file. Questa volta, contrariamente a quando avveniva con le View, Controller e Model, i file contenenti le classi devono avere l'iniziale maiuscola. Per esempio **Nome-della-classe.php**



- le classi. Nel codice le classi saranno definite con la lettera iniziale maiuscola
- le estensioni. L'estensione del file sarà sempre “.php”.

L'esempio seguente mostra un prototipo della sintassi utilizzata per la definizione di una classe:

```
<?php
// controllo sul percorso della Library
if (!defined('BASEPATH')) exit('No direct script access allowed');

// dichiarazione della classe
class Nome_classe {
    // funzione di classe
    function funzione() {
        // operazioni ed eventuale valore di ritorno
        ...
    }
}
?>
```

L'istruzione condizionale *if* effettua una verifica sul percorso della Libreria e in caso di valore booleano “FALSE” interrompe l'accesso alle classi ivi definite. Si ricordiamo che la propria Libreria andrà salvata in un percorso differente rispetto a quello previsto per le Librerie di CodeIgniter, ovvero **/application/libraries/**.

Le istruzioni per caricare la propria Libreria e invocare la relativa classe, rispettano la sintassi esaminata precedentemente:

```
$this->load->library('nome_classe'); // carica la Libreria
$this->nomeclasse->funzione(); // utilizza il metodo funzione()
```

Nota: il nome della Libreria, nel nostro caso `nome_classe` può essere riportato tutto in minuscolo oppure con l'iniziale maiuscola: CodeIgniter non discrimina nessuno dei due sistemi. Quindi anche il seguente esempio è corretto.

```
$this->load->library('Nome_classe'); // iniziale maiuscola
```

## PASSAGGIO DI PARAMETRI

Come ogni funzione/metodo che si rispetti, è possibile passare dei parametri [4.2 a pagina 43](#) al metodo della classe quando questa viene “caricata”, come nell'esempio seguente:

```
// Passaggio dinamico di variabili ad una Library personalizzata
// definizione dei parametri sotto forma di array
$parametri = array('tipo' => 'auto', 'colore' => 'giallo');

$this->load->library('nome_classe', $parametri);
```

Dopo aver definito un array (nel nostro esempio con due indici e relativi valori), si carica la libreria `nome_classe` a cui si passa l'array assegnato alla variabile `$parametri`.

Per utilizzare questo sistema, è però obbligatorio configurare la classe costruttore affinché possa ricevere i dati “passati”:

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

class Nome_classe {

    public function __construct($parametri)
    {
        // Qualcosa riferito ai $parametri
    }
}
?>
```

Gli stessi parametri potranno comunque essere passati direttamente come argomento del metodo:

```
// Passaggio dinamico di variabili alla funzione di classe
function funzione($parametri) {
    // operazioni ed eventuale valore di ritorno
    ...
}
```

Nota: è prevista anche un'ulteriore possibilità: passare i parametri attraverso un file di configurazione. Basta creare un file con lo stesso identico nome della classe contenuta all'interno e memorizzarlo nel percorso `/application/config/`. Questo metodo non è però utilizzabile per il passaggio dinamico di parametri.

## ESTENSIONE E SOVRASCRITTURA

É possibile aggiungere delle nuove funzionalità alle Librerie di sistema, oppure sovrascriverle completamente, inserendo nel percorso `/application/libraries/` la nostra implementazione con lo stesso “identico” nome.

Se quindi creiamo una nostra personale Libreria per la gestione delle tabelle, basterà inserire il file **Table.php** con la definizione delle classi nel percorso **/application/libraries**. In questo modo, sovrascriveremo la Libreria predefinita di CodeIgniter situata invece in **/system/libraries/**.

Riepilogando:

- É possibile creare una nuova, inedita Libreria
- É possibile estendere quelle native
- É possibile rimpiazzare quelle native

Nota: le classi del *Database* sono la sola e unica eccezione tra le Librerie di CodeIgniter: esse non possono venire sovrascritte o estese.

## UTILIZZARE LE RISORSE DI CODEIGNITER

Per accedere alle risorse native di CodeIgniter con le proprie librerie è necessario utilizzare il metodo `get_instance()` che restituisce un super oggetto. Normalmente ogni funzione di CodeIgniter viene richiamata dal proprio Controller con il costrutto `$this`. Per esempio:

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

Il costrutto `$this` però, funziona solo all'interno dei propri Controller, Modelli, o Viste. Se invece si desiderano utilizzare le classi native dentro le proprie classi è necessario prima di tutto assegnare l'oggetto di CodeIgniter alla variabile:

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

Ora si utilizzerà `$CI` al posto della variabile `$this`.

```
$CI =& get_instance();  
  
$CI->load->helper('url');  
$CI->load->library('session');  
$CI->config->item('base_url');
```

Nota: si noterà che la funzione di cui sopra `get_instance()` viene passata per riferimento. Questo è molto importante perché utilizzando un'assegnazione per riferimento si può utilizzare l'originale oggetto di CodeIgniter piuttosto che una sua copia.

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

## SOVRASCRIVERE LE LIBRERIE NATIVE

Rinominando i nomi dei file delle proprie Librerie con quelle di CodeIgniter, si utilizzeranno le classi sviluppate autonomamente al posto di quelle native del framework. Per esempio se si è sviluppata una propria Libreria per la gestione delle “email” basterà creare un file di nome **Email.php**, inserirlo nel percorso **/application/libraries/** e dichiararlo con la sintassi:

```
class CI_Email {  
  
}
```

Nota: la maggior parte delle classi native rechneranno il prefisso **CI\_**.

Per caricare le proprie Librerie basterà utilizzare la funzione consueta per il caricamento:

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

Nota: come detto precedentemente la classe Database costituisce una eccezione: non può essere sostituita dalla propria versione.

## ESTENDERE LE LIBRERIE NATIVE

Per i più esigenti si possono introdurre nuove funzionalità non presenti in CodeIgniter: non sovrascrivendo l'intera Libreria nativa, ma semplicemente estendendola. Vi sono però due eccezioni:

- la dichiarazione della classe deve estendere quella del genitore
- il nome della nuova classe e del file che la contiene, devono avere il prefisso **MY\_** (questo aspetto è comunque configurabile)

Per esempio l'estensione della Libreria **Email** si ottiene creando il seguente file al percorso **/application/libraries/MY\_Email.php**

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

Se si rende necessario l'uso di un costruttore, si può estendere quello genitore:

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

## CARICARE UNA SOTTOCLASSE

Per caricare una propria sottoclasse, si dovrà usare una sintassi standard, senza utilizzare il proprio prefisso. Ad esempio, per caricare l'esempio precedente che estende la classe Email, si utilizzerà:

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

Una volta caricata, la variabile di classe sarà disponibile come la classe che si sta estendendo. Nel caso della classe email, per tutte le chiamate si utilizzerà:

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

## CONFIGURARE IL PREFISSO

Se si desidera modificare il prefisso standard previsto da CodeIgniter, basta definire il valore assegnato nel consueto file **config.php** in **/application/config/**.

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

## 6.2 I DRIVER

I Driver sono delle “speciali” Librerie che possiedono una classe genitore e un, potenzialmente, infinito numero di classi figlie. La peculiarità di queste ultime è che possono accedere alla classe genitore, ma non a quelle dei “fratelli”. I Driver, insomma, costituiscono una soluzione elegante per ripartire la gestione delle Librerie in classi discrete.

I Driver si trovano in **/system/libraries/** all'interno di una cartella il cui nome è identico alla classe della Libreria genitore. All'interno di questa cartella vi è una sottocartella denominata **drivers**, che contiene tutte le sottoclassi figlio (si veda la struttura gerarchica in questa pagina).

Per utilizzare un Driver è necessario inizializzarlo con un Controller con la seguente funzione:

```
$this->load->driver('class_name'); // class_name è il nome della classe driver  
da invocare
```

Se si desidera caricare un Driver di nome “Alcuni genitori” si utilizzerà la funzione:

```
$this->load->driver('alcuni_genitori'); // alcuni_genitori è il nome della  
classe driver da invocare
```

I metodi di una classe saranno successivamente invocati:

```
$this->alcuni_genitori->alcuni_metodi();
```

Le classi figlio possono richiamare questi stessi metodi attraverso la classe genitore, senza però iniziarli:

```
$this->alcuni_genitori->figlio_uno->alcuni_metodi();  
$this->alcuni_genitori->figlio_due->altro_metodo();
```

## CREARE UN DRIVER

Qui di seguito si mostra un esempio della struttura gerarchica del Driver:

```
/application/libraries/Nome_driver  
Nome_driver.php  
drivers  
Nome_driver_sottoclasse_1.php  
Nome_driver_sottoclasse_2.php  
Nome_driver_sottoclasse_3.php
```

Nota: per mantenere la compatibilità nella struttura (i nomi sono case-sensitive) è consigliato utilizzare per Nome\_driver il metodo **ucfirst()** che rende l'iniziale maiuscola.

## RIEPILOGO

Abbiamo esaminato le Librerie, potenti raccolte di funzioni messe a disposizione da CodeIgniter per gestire molti elementi del nostro progetto come le operazioni di una email o la validazione dei campi di un forum, liberando da queste incombenze il programmatore. I Driver sono anch'esse delle Librerie con delle peculiarità importanti. É anche possibile sviluppare le proprie Librerie, aggiungere o ridefinire le classi in modo da personalizzare gli "attrezzi di lavoro" con una sola limitazione: la propria fantasia.

## 6.3 CLASSE CORE

### CREAZIONE DELLE CLASSI DI SISTEMA

Ogni volta che CodeIgniter è in azione, diverse classi di base vengono inizializzate automaticamente come parte del core del framework. È possibile spesso sostituire le classi del core system con quelle personali oppure estenderle. Solitamente si tratta di un argomento che interessa marginalmente gli sviluppatori che utilizzano CodeIgniter, anche perché modifiche non corrette minano la stabilità dell'intero framework. La possibilità comunque di personalizzare il funzionamento delle Librerie Core è un esempio della flessibilità di CodeIgniter.

### LISTA DELLE CLASSI DI SISTEMA

Qui di seguito una lista della Classi del core system che vengono eseguite ogni volta che CodeIgniter è in azione.

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

### SOSTITUIRE LE CLASSI DI SISTEMA

Per utilizzare una delle proprie classi invece di quelle predefinite è consigliabile inserire la propria versione all'interno del percorso **/application/core/** come per esempio:



```
application/core/alcune_classi.php
```

Nota: se la directory non esiste, è necessario crearla. La classe che si desidererà sostituire dovrà essere chiamata nello stesso identico modo di quelle definite nel core system, con l'aggiunta del prefisso **CI\_**. Per esempio, nel codice seguente la classe di sistema **Input.php** viene sovrascritta con una propria versione:

```
class CI_Input {  
}
```

## ESTENDERE LE CLASSI DI SISTEMA

Se invece si preferisce estendere le funzionalità di una Libreria di sistema, aggiungendo una, o due funzionalità, allora è meglio operare in un altro modo, meno invasivo per il framework. Non ci sono comunque controindicazioni nell'estendere una classe, tranne che per due punti:

- la dichiarazione della classe deve estendere quella del genitore
- la nuova classe e il nome del file devono contenere il prefisso **MY\_** (comunque configurabile)

Per esempio per estendere la classe nativa **Input** è sufficiente creare un file **MY\_Input.php** nella directory **/application/core/** e dichiarare la propria classe come:

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

Nota: se si utilizza nella propria classe un costruttore, è necessario estendere anche il metodo costruttore genitore:

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

Nota: ogni funzione nella propria classe deve essere chiamata nello stesso identico modo della classe genitore che sarà usata al posto di quella nativa: questo sistema

è chiamato “sovrascrittura del metodo”. In questo modo ci si assicura che non si verifichino modifiche al core di CodeIgniter.

Se si desidera estendere la classe del core Controller, ci si assicuri di estendere la propria, nuova classe nel costruttore del proprio Controller.

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

## CONFIGURAZIONE DEL PREFISSO

Come accennato precedentemente anche il prefisso MY\_ può essere modificato: è necessario aprire il file `fil/application/config/config.php`, e settare il prefisso desiderato, con l'accortezza di utilizzare qualsiasi stringa, tranne quella riservata CI\_.

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

## 6.4 GLI HOOK

Gli Hook sono delle componenti del framework CodeIgniter con cui probabilmente molti sviluppatori non dovranno mai avere niente a che fare: è comunque importante essere consapevoli della loro esistenza. CodeIgnition è molto flessibile e ogni suo aspetto può essere facilmente configurato per adattarsi alle preferenze dello sviluppatore e al progetto stesso. Personalizzare però gli elementi core del framework è un lavoro che non dovrebbe mai essere svolto, se non da professionisti, altrimenti il rischio è quello di compromettere le funzionalità del tool di funzioni con conseguenze sulla loro stabilità ed efficienza.

Nonostante CodeIgniter funzioni in base ad struttura ben organizzata (vedi in questa pagina), è possibile modificare alcune operazioni nel normale flusso di esecuzione, come per esempio caricare un particolare script prima che venga eseguito un Controller, oppure subito dopo. Questa operazione si può attuare grazie ad un Hook che modifica il normale flusso di esecuzione senza agire sul core di CodeIgniter. Gli strumenti che CodeIgnition mette a disposizione dello sviluppatore per svolgere queste personalizzazioni avanzate, (senza compromettere la stabilità del core), prendono il nome di **Hook**.

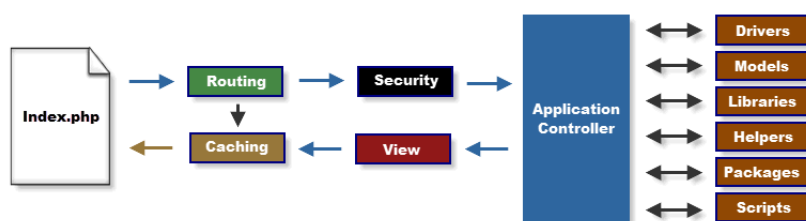


Fig. 16: Flow chart

Gli Hook si possono abilitare o meno, personalizzando il file **config.php** al percorso **application/config/config.php** (si veda sezione [3.1 a pagina 24](#)).

Modificando il valore booleano del parametro `$config['enable_hooks']` si potranno abilitare gli Hook (TRUE) o disabilitarli (FALSE). Quest'ultimo è il valore di default.

### UTILIZZARE GLI HOOK

La definizione degli Hook è semplice, e avviene agendo sul file **hooks.php** che si trova al percorso **application/config/hooks.php**. La sintassi utilizzata è quella propria di un array in cui è necessario specificare la classe, il metodo, il file, il percorso e gli eventuali parametri. Il seguente esempio mostra un prototipo:

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

Nota: la parte radice (index) dell'array che nel nostro esempio è `pre_controller`, serve per creare un punto di ancoraggio con l'Hook che si intende utilizzare. Inoltre nell'array sono presenti una serie di parametri/indici che devono essere opportunamente configurati:

**CLASS** il nome della classe che si desidera invocare. Se si preferisce usare una funzione procedurale invece di una classe, è necessario lasciare questo campo vuoto

**FUNCTION** si riferisce al nome della funzione da invocare

**FILENAME** è il nome del file che contiene la classe o la funzione

**FILEPATH** si riferisce al nome della directory che contiene il nostro script che si trova sempre all'interno della directory **application/**. Per esempio se lo script è all'interno di **application/hooks/**, ci si riferirà ad esso semplicemente con **hooks**. Se invece lo script, sempre facendo un altro esempio, si trova in **application/hooks/utilities/**, allora il percorso relativo da utilizzare sarà **hooks/utilities** senza lo slash / iniziale e finale

**PARAMS** ogni parametro, se lo si ritiene necessario, può essere passato al proprio script. I parametri sono opzionali

Per un maggiore ordine è vivamente consigliato inserire i propri Hook all'interno di una cartella creata per tale scopo. Se per esempio si volessero inserire gli script creati in una cartella chiamata "mieiscript", il relativo percorso sarà **application/hooks/Mieiscript** mentre il valore da definire nel **filepath** sarà di conseguenza **hooks/Mieiscript**.

## GLI HOOK POINT

Gli array in cui sono definiti i parametri descritti prendono il nome di Hook point. Ne esistono diversi, realizzati dagli sviluppatori di CodeIgniter e già pronti all'uso. Quello seguente ne è una parte:

- `pre_system`: viene chiamato immediatamente dopo la fase di esecuzione, permette unicamente il caricamento delle classi per il benchmark e quelle relative agli Hook stessi;
- `pre_controller`: ha la massima priorità: viene chiamato prima di qualsiasi Controller; permette il caricamento di classi base come quelle per il routing e la sicurezza
- `post_controller_constructor`: viene invocato subito dopo che un Controller è istanziato, ma prima di qualsiasi metodo di classe in esso definito
- `post_controller`: viene chiamato subito dopo l'esecuzione completa di un Controller
- `display_override`: presiede all'override (riscrittura di un metodo ereditato) della funzione `_display()`, utilizzata per inviare gli output al browser web alla fine di un'esecuzione. È necessario referenziarsi al super oggetto con `$this->CI =& get_instance();` i dati finalizzati saranno disponibili attraverso la chiamata `$this->CI->output->get_output()`
- `cache_override`: consente di utilizzare una propria funzione al posto di quella di default `_display_cache()` in modo da creare un sistema di cache personalizzato
- `post_system`: viene invocata dopo l'invio dell'output al browser al termine di tutte le esecuzioni del sistema

## CHIAMATE MULTIPLE ALLO STESSO HOOK

Si può richiamare lo stesso Hook (mediante un punto di ancoraggio) con più di uno script, semplicemente rendendo l'array, con cui facciamo la dichiarazione, multidimensionale; si notino le parentesi quadre dopo le prime in cui viene definito il punto di ancoraggio:

```
$hook['pre_controller'][] = array(

    'class'      => 'MyClass',

    'function'   => 'Myfunction',

    'filename'   => 'Myclass.php',

    'filepath'   => 'hooks',

    'params'     => array('birra', 'vino', 'patatine')

);

$hook['pre_controller'][] = array(

    'class'      => 'MyOtherClass',

    'function'   => 'MyOtherfunction',

    'filename'   => 'Myotherclass.php',

    'filepath'   => 'hooks',

    'params'     => array('rosso', 'giallo', 'blu')

);
```

Le seconde parentesi quadre, come nella prima riga `$hook['pre_controller'][]` permettono di raggiungere lo scopo: definire un array multidimensionale per utilizzare il nostro Hook in più script.

## CARICAMENTO DELLE RISORSE

CodeIgniter utilizza un sistema di precaricamento che permette alle Librerie, Helper e Modelli di essere inizializzati automaticamente ogni volta che il sistema le richiede. Se invece si vuole che alcune risorse siano sempre disponibili globalmente, si dovrà configurare la funzionalità di CodeIgniter definita come “auto-load”. Le seguenti risorse possono essere caricate automaticamente:

```
\item le classi core che si trovano nelle directory relative alle \sys{
  libraries}
\item i file Helper che si trovano nelle directory relative agli \sys{helpers}
\item i file di configurazione che si trovano nella directory \sys{config}
\item i file del linguaggio che si trovano nella directory \sys{system}/
  language/}
\item i Modelli nella directory relative ai \sys{models}
```

Per apportare delle modifiche personalizzate è sufficiente agire sul file **auto-load.php** situato nel percorso **/application/config/**, semplicemente inserendo le risorse che si desiderano precaricare nell'array. I commenti presenti nel codice dovrebbero essere sufficientemente autoesplicativi.

Nota: si ricorda di non inserire l'estensione **.php** quando si specifica la risorsa.

Riassumendo quanto spiegato in questa sezione, possiamo dire che gli Hook permettono di intervenire nei meccanismi del funzionamento del framework senza modificarne il core, ovvero il cuore del sistema. Questo permette di evitare pericolose instabilità. Ogni sviluppatore può realizzare i propri Hook definendo delle apposite classi e i relativi metodi, oppure utilizzare quelli predefiniti di CodeIgniter: l'ordine con cui sono richiamati, definisce anche la sequenza di caricamento e di conseguenza la priorità. Infine si è visto che, anche se CodeIgniter cerca di caricare le risorse solo quando necessarie, è possibile "forzare" questo comportamento grazie all'auto-load manuale.

## 6.5 WEB PAGE CACHING

Uno degli obiettivi primari che CodeIgniter si prefissa è quello di massimizzare le prestazioni attraverso una serie di operazioni. Una di queste consente di memorizzare nella cache le pagine prodotte dal proprio progetto. Anche se CodeIgniter è abbastanza veloce, la quantità di informazioni dinamiche visualizzate nelle proprie pagine, unita al numero degli utenti che le producono, possono saturare le risorse del server come la memoria e la potenza di calcolo e quindi influenzare la velocità di caricamento delle pagine. Con il caching<sup>1</sup> delle pagine è possibile ottenere delle prestazioni migliori che si avvicinano a quella delle pagine web statiche.

### IL FUNZIONAMENTO

La cache è una zona di archiviazione temporanea: quando questa memorizza una pagina dinamica, conserva anche tutti i dati in essa contenuti. Il server tira un sospiro di sollievo: non deve interrogare il database, richiedere i valori dinamici e impegnare preziose risorse: gli basta caricare la cache dove è contenuta una “copia temporanea” della pagina. Un aspetto importante, è che la modalità caching può essere abilitata e configurata pagina per pagina, decidendo per esempio per quanto tempo una informazione debba essere conservata. Quando una pagina è caricata per la prima volta il file di cache sarà scritto nella cartella **/application/cache/** e le successive richieste non graveranno sul server, ma caricheranno semplicemente il file di cache che verrà inviato al browser del navigatore. Se il file di cache scade (si può impostare un limite temporale per conservarlo) il file di cache verrà cancellato e successivamente aggiornato prima di essere inviato al browser.

Nota: Il tag “Benchmark” non viene memorizzato nella cache.

### ABILITARE LA CACHE

Per abilitare il caching è necessario impostarlo in ogni metodo del proprio controller:

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

Dove il simbolo **n** indica i minuti per cui la pagina sarà mantenuta in cache prima che questa venga cancellata e quindi ricaricata. Il codice sopra riportato può inserito ovunque all’interno di una funzione. Non è influenzato dall’ordine in cui appare, così che lo si possa inserire dove è più congeniale. Una volta che il codice è correttamente inserito, le pagine verranno memorizzate nella cache.

Attenzione: A causa del modo con cui CodeIgniter memorizza i contenuti per l’output, il caching funziona solo se si stanno generando contenuti per la visualizzazione per il controller con una Vista.

<sup>1</sup> è una memoria temporanea che memorizza un insieme di dati che possano successivamente essere velocemente recuperati su richiesta.



Nota: Prima che i file di cache possono essere scritti è necessario impostare i permessi della cartella **/application/cache** in modo ci si possa scrivere.

## CANCELLARE LA CACHE

Se non si desidera più memorizzare nella cache un file è possibile rimuovere il codice caching e la pagina non sarà più aggiornata quando scade. La rimozione del codice non elimina immediatamente la cache: si dovrà attendere che essa scada normalmente. Se invece è importante rimuovere la cache immediatamente si dovrà eliminare manualmente il file dalla cartella della cache.

## 6.6 PROFILING

### PROFILARE LE APPLICAZIONI

Con questo termine si intende la misurazione delle prestazioni<sup>2</sup> delle query eseguite nel sistema; il risultato viene visualizzato in una variabile `$_POST` alla fine della proprie pagine. Questa informazione può essere utile durante lo sviluppo per fornire preziosi aiuti nella fase di debugging e nell'ottimizzazione del progetto.

<b>BENCHMARKS</b>	
Loading Time: Base Classes	0.0026
Controller Execution Time ( Dvd / Dvd Insert )	0.0057
Total Execution Time	0.0087
<b>GET DATA</b>	
No GET data exists	
<b>MEMORY USAGE</b>	
2,251,760 bytes	
<b>POST DATA</b>	
No POST data exists	
<b>URI STRING</b>	
dvd_insert	
<b>CLASS/METHOD</b>	
dvd/dvd_insert	
<b>DATABASE: <a href="#">archivio</a> <a href="#">QUERIES: 0 (Hide)</a></b>	
No queries were run	
<b>HTTP HEADERS (Show)</b>	

Fig. 17: Esempio di report benchmark

Per abilitare la profilazione è sufficiente inserire in qualsiasi punto del proprio Controller la seguente funzione:

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

Quando è abilitata verrà generato un report che sarà inserito nella parte inferiore delle pagina. Per disattivare la profilazione si dovrà invece specificare:

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

La classe non ha bisogno di essere inizializzata poiché è caricata automaticamente dalla **Classe Output** se la profilazione è abilitata.

<sup>2</sup> Definito anche benchmark

## PUNTI DI PROFILAZIONE

Per visualizzare il report con le prestazioni si devono definire dei punti di “profilazione” che vanno utilizzati come punti di riferimento per la misurazione delle prestazioni. Si deve utilizzare la sintassi seguente (dopo aver letto le informazioni sulla Classe Benchmark a pagina [143](#)):

Ogni sezione dei dati del Profiler può essere abilitata o disabilitata impostando una variabile di configurazione con il valore booleano TRUE o FALSE. Questo può essere fatto in due modi. Il primo metodo prevede la modifica dei valori di default che si trovano nel file di configurazione `application/config/profiler.php`.

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

L'altro metodo, più flessibile, si basa sullo sovrascrivere le impostazioni predefinite e i valori del file di configurazione chiamando i metodi `set_profiler_sections()` della **Classe Output**:

```
$section = array(  
    'config' => TRUE,  
    'queries' => TRUE  
);  
  
$this->output->set_profiler_sections($section);
```

La personalizzazione della profilazione potrà essere effettuata tramite il codice appena riportato, inserendolo nei propri Controller.

---

benchmarks	Tempo trascorso dei punti di benchmark e il tempo totale di esecuzione	TRUE
config	CodeIgniter variabili Config	TRUE
controller_info	La classe del Controller e il metodo richiesto	TRUE
get	Ogni dato GET passato nel request	TRUE
http_headers	Gli header HTTP per il request corrente	TRUE
memory_usage	Quantità di memoria consumata dalla richiesta corrente in byte	TRUE
post	Ogni dato POST passato nel request	TRUE
queries	Elenco di tutte le query di database eseguite, tra cui il tempo di esecuzione	TRUE
uri_string	l'URI della corrente richiesta	TRUE
query_toggle_count	Il numero di query dopo le quali, il blocco di query seguente verrà nascosto.	25

---

## 6.7 GESTIRE LE APPLICAZIONI

Per impostazione predefinita CodeIgniter gestisce solo un progetto alla volta definito nel percorso **/application/**. A volte però lo sviluppatore ha la necessità di gestire diversi progetti contemporaneamente con una sola installazione del framework; in questo caso si può optare per rinominare o spostare la cartella **/application/**.

Se si intende rinominare la cartella **/application/**, è necessario aprire il file **index.php** e modificare il seguente parametro, specificando il nome desiderato.

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

É possibile spostare la cartella in un differente percorso agendo sempre sul **index.php** e modificando il valore:

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

## PIÙ APPLICAZIONI ALLA VOLTA

Se si desidera lavorare su più progetti utilizzando una sola installazione di CodeIgniter è necessario organizzare i propri progetti in sotto directory, ognuna delle quali conterrà un singolo progetto. Per esempio, nel caso si vogliano creare due applicazioni denominate “foo” e “bar”, la struttura gerarchica delle proprie cartella sarà la seguente:

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

Per selezionare il progetto desiderato basterà, come spiegato precedentemente, agire sul file **filindex.php**. Ogni applicazione ha infatti bisogno del file **index.php** che richiama il progetto associato. Per esempio, per selezionare l’applicazione “foo”:

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

## 6.8 TEMPLATE ENGINE

### INTRODUZIONE

La rappresentazione del proprio progetto avviene attraverso le Viste, in cui viene riportato solitamente il codice [PHP](#). Questa però non è l'unica e la migliore strada. L'utilizzo di codice di markup per la presentazione, misto a quello demandato a definire la logica, può rendere l'interpretazione delle Viste, caotica e non facilmente gestibile da più sviluppatori contemporaneamente.

CodeIgniter permette di risolvere agevolmente questo problema grazie ad un template engine che rende le Viste più comprensibili senza pregiudicarne le funzionalità. Ma cosa si intende con esattamente con "template engine"? Con questo termine si intende un sistema che prende in ingresso il linguaggio [PHP](#) e restituisce un metalinguaggio più semplice da individuare, formattare e scrivere. In pratica viene utilizzata una nuova, più efficiente sintassi per descrivere le funzioni del [PHP](#) che elimina per esempio tutti gli stati "echo" e le parentesi dal codice: questo rende più semplice la cooperazione a tutte quelle persone che, contemporaneamente devono agire sugli stessi file.

Ci sono però degli inconvenienti nell'utilizzo di un nuovo linguaggio come per esempio il dover imparare un nuovo linguaggio per interpretare quello che precedentemente si svolgeva con il [PHP](#). Se si intende padroneggiare ogni aspetto di CodeIgniter per utilizzarlo costantemente nei propri progetti futuri, dedicare un po' di tempo alla comprensione del template engine, si rivelerà un investimento redditizio. Bisogna comunque anche tenere conto anche di un altro aspetto, sì purtroppo anch'esso negativo: l'uso di un metalinguaggio penalizza le prestazioni del framework poiché parte del calcolo viene speso per "interpretare" il codice da trasformare, in fase di esecuzione, in [PHP](#). Quale sia l'impatto di questa scelta sulla reattività del progetto è difficile da quantificare a priori, ma solitamente si tratta di un aspetto trascurabile. In tutti i casi la stima dell'efficienza del progetto con o senza template engine può essere facilmente calcolata con l'uso della "profilazione" (si veda [6.6 a pagina 96](#)).

### SINTASSI ALTERNATIVA

Il [PHP](#) permette di abbreviare molti comandi, soprattutto quelli condizionali e interattivi rendendoli più compatti. Non si tratta di una funzione esclusiva di CodeIgniter, ma solo di una sintassi alternativa, che non pregiudica le prestazioni del progetto. Per esempio, il comando "echo" per stampare una variabile su schermo ha una sintassi del tipo:

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

Dove "<?php" e "?>" racchiudono ogni codice [PHP](#), mentre \$variable è la variabile il cui contenuto sarà stampato su schermo. Ecco una sintassi alternativa per ottenere lo stesso risultato:

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

Forse è un po' più ermetica rispetto a quella che utilizzava l'istruzione `echo`, ma indiscutibilmente è più veloce da scrivere e anche più compatta. Anche le funzioni di controllo come `'if'`, `'for'`, `'foreach'`, `'while'` possono essere semplificate facilmente.

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

<h3>Ciao Alberto</h3>

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

<h3>Ciao Zoe</h3>

<?php else: ?>

<h3>Ciao sconosciuto</h3>

<?php endif; ?>
```

La sintassi alternativa che utilizza gli short tag non prevede parentesi ma utilizza un sistema che delimita il codice in "blocchi" ben definiti e facilmente distinguibili. Ogni funzione di controllo prevede un corrispettivo "tag di chiusura": `endif`, `endfor`, `endforeach`, e `endwhile`. Inoltre ognuna di queste strutture, eccetto l'ultima, prevedono nel tag l'uso del carattere `":"` (due punti). Vediamo come si può riscrivere il ciclo `foreach`:

```
<ul>

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

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

<?php endforeach; ?>

</ul>
```

Se la sintassi si qui definita non funziona come dovrebbe, la soluzione va ricercata in una impostazione presente nel file `filphp.ini` che disabilita gli "short tag". In questo caso, il problema è facilmente aggirabile modificando nel framework il file `/config/config.php` con il valore `TRUE`.

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



## 6.9 FUNZIONI COMUNI

CodeIgniter utilizza alcune funzioni per i suoi compiti che agiscono globalmente: queste sono sempre disponibili e non richiedono il preventivo caricamento di Librerie o Helper.

- `is_php('numero_versione')` è utilizzata nelle istruzioni condizionali per determinare la versione del [PHP](#) (restituisce un valore booleano)

```
// se la versione del PHP è la 5.3.0 viene eseguita l'istruzione  
  
if (is_php('5.3.0')) {  
  
    $str = quoted_printable_encode($str);  
  
}
```

- `is_really_writable('path/to/file')` questa funzione restituisce TRUE nei server Windows quando non si hanno i permessi di scrittura sul file. Invece viene restituito FALSE solo se l'attributo è di sola lettura. Questa funzione è utile perché permette di capire se è *consentito* scrivere su di un file, senza però effettivamente scriverci. Si ricorda che l'operazione di scrittura su un qualsiasi file può causare la perdita delle informazioni ivi contenute. Generalmente si consiglia l'utilizzo della funzione solo su piattaforme dove le informazioni sui file non possono essere determinate a priori.

```
if (is_really_writable('file.txt')) {  
  
    echo "Posso scrivere sul file, se lo desidero";  
  
}  
  
else {  
  
    echo "Non si può scrivere sul file";  
  
}
```

- `config_item('item_key')` la Libreria Config è il metodo consigliato per accedere alle informazioni sulla configurazione. Tuttavia questa funzione può essere utilizzata per recuperare singoli parametri.
- le tre funzioni seguenti sono legate alla gestione degli errori:
  - `show_error('message')`
  - `show_404('page')`

– `log_message('level', 'message')`.

- `set_status_header(code, 'text')`; permette di settare manualmente l'header dello stato del server. Per esempio:

```
set_status_header(401);  
  
// Sets the header as: Unauthorized
```

- `remove_invisible_characters($str)` questa funzione previene l'inserimento di caratteri nulli tra i caratteri ASCII, come per esempio `Java\0script`.
- `html_escape($mixed)` fornisce una scorciatoia alla funzione `htmlspecialchars()` utilizzata per l'escape delle stringhe. In questo modo si previene il fatto che codice esterno possa essere eseguito sul server e scatenare attacchi di tipo [XSS](#).

## FUNZIONI PER LA GESTIONE DEGLI ERRORI

In fase di debug risultano preziose le funzioni messe a disposizione da CodeIgniter per la gestione degli errori e di tutte le problematiche fastidiose che incorrono nella fase di sviluppo. CodeIgniter consente di creare dei report sugli errori che si verificano nelle applicazioni grazie alle funzioni descritte di seguito. Inoltre, possiede una classe che consente di salvare come messaggi di testo tutti gli errori e gli avvisi di debug.

Per impostazione predefinita, CodeIgniter visualizza tutti gli errori generati dal [PHP](#). Questo fatto è utile in fase di sviluppo, ma si potrebbe voler modificare questo comportamento una volta che ci si trova nella fase di produzione. Per cambiare questo aspetto si può agire sulla funzione `error_reporting()` che si trova nella parte superiore del file `index.php` principale. Disabilitare la segnalazione degli errori non impedirà comunque che il file di log venga generato.

A differenza di molti sistemi in CodeIgniter, le funzioni di errore sono semplici interfacce procedurali che sono disponibili globalmente in tutta l'applicazione. Questo approccio consente di avere un completo log riguardante i messaggi di errore senza doversi preoccupare di effettuare dei settaggi particolari sulle classi o i metodi.

Il framework mette a disposizione le seguenti funzioni per la gestione degli errori:

- `show_error('message' [, int $status_code= 500])`: visualizza i messaggi di errore al seguente percorso: `application/errors/error_general.php`. Il parametro `$status_code` è facoltativo: se utilizzato, determina quale codice di stato HTTP dovrebbe essere inviato con l'errore.
- `show_404('page' [, 'log_error'])`: questa funzione visualizza i messaggi di errore al seguente percorso: `application/errors/error_404.php`. Essa si aspetta che la stringa passata corrisponda ad una pagina inesistente (di cui non si trova il percorso). CodeIgniter visualizza sempre, automaticamente l'errore

tramite questa funzione predefinita; è comunque possibile impostare il secondo parametro su FALSE in modo da ignorare questo tipo di errori (skip log).

- `log_message('livello', 'messaggio')`: questa funzione consente di scrivere messaggi personalizzati sui propri file di log. È necessario fornire uno dei tre livelli nel primo parametro, che indica il tipo di messaggio o etichetta se si preferisce: debug, error, info. Il secondo parametro “messaggio” specifica un testo personalizzato che si desidera visualizzare. Per esempio:

```
if ($alcune_variabili == "")
{
    log_message('error', 'Alcune variabili non contengono un valore');
}
else
{
    log_message('debug', 'Alcune variabili non sono correttamente
    configurate');
}

log_message('info', 'Lo scopo di una variabile è di fornire un certo
valore');
```

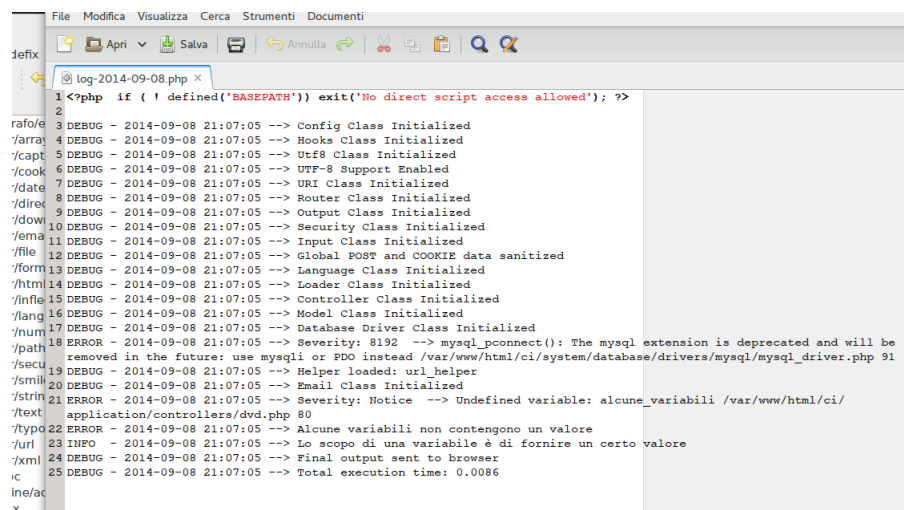


Fig. 18: Il file log personalizzato

Ci sono tre tipi di messaggio:

1. messaggi di errore: riguardano gli avvisi così come sono prodotti dagli errori dovuti al PHP o alle errate impostazione dell'utente
2. messaggi di debug: assistono l'utente nell'individuazione dei problemi indicando per esempio, se una classe è stata inzializzata

3. messaggi informativi: questi sono i messaggi con priorità più bassa. Sem-  
plicemente forniscono informazioni riguardanti qualche processo. Co-  
deIgniter non genera come impostazione predefinita alcun messaggio di  
questo tipo, ma l'utente può utilizzarli per raccogliere utili informazioni

Nota: affinché il file di log possa essere generato, la cartella **logs** deve avere i  
permessi di scrittura. Inoltre, è necessario impostare la soglia per l'accesso di  
`$config['log_threshold']` in **application/config/config.php**. Si potrebbe, per  
esempio, voler produrre solo i messaggi di errore che devono essere registrati  
e non gli altri due tipi. Se si imposta la registrazione (logging) a zero, questo  
verrà disabilitato. Per completezza elenchiamo tutti i parametri:

- 0 = Disabilita la registrazione dei messaggi di errore
- 1 = Messaggi di errore (compresi quelli PHP)
- 2 = Messaggi di debug
- 3 = Messaggi di informazione
- 4 = Tutti i messaggi

## 6.10 FONDAMENTI DI SICUREZZA

La sicurezza è uno degli argomenti più importanti e spesso sottovalutati nello sviluppo di un progetto. Per fortuna CodeIgniter mette a disposizione una ricca serie di funzioni che aiutano a curare questo aspetto.

- Sicurezza negli URI: per minimizzare gli attacchi dovuti a codice malevole inviato al sito web, CodeIgniter è molto restrittivo sul tipo di caratteri che possono essere inclusi negli URI: questi di seguito sono gli unici caratteri permessi:
  - testo alfanumerico
  - tilde (~)
  - punto (.)
  - due punti (:)
  - underscore (\_)
  - dash (-)
- `Register_globals` durante l’inizializzazione del sistema tutte le variabili globali non sono impostate, eccetto quelle contenute negli array `$_GET`, `$_POST` e `$_COOKIE`. Le routine non impostate sono le stesse di `register_globals = off`
- `error_reporting` nell’ambiente di produzione è consigliato disabilitare la visualizzazione di qualsiasi tipo di errore o avviso del [PHP](#), impostando su o il flag interno `error_reporting`. È importante limitare la visualizzazione di informazioni sensibili, poiché queste potrebbero essere sfruttate per minare la sicurezza del progetto. Questo aspetto può essere gestito tramite CodeIgniter modificando il file `filindex.php` nella sezione “application enviroment” e inserendo uno dei tre valori:  
**DEVELOPMENT** Ambiente di sviluppo. Tutti gli errori e avvisi sono abilitati  
**TESTING** Ambiente testing  
**PRODUCTION** Ambiente di produzione. Tutti gli errori e avvisi sono disabilitati
- `magic_quotes_runtime` la direttiva `magic_quotes_runtime` durante l’inizializzazione del sistema è impostata su “off” in modo da non dover rimuovere il carattere “slash” quando si recuperano i dati dalla base di dati.

## BEST PRACTICE

Una raccolta di buone abitudine condivise dagli sviluppatori prende il nome di “best practice”. Prima di accettare qualsiasi tipo di dati nella propria applicazione, sia che si tratti di dati di tipo POST presi da un form, cookie, URI, XML-RPC, o anche array SERVER, è sempre una buona prassi rispettare nell’ordine queste tre regole:

1. Non fidarsi. È buon norma filtrare i dati come se fossero sempre pericolosi

2. Convalidare i dati per assicurarsi che essi rispettino i parametri originariamente impostati come il tipo corretto, la lunghezza, le dimensioni, ecc (a volte questo punto può sostituire quello precedente)
3. Effettuare sempre l'escape dei dati prima di inviarli al database. Con l'escape dei dati si intende la sostituzione dei caratteri speciali per evitare che il nostro database possa eseguire codice malevolo, invece di semplici dati

CodeIgniter fornisce le seguenti funzioni per assistere lo sviluppatore nella sicurezza del proprio software:

**FILTRO XSS** CodeIgniter è dotato di un filtro Cross Site Scripting che esamina le tecniche comunemente utilizzate per inserire codice Javascript dannoso o simili che tentano di dirottare i cookie

**VALIDAZIONE DEI DATI** Il framework fornisce diversi strumenti che analizzano la conformità dei dati inseriti agli standard attesi

**ESCAPE DEI DATI** Prima di inviare alla base di dati i valori ottenuti dall'esterno, per esempio tramite un form, è sempre una buona prassi effettuare l'escape dei caratteri

## GESTIONE DEGLI ERRORI

Il framework CodeIgniter è ricco di funzioni native per la gestione degli errori nella fase di sviluppo. Gran parte del tempo nella realizzazione di prodotti complessi viene infatti speso nella fase di debug e poter contare su strumenti efficienti per trovare, visualizzare e gestire ogni possibile "inconveniente" è uno dei punti di successo di un buon framework.

CodeIgniter mette a disposizione una classe per il logging degli errori che consente il salvataggio delle notifiche di errore e i messaggi prodotti dal debugging all'interno di un comodo file di testo. Per impostazione predefinita, gli errori prodotti dalle applicazioni vengono sempre mostrati: questa scelta può essere modificata passando il valore "0" (zero) alla funzione `error_reporting()` posizionata all'inizio del file **index.php**. In tutti i casi, disabilitare la visualizzazione degli errori non impedirà la scrittura degli stessi all'interno dei file di log. Le funzioni disponibili per la gestione degli errori sono le seguenti:

- `show_error('messaggio' [, int $status_code= 500 ])` si tratta di una funzione che visualizza gli errori attraverso un template che si trova al percorso: `application/errors/error_general.php`

Il parametro `$status_code` non è obbligatorio, ma può essere utilizzato per determinare lo status [HTTP](#) che verrà associato all'errore inviato al browser.

`show_404('pagina')` è una funzione che presiede alla visualizzazione del messaggio di errore corrispondente allo status 404 ("Not Found"); la tipologia di notifica è stabilita attraverso un template che si trova nel percorso: `application/errors/error_404.php`. La funzione produce la notifica di un errore nel caso in cui non sia visualizzabile la pagina che le viene passata come

argomento; il framework è comunque impostato per restituire un messaggio di errore 404 nel caso in cui il tentativo di caricamento di un Controller non abbia successo.

- `log_message('livello', 'messaggio')` è una funzione che permette di scrivere dei messaggi all'interno dei log degli errori; la voce `livello` prevista come parametro, serve per indicare il tipo di notifica che si desidera scrivere e, a questo scopo, sono previsti tre livelli: `debug`, `error` e `info`. Il seguente esempio chiarirà le differenze tra i tre livelli previsti anche attraverso i commenti:

```
// Livelli per la notifica degli errori
$piippo = 0;
if ($piippo != 0)
{
    // il livello error indica l'errore corrente
    log_message('error', 'Valore inaspettato per la variabile.');
```

```
}
else
{
    // il livello debug espone notifiche utili per lo sviluppatore
    log_message('debug', 'La variabile deve avere valore 0');
```

```
}

// il livello info fornisce informazioni su un processo
log_message('info', 'Errore relativo ad una variabile.');
```

CodeIgniter è stato sviluppato con un occhio attento alla sicurezza. Per esempio si dimostra altamente restrittivo nei confronti dei caratteri utilizzabili per la generazione degli [URI](#); in questo caso infatti è permesso utilizzare nelle stringhe soltanto i caratteri: tilde, punto, due punti, underscore, dash (trattino) nonché, naturalmente, i comuni caratteri alfanumerici.

Per quanto riguarda i dati inviati per metodo, è importante sottolineare che **GET** viene automaticamente disabilitato quando si utilizzano le [URL](#) segmentate al posto di quelle basate sulle query string; ciò avviene perché CodeIgniter impedisce, già in fase di inizializzazione, la generazione dell'array globale `_GET`; ciò non avviene per `$_POST` e `$_COOKIE`. In pratica si può dire che il trattamento dei parametri di input da parte del framework, simuli del tutto una configurazione di [PHP](#) con direttiva `register_globals` impostata su `OFF`.

CodeIgniter è fornito di un filtro nativo contro gli attacchi di tipo [XSS](#) che impedisce in numerosi casi che nelle pagine di un sito venga eseguito del codice Javascript malevolo, ed inoltre blocca i tentativi di hijacking dei cookie. Per utilizzare questo filtro è sufficiente una semplice chiamata:

```
// Utilizzo del filtro contro gli XSS
\verb|$data = $this->input->xss_clean($data);|
```

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

## INTRODUZIONE AD .HTACCESS

Chi ha avuto modo di amministrare un sito web, prima o poi ha avuto prima a che fare con il file **.htaccess**. Si tratta di un comune file dalla grande importanza perché permette di personalizzare le impostazioni globali di accesso del server. Esso è accessibile nella directory root del progetto web, ma può trovarsi anche in sottodirectory di cui si vuole personalizzare l'accesso.

Le sue funzioni sono innumerevoli e di cruciale importanza nell'ambito della sicurezza. È infatti possibile bloccare l'accesso al sito da parte di intere nazioni, numeri IP, definire le pagine di errore o semplicemente gestire l'accesso alle cartelle o ai file secondo determinati criteri. La configurazione del file **.htaccess** permette in pratica di gestire le richieste in entrata con precisione e con precedenza su qualsiasi altra impostazione definita da altri software, tra cui il framework CodeIgniter.

## CONFIGURARE I PERMESSI IN ENTRATA

In CodeIgniter, il file **.htaccess** si trova nella directory **application/** e **system/** e non può essere individuato tramite il comando da terminale **ls** (il punto che precede il nome file ha proprio la funzione di nascondere un file/cartella). Si scriva pertanto in una delle due cartelle sopra menzionate il comando bash seguente:

```
ls -la
```

Il quale possiede due argomenti dopo il simbolo del - (trattino):

- “a” permette di visualizzare tutti i file, anche quelli nascosti
- “l” semplicemente visualizza la lista in maniera più ordinata

Ora che si è soddisfatta questa curiosità, si possono anche modificare i parametri interni (in realtà non è necessario vedere il file per aprirlo e quindi modificarlo). Si apra il file con uno dei comandi seguenti a seconda dell'editor di testo preferito, prestando attenzione ad avere i permessi di superuser:

- vi **.htaccess**
- gedit **.htaccess**

CodeIgniter mostrerà la stringa “Deny from all” che significa accesso negato per chiunque in entrata.



## BLOCCARE GLI INDIRIZZI

É possibile bloccare un solo indirizzo IP o anche un intervallo definito. Il seguente codice commentato è autoesplicativo:

```
order allow,deny
deny from 1.2.3.4          # Blocca un singolo IP
deny from 1.2.3.          # Blocca tutti gli IP da 1.2.3.0 a 1.2.3.255
allow from all
```

## PAGINA PRINCIPALE

Solitamente la prima pagina che viene caricata è quella con nome “index” ed estensione “.html”. É possibile modificare questa priorità, indicando al sistema quale pagina debba avere la priorità:

```
DirectoryIndex NuovoFileIndex.html # Nuova pagina index
```

Oppure si possono indicare una sequenza di file da caricare, nel caso uno di essi non esistesse:

```
DirectoryIndex index1.html index2.html index3.html
```

## GESTIRE IL WWW DEL DOMINIO

É possibile caricare il proprio sito rendendo obbligatorio o meno il prefisso **www** utilizzando delle espressioni regolari.

```
# Da sito privo di www a sito con www
Options +FollowSymlinks
RewriteEngine on
RewriteCond %{HTTP_HOST} ^miosito.it$
RewriteRule ^(.*)$ http://www.miosito.it/$1 [R=301,L]
```

```
# Da sito con www a sito privo di www
Options +FollowSymlinks
RewriteEngine on
RewriteCond %{HTTP_HOST} ^www.miosito.it$
RewriteRule ^(.*)$ http://miosito.it/$1 [R=301,L]
```

## REINDIRIZZARE UN URL

Se il proprio sito è stato aggiornato per un restyling oppure è stata effettuata una migrazione di dominio, ci si può ritrovare nella spiacevole situazione in cui numerose richieste [HTTP](#) generano un errore 404. Questo inconveniente può essere gestito attraverso il codice di stato “Redirect 301 Moved Permanently”. Se gli errori di tipo 404 sono pochi, allora è consigliabile adottare una soluzione come quella mostrata nell’esempio:

```
Redirect 301 /vecchia-pagina.htm http://www.miosito.it/nuova-pagina.htm
```

Una soluzione migliore, anche se leggermente più complicata, prevede l’utilizzo del `mod_rewrite` di Apache: attraverso le espressioni regolari è possibile creare o modificare gli indirizzi di un progetto per adattarli dinamicamente alle proprie esigenze. Il seguente esempio definisce una serie di regole che intercettano tutte le pagine con estensione “.html” ridefinendole con estensione “.php”. Si tratta di una esigenza comune per chi trasforma un sito statico in uno dinamico.

```
<ifModule mod_rewrite.c>
RewriteRule ^(.*)\.Html$ http://www.miosito.com/$1.php [R=301,NC]
</ifModule>
```

Esempio di reindirizzamento di tutte quelle pagine che hanno all’interno dell’[URL](#) una variabile uguale del tipo: `http://www.miosito.it?pagina=4`

```
<ifModule mod_rewrite.c>
RewriteCond %{QUERY_STRING} pagina=(.*)$
RewriteRule . http://www.miosito.it/nuova-pagina? [R=301,L]
</ifModule>
```

Un altro esempio è fornito dal sito che presenta un indirizzo del tipo:

```
http://www.miosito.it/eventi.php?categoria=musica&id=4
```

Questo può trasformato in un [URL](#) seo-friendly del tipo:

```
http://www.miosito.it/categoria/musica/4
```

Utilizzando le seguenti espressioni regolari:

```
<ifModule mod_rewrite.c>
RewriteRule ^categoria/([^/]*)/([^/]*)$ /eventi.php?categoria=$1&id=$2 [L]
</ifModule>
```

## RIEPILOGO

Mai commettere l'imprudenza di sottovalutare la sicurezza in un progetto. Esistono molte best practice per prevenire gli attacchi esterni, ma nonostante questo solo lo stolto si sente al sicuro da qualsiasi malintenzionato. Come le cronache ci hanno ormai abituato, basta un piccolo bug in un modulo software o poche righe di codice mal scritte per rendere accessibili a chiunque i propri dati sensibili. Adottando tutte le possibili precauzioni e gli strumenti che CodeIgniter mette a disposizione si possono contrastare la maggior parte dei subdoli attacchi esterni: l'importante è mantenere un livello di guardia sempre alto e costante!



## 7.1 STILE E SINTASSI DEL PHP

Qui di seguito verranno descritte le regole generali nello sviluppo del codice con CodeIgniter.

### FORMATO DEI FILE

I file dovrebbero essere salvati sempre con l'encoding Unicode UTF-8. Il formato BOM, molto comune negli editor di default di Windows, non dovrebbe mai essere utilizzato: questo formato inserisce dei caratteri nel file che può apportare degli effetti negativi sul codice [PHP](#). È importante inoltre scegliere come terminatore di riga<sup>1</sup> l'encoding Unix LF.

Qui di seguito vengono descritti i parametri per configurare due editor testuali molto utilizzati:

#### Textmate

- Aprite le Preferenze dell'Applicazione
- Selezionate Avanzata e quindi Salvataggio
- Selezionate UTF-8 (raccomandato) in "File Encoding"
- Mentre in "Line Endings", si utilizzi la voce Seleziona LF (raccomandato)

#### BBEdit

Aprite le Preferenze dell'Applicazione. Si selezioni "Text Encodings" sulla sinistra. In "Default text encoding for new documents" si selezioni "Unicode (UTF-8, no BOM)". Si selezioni "Text Files" sulla sinistra. In "Default line breaks", l'opzione corretta da utilizzare è "Mac OS X and Unix (LF)".

### TAG DI CHIUSURA PHP

Nel [PHP](#) il tag di chiusura utilizzato è `?>` anche se spesso non viene riportato, poiché per il parser del linguaggio questo tag è opzionale. Se quindi, contrariamente ad altri linguaggi non lo vedete a fine pagina, non ci si preoccupi: per gli sviluppatori

---

<sup>1</sup> In inglese line ending

professionisti è normale trascurarlo. Infatti, il problema che può verificarsi è che vengano introdotti dopo il tag di chiusura, eventuali spazi vuoti che possono causare errori [PHP](#), pagine vuote, o problemi nelle applicazioni [FTP](#). Per questo motivo è consigliato omettere sempre il tag di chiusura e utilizzare invece un blocco di commento per segnare la fine del file e la sua posizione rispetto alla radice dell'applicazione. Ciò consente di marcare il file come completo.

```
// esempio non corretto
<?php

echo "Questo è il mio codice non corretto!";

?>

// esempio corretto
<?php

echo "Questo è il mio codice corretto!";

// fine del file myfile.php
// percorso ./system/modules/mymodule/myfile.php
```

## NOMI DELLE CLASSI E DEI METODI

I nomi delle classi devono incominciare sempre con la prima lettera maiuscola, mentre le parole all'interno del file devono essere separate tramite il carattere underscore, senza utilizzare la convenzione polacca<sup>2</sup>

I nomi dei metodi invece devono essere autoesplicativi, chiari e preferibilmente includere un verbo; inoltre devono essere scritti interamente in minuscolo. Cercate di evitare nomi troppo lunghi e dettagliati.

```
// Non corretto
class superclass
class SuperClass

// Corretto
class Super_class
```

<sup>2</sup> Conosciuta anche come CamelCase, è una convenzione con cui i nomi composti da più parole vengono scritti senza spazi, con l'iniziale maiuscola come per esempio "nomeDelFile".

```
class Super_class {  
  
    function __construct()  
    {  
  
    }  
}
```

Qui invece si riportano degli esempi corretti, e meno sulla denominazione dei metodi:

```
// Non corretto  
function fileproperties() // non descrittivo, manca l'underscore  
function fileProperties() // non descrittivo e usa la convenzione CamelCase  
function getfileproperties() // meglio, ma continua a mancare l'underscore  
function getFileProperties() // utilizza CamelCase  
function get_the_file_properties_from_the_file() // prolisso  
  
// Corretto  
function get_file_properties() // descrittivo, underscore e lettere minuscole
```

## COMMENTI

In generale il codice dovrebbe essere commentato con cura. Questo perché aiuta i programmatori meno esperti che devono comprendere il nostro lavoro, e noi stessi quando si deve ritornare sul codice a distanza di mesi. Non c'è un formato richiesto per i commenti ma si raccomanda quanto segue.

I commenti in stile DocBlock<sup>3</sup> precedono la dichiarazione delle classe e dei metodi che possono essere raccolti dagli Integrated Development Environment ([IDE](#)).

```
/**  
 * Super Class  
 *  
 * @package Package Name  
 * @subpackage Subpackage  
 * @category Category  
 * @author Author Name  
 * @link http://example.com  
 */  
class Super_class {
```

<sup>3</sup> <http://phpdoc.org/docs/latest/guides/docblocks.html>

```
/**
 * Encodes string for use in XML
 *
 * @access public
 * @param string
 * @return string
 */
function xml_encode($str)
```

Utilizzare i commenti a singola linea per brevi spiegazioni, invece lasciare una riga vuota tra grandi blocchi di commento e il codice.

```
// break up the string by newlines
$parts = explode("\n", $str);

// A longer comment that needs to give greater detail on what is
// occurring and why can use multiple single-line comments. Try to
// keep the width reasonable, around 70 characters is the easiest to
// read. Don't hesitate to link to permanent external resources
// that may provide greater detail:
//
// http://example.com/information_about_something/in_particular/

$parts = $this->foo($parts);
```

## LE COSTANTI

Le costanti non differiscono dalla regole di stile definite per le variabili, tranne per il fatto che queste vanno indicate completamente in maiuscolo.

```
// Non corretto
myConstant // manca l'underscore separatore e non è in maiuscolo
N // sbagliato riportare le costanti con un singolo carattere
S_C_VER // non descrittivo
$str = str_replace('{foo}', 'bar', $str); // usare le costanti LD e RD

// Corretto
MY_CONSTANT
NEWLINE
SUPER_CLASS_VERSION
$str = str_replace(LD.'foo'.RD, 'bar', $str);
```



## TRUE, FALSE, E NULL

Sono parole chiave che si dovrebbero riportate sempre in maiuscolo.

```
// Non corretto
if ($foo == true)
$bar = false;
function foo($bar = null)

// Corretto
if ($foo == TRUE)
$bar = FALSE;
function foo($bar = NULL)
```

## OPERATORI LOGICI

L'utilizzo di `||` (doppio pipe) è sconsigliato sia perché rende il codice meno intuitivo, sia perché si può confondere con il numero 11: meglio usare `OR`. Invece `&&` è preferibile a `AND`, ma ambedue sono corretti. Nell'utilizzo dell'operatore di negazione `!` è consigliato farlo precedere e seguire da uno spazio.

```
// Non corretto
if ($foo || $bar)
if ($foo AND $bar) // è corretto ma si raccomanda di utilizzare &&
if (!$foo)
if (! is_array($foo))

// Corretto
if ($foo OR $bar)
if ($foo && $bar) // raccomandato
if ( ! $foo)
if ( ! is_array($foo))
```

## TYPECASTING

Alcune funzioni [PHP](#) restituiscono `FALSE` in caso di errore, ma possono anche avere un valore di ritorno come `""` oppure `0`, che sono valutati come `FALSE` in molti confronti con operatori logici. Per evitare incongruenze è necessario essere espliciti quando si utilizzano valori di ritorno<sup>4</sup> in modo che il loro successivo utilizzo all'interno di operatori come `AND` oppure `OR`, per esempio, non fornisca ambiguità e quindi un risultato non corretto.

---

<sup>4</sup> Sono dati resi dall'istruzione `return`.

È consigliato utilizzare delle verifiche rigorose sul tipo di dati delle proprie variabili, utilizzando `===` e `!==` quando necessario. La differenza rispetto ai consueti operatori di confronto è esplicitata nella tabella 7.1

Operatore	Descrizione
<code>==</code>	uguale
<code>===</code>	identico (cioè con uguale valore e medesimo tipo)
<code>!=</code>	diverso
<code>!==</code>	diverso per valore, ma del medesimo tipo
<code>&gt;</code>	maggiore
<code>&gt;=</code>	maggiore o uguale
<code>&lt;</code>	minore
<code>&lt;=</code>	minore o uguale

```
// Non corretto
// Se 'foo' è all'inizio della stringa, strpos à restituit 0 e
// l'istruzione condizionale à restituir TRUE
if (strpos($str, 'foo') == FALSE)

// Corretto
if (strpos($str, 'foo') === FALSE)
```

```
// Non corretto
function build_string($str = "")
{
    if ($str == "") // Cosa succede se viene passato FALSE o il valore intero 0?
    {

    }
}

// Corretto
function build_string($str = "")
{
    if ($str === "")
    {

    }
}
```

### Conversione in boolean nel PHP

Quando si converte un valore verso un tipo booleano, i seguenti valori sono considerati FALSE (tutti gli altri TRUE):

- FALSE ovviamente
- l'intero 0 (zero)
- il float 0.0 (zero)
- la stringa vuota e la stringa "0"
- un array con zero elementi
- un oggetto con nessuna variabile
- il valore speciale NULL
- un oggetto SimpleXML creato da tag vuoti

Nota: -1 è considerato TRUE così come altri valori numerici diversi da zero (sia positivi che negativi).

```
<?php
var_dump((bool) "");           // bool(false)
var_dump((bool) 1);            // bool(true)
var_dump((bool) -2);           // bool(true)
var_dump((bool) "foo");        // bool(true)
var_dump((bool) 2.3e5);         // bool(true)
var_dump((bool) array(12));    // bool(true)
var_dump((bool) array());      // bool(false)
var_dump((bool) "false");      // bool(true)
?>
```

Si tratta di un argomento di non secondaria importanza, e per questo si rimanda alla documentazione ufficiale: <http://us3.php.net/manual/en/language.types.type-juggling.php#language.types.typecasting> che è molto chiara sull'argomento. Il typecasting presenta molte sfaccettature che possono portare a risultati non desiderati.

```
$str = (string) $str; // cast $str come una stringa
```

## CODICE DI DEBUG

Nessuna istruzione utilizzata per il debug può essere lasciata nel codice senza essere opportunamente commentata e quindi resa inefficace. Istruzioni come per esempio `var_dump()`, `print_r()`, `die()`, e `exit()` devono essere sempre commentate alla fine del loro utilizzo o direttamente rimosse.

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

## SPAZI VUOTI NEI FILE

Nessun spazio dovrebbe precedere il tag di apertura del [PHP](#), così come non dovrebbero essercene dopo il tag di chiusura (che in questo linguaggio si può omettere). L'output infatti viene memorizzato (lo spazio è un dato) e CodeIgniter può interpretarlo erroneamente proprio a causa di queste informazioni fuorvianti.

## COMPATIBILITÀ

Tutto il codice del proprio progetto deve essere scritto tenendo come punto di riferimento la versione 5.1 o superiore del [PHP](#), a meno che questo non sia espressamente indicato nei commenti esplicativi del codice. È anche consigliato non utilizzare librerie non-standard<sup>5</sup> a meno che non sia fornita una alternativa quando esse non sono disponibili.

## NOMI COMUNI DEI FILE E DELLE CLASSI

Quando la classe o il nome del file è una parola di uso comune, o potrebbe molto probabilmente essere già stata utilizzata in un altro script [PHP](#), il rischio che ci siano più variabili con lo stesso nome è molto alta. Fornire un prefisso univoco può aiutare a prevenire questo genere di problemi definiti anche collisioni. Siccome gli utenti finali potrebbero utilizzare altri add-on o script [PHP](#) di terze parti ed introdurre a loro volta delle collisioni, è consigliabile scegliere un prefisso che sia univoco come il proprio identificativo come sviluppatore o società.

```
// Non corretto
class Email    pi.email.php
class Xml      ext.xml.php
class Import    mod.import.php

// Corretto
class Pre_email    pi.pre_email.php
class Pre_xml      ext.pre_xml.php
class Pre_import    mod.pre_import.php
```

## NOME DELLE TABELLE DI UN DATABASE

Anche nella definizione dello schema di una base di dati può presentarsi il problema delle collisioni. Per questo motivo ogni nome di tabella dovrebbe contenere

<sup>5</sup> Le librerie standard sono quelle previste nell'installazione base del [PHP](#).

il prefisso `exp_` seguito da un identificativo univoco legato alla propria identità o società

```
// Non corretto
email_addresses // mancano i due prefissi
pre_email_addresses // manca il prefisso exp_
exp_email_addresses // manca il prefisso identificativo

// Corretto
exp_pre_email_addresses
```

Nota: É importante essere consapevoli che MySQL ha un limite di 64 caratteri per i nomi di tabella. Questo non dovrebbe essere un problema in quanto i nomi che supererebbe questo limite sarebbero irragionevolmente lunghi. Ad esempio, il seguente nome della tabella supera i 64 caratteri.

```
exp_pre_email_addresses_of_registered_users_in_seattle_washington
```

## UN FILE PER CLASSE

É consigliato utilizzare file separati per ciascuna classe utilizzata nel proprio progetto, a meno che le classi non siano strettamente correlate. In CodeIgniter per esempio, il file di classe Database contiene a sua volta più classi, ovvero la classe DB, la classe DB\_Cache, e il plugin Magpie (a sua volta Magpie contiene le classi Magpie e Snoopy).

## SPAZI NEL CODICE

Per lasciare degli spazi nel codice è sempre consigliato utilizzare il tasto di tabulazione (tab). Questa scelta è sempre preferibile al singolo spazio, ottenuto con la barra spaziatrice, perchè permette di formattare meglio il codice e di rendere evidenti gli spazi al primo colpo d'occhio. L'uso del tab inoltre permette di avere un codice più compatto: lo spazio in byte per memorizzare un tab è uguale a quattro singoli caratteri spazio.

## INTERRUZIONE DI LINEA

I file devono essere salvati utilizzando l'interruzione di linea di Unix<sup>6</sup> che ricordiamo è un carattere speciale usato per gestire la fine di una riga di testo (e quindi non un vero e proprio carattere visibile sullo schermo). Il nome deriva dal fatto che il carattere successivo dopo di esso viene visualizzato su una nuova riga. Solitamente sono gli editor predefiniti di Windows o quelli più elementari (sempre per Windows) a non supportare il sistema LF.

## INDENTAZIONE DEL CODICE

Sull'indentazione delle righe del proprio codice di programmazione esistono molte correnti di pensiero, tutte più o meno valide. Ognuno di noi ha poi un proprio pensiero riguardo alla formattazione del codice: chi preferisce lasciare molti spazi, chi invece ama un testo compatto. In tutti i casi, il codice deve essere scritto non solo per essere efficiente, ma anche facile da leggere per chiunque.

### Definizione 8: Indentazione

<b>Indentazione</b>	Rappresenta le regole che disciplinano la formattazione dei blocchi di codice: lo scopo prefissato è rendere chiaro e organizzato il codice sorgente allo sviluppatore che lo ha prodotto e a chi lo consulta per la prima volta.
---------------------	---

Tra gli stili utilizzati, prenderemo in considerazione quello conosciuto come lo Stile Allman, che prende il nome da Eric Allman, noto programmatore americano. Questo tipo di indentazione è anche conosciuta come stile BSD poiché è stato adottato dallo stesso Allman per la scrittura di molti programmi per BSD Unix.

```
while (a == b)
{
    doSomething();
    doSomething2();
}
finalize();
```

Come si può notare dall'esempio, le parentesi del ciclo while sono inserite in una propria linea e sono formattate allo stesso livello dell'indentazione del ciclo. Questo stile migliora la leggibilità e rende più facile trovare parentesi graffe corrispondenti ad un'istruzione.

<sup>6</sup> Definito anche come ritorno a capo, o nuova riga o a capo (in inglese newline, line break o carattere end-of-line / EOL)

```
// Non corretto
function foo($bar) {
    // ...
}

foreach ($arr as $key => $val) {
    // ...
}

if ($foo == $bar) {
    // ...
} else {
    // ...
}

for ($i = 0; $i < 10; $i++)
{
    for ($j = 0; $j < 10; $j++)
    {
        // ...
    }
}

// Corretto
function foo($bar)
{
    // ...
}

foreach ($arr as $key => $val)
{
    // ...
}

if ($foo == $bar)
{
    // ...
}
else
{
    // ...
}

for ($i = 0; $i < 10; $i++)
{
    for ($j = 0; $j < 10; $j++)
    {
        // ...
    }
}
```

## SPAZI TRA PARENTESI

in generale non si dovrebbero utilizzare degli spazi tra le parentesi tonde e quadre. Una eccezione è ammissibile solo in presenza di strutture di controllo [PHP](#) (dichiarazioni, do-while, elseif, for, foreach, if, switch, while), in modo da distinguerle più facilmente dalle funzioni.

```
// Non corretto
$arr[ $foo ] = 'foo';

// Corretto
$arr[$foo] = 'foo'; // nessuno spazio nell'array

// Non corretto
function foo ( $bar )
{

}

// Corretto
function foo($bar) // nessuno spazio intorno alle parentesi
{

}

// Non corretto
foreach( $query->result() as $row )

// Corretto
foreach ($query->result() as $row) // un solo spazio per separare la funzione
    dall'argomento
```

## TESTO LOCALIZZATO

Qualsiasi testo in uscita nel pannello di controllo deve utilizzare i metodi messi a disposizione da CodeIgniter in modo da permettere agevolmente la localizzazione del testo nella propria lingua.



```
// Non corretto
return "Invalid Selection";

// Corretto
return $this->lang->line('invalid_selection');
```

## METODI PRIVATI E VARIABILI

I metodi e le variabili che sono unicamente accessibili tramite la propria classe come utility ed helper dovrebbero utilizzare un prefisso con il simbolo underscore:

```
convert_text() // metodo pubblico
_convert_text() // metodo privato
```

## ERRORI PHP

Il codice deve essere eseguito senza errori e senza fornire messaggi che segnalano malfunzionamenti (warning). Ad esempio, è una buona pratica non accedere mai ad una variabile (come `$ _POST`) senza prima aver controllato che esista con `isset()`.

Assicurarsi che durante lo sviluppo del proprio progetto, la segnalazione degli errori sia attivata per tutti gli utenti (ALL users), e che `display_errors` sia abilitata nell'ambiente [PHP](#) utilizzato. È possibile controllare questa impostazione con:

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

Su alcuni server dove `display_errors` è disabilitata e non si ha la possibilità di cambiare questa impostazione nel file **php.ini**, spesso è possibile attivarla con:

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

Nota: Impostare `display_errors` con `ini_set()` in fase di esecuzione non è identico ad averlo abilitato in ambiente [PHP](#). Vale a dire, che non avrà alcun effetto se lo script contiene errori fatali.

## SHORT OPEN TAG

Utilizzare sempre gli `open tag`, nel caso in server non abbia abilitato il supporto ai tag short: `short_open_tag enabled`.

```
// Non corretto
<? echo $foo; ?>

<?=$foo?>

// Corretto
<?php echo $foo; ?>
```

## UNA ISTRUZIONE PER LINEA

Abituarsi a sviluppare un codice pulito e chiaro, quindi mai combinare più istruzioni in una sola linea.

```
// Non corretto
$foo = 'this'; $bar = 'that'; $bat = str_replace($foo, $bar, $bag);

// Corretto
$foo = 'this';
$bar = 'that';
$bat = str_replace($foo, $bar, $bag);
```

## STRINGHE

Per racchiudere le stringhe si dovrebbero utilizzare sempre i singoli apici, invece delle virgolette (doppi apici). L'unica eccezione in cui sono ammissibili è quando all'interno della stringa si utilizza anche il singolo apice, in modo da evitare l'uso di simboli di escape.

```
// Non corretto
"My String" // usare i singoli apici
"My string $foo" // mancano le parentesi
'SELECT foo FROM bar WHERE baz = \'bag\'' // incomprensibile

// Corretto
'My String'
"My string {$foo}"
"SELECT foo FROM bar WHERE baz = 'bag'"
```

## 7.2 QUERY SQL

Le parole chiave di MySQL devono essere rappresentate in maiuscolo: SELECT, INSERT, UPDATE, WHERE, AS, JOIN, ON, IN, etc. Le query lunghe devono essere sempre spezzate su più righe preferibilmente una per ogni elemento chiave.

```
// Non corretto
// le parole chiave sono in minuscolo e
// la query è troppo lunga per una singola linea
$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");

// Corretto
$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");
```

## ARGOMENTI DI DEFAULT DELLA FUNZIONE

Quando è possibile, fornire sempre un argomento predefinito nelle funzioni. Questo aiuta a prevenire errori di PHP dovuti a chiamate sbagliate e fornisce valori certi di default. Esempio:

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

## 7.3 URI ROUTING E CACHING

Abbiamo già introdotto l'argomento del routing, ovvero dell'instradamento collegato agli [URI](#) nel capitolo [4.1 a pagina 33](#). In quella occasione abbiamo spiegato come il Controller, analizzando un indirizzo possa poi instradarci verso una sezione del sito, visualizzare un messaggio o interfacciarsi con una base di dati. In tale contesto abbiamo anche visto come CodeIgniter (ma anche tanti altri framework) adottino una struttura degli [URL](#) segmentata, ovvero separata da slash secondo una sintassi ben definita:

```
http://miosito.com/classeController/metodo/argomento
```

Questa struttura non è obbligatoria e può lasciare spazio a quella classica basata sulla visualizzazione delle variabili e dei loro valori, oppure può essere ridefinita a proprio piacimento tramite una procedura chiamata "rimappatura delle [URI](#)". In questo capitolo introdurremo le potenzialità dell'URI Routing, ovvero la relazione che intercorre tra la composizione di un [URI](#) e le corrispondenti classi e metodi di un Controller.

### IL CUORE DEL ROUTE

Come CodeIgniter ci ha abituato, esiste un file all'interno della sua struttura in cui è possibile analizzare e definire le regole del routing. Questo file è denominato **routes.php** e si trova al percorso **/application/config/**. All'interno vi è un vettore denominato `$route` che consente di specificare delle regole (routes) personalizzate per l'instradamento attraverso metacaratteri (wildcard) ed espressioni regolari. Insomma, configurando adeguatamente questo file è possibile sfruttare appieno le potenzialità di un framework. Analizziamo alcune delle possibilità offerte dall'opportuno settaggio del file **routes.php**.

### DEFINIRE UN CONTROLLER DI DEFAULT

Quando si sviluppa un progetto consistente, questo probabilmente conterrà anche diversi Controller. È importante quindi definire il "controller di default", quello che verrà caricato quando un [URI](#) non è definito, o quando si inserisce l'[URL](#) radice del proprio sito (miosito.com). Il Controller di default di CodeIgniter è:

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

Ebbene sì è lui il "responsabile" del caloroso messaggio di benvenuto. Possiamo modificare questo parametro inserendo il nome di qualsiasi Controller creato in precedenza, per caricarlo con la priorità massima.

Analizziamo, per fare un po' di pratica, anche altri valori del file **routes.php**:

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

Il valore “404 override” è la regola che interviene quando viene specificato un URL che non corrisponde ad alcun URI esistente.

```
// rimappatura con sostituzione di un singolo segmento  
$route['pagina'] = "sito";
```

In questo caso la regola di routing impone che il termine “pagina” sia rimappato nel primo segmento dell’URI tramite la classe “sito”.

```
// rimappatura di diversi segmenti  
$route['pagina/viaggi'] = "sito/news/10";
```

In questo secondo caso un’URI che contiene i segmenti “pagina/viaggi” sarà rimappata dalla classe “sito” e dal metodo “news”, mentre l’identificatore sarà impostato su “10”.

## RIMAPPARE LA CHIAMATA DEI METODI

Un’altra possibilità offerta è il rimappamento dei metodi tramite la funzione `_remap()` che si presenta con la sintassi:

```
public function _remap()  
{  
    // Qui ci va il codice  
}
```

Il compito di questa funzione è sovrascrivere la chiamata di un metodo, caricandone un altro. Un po’ come redirigere tutto il traffico diretto al punto A, verso un nuovo punto B.

```
public function _remap($metodo)
{
    if ($metodo == 'alcunimetodi')
    {
        $this->$metodo2();
    }
    else
    {
        $this->default_metodo();
    }
}
```

L'esempio appena mostrato contiene il metodo `_remap()` che accetta come argomento **\$metodo**. L'istruzione condizione *if* effettua una verifica: se il metodo in questione è uguale a quelli elencati in **alcunimetodi** allora verrà caricato **metodo2** (sovrascrittura o override del metodo), altrimenti *else* verrà caricato il `default_metodo`. Questo sistema, se padroneggiato, permette di redirigere le rotte generate da determinati **URI** alla destinazione desiderata, in situazioni particolari o di emergenza.

È possibile passare un secondo parametro opzionale al metodo `_remap()`. Nell'esempio che segue, l'array può infatti essere utilizzato in combinazione con la funzione del **PHP** `call_user_func_array` per emulare il funzionamento predefinito di 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();
}
```

## PROCESSARE L'OUTPUT

CodeIgniter possiede una classe di output che permette di inviare i dati da visualizzare automaticamente ad un browser web. Questa funzione chiamata `_output()` mostra automaticamente l'output finale. Ecco la sintassi:

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

## FUNZIONI PRIVATE

In alcune occasioni è necessario che certi metodi siano nascosti all'accesso pubblico, definendo per l'appunto un metodo "privato". La sintassi necessaria per rendere un metodo inaccessibile mediante l'URI è utilizzare il simbolo *underscore* (`_`) davanti al nome del metodo:

```
private function _utility()
{
    // il codice del metodo
}
```

Se ora si prova a digitare direttamente l'URI del metodo, questo non sarà accessibile pubblicamente:

```
miosito.com/index.php/blog/_utility/
```

## ORGANIZZARE UN GRANDE PROGETTO

Quando si realizza un progetto di grandi dimensioni è problematico inserire tutti i Controller, le Viste e i Modelli nelle loro directory principali, senza creare una gerarchia che metta ordine. È possibile, anzi è caldamente consigliato, organizzare il prodotto da sviluppare in sottodirectory omogenee e coerenti, ricordando che nella sintassi dell'URI si dovrà tenere conto della eventuale gerarchia, indicando anche le sottodirectory. Per esempio, se si inseriscono i Controller per la gestione delle scarpe di un magazzino, in una directory chiamata *prodotti*, il percorso per richiamare il Controller **scarpe** sarà:

```
application/controllers/prodotti/scarpe.php
```

## CLASSE COSTRUTTORE

Se si intende utilizzare un costruttore nel proprio Controller, è assolutamente necessario inserire al suo interno il codice:

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

Il motivo è presto detto: il costruttore locale dovrà essere sovrascritto da una classe controller genitore che deve essere richiamata manualmente:

```
<?php
class Blog extends CI_Controller {

    public function __construct()
    {
        parent::__construct();
        // Il codice del nostro Costruttore
    }
}
?>
```

I Costruttori sono utili se si ha la necessità di configurare alcuni valori di default, o eseguire un processo predefinito quando la propria classe viene istanziata. I Costruttori non restituiscono un valore ma sono utilizzati per svolgere dei lavori predefiniti.

## 7.4 UTILIZZO DI WILDCARD ED REGEX

Solitamente vi è una corrispondenza uno-a-uno tra la stringa [URL](#) e la classe/metodo che fa capo ad un Controller. È possibile in tutti i casi modificare la struttura con cui CodeIgniter mappa ogni [URI](#), magari perché si proviene da un altro framework, o solo per il gusto di sperimentare. Per apportare le modifiche alle regole di routing si deve agire sul file **config** in **/application/config/** e utilizzare le Regex o le wildcard<sup>7</sup>

## 7.5 WILDCARD

Solitamente un rotta in cui è presente una wildcard ha una forma del tipo:

<sup>7</sup> Definito anche metacarattere o carattere jolly o wild character o wildcard character, è un carattere che, all'interno di una stringa, non rappresenta sé stesso bensì un insieme di altri caratteri o sequenze di caratteri.



```
$route['prodotto/:num'] = "catalogo/cercaprodotti";
```

La chiave della variabile `$route` definisce l'URI con cui verificare una corrispondenza. Nell'esempio, la chiave è costituita da due segmenti distinti. Il primo è identificato con la stringa `prodotto`, mentre il secondo rappresenta un wildcard. Si può interpretare come: *se l'URI contiene la parola stringa prodotto (primo segmento) seguito da uno o più numeri (secondo segmento) allora eseguire il metodo cercaprodotti del Controller catalogo.*

Per verificare valori alfanumerici si utilizzano i seguenti wildcard:

- `(:num)` cercherà una corrispondenza nella parte della stringa che contiene solo numeri
- `(:any)` cercherà una corrispondenza nella parte della stringa che contiene ogni carattere

Un aspetto da tenere sempre a mente è che le regole di routing vengono eseguite secondo l'ordine in cui sono definite. È un aspetto questo da non sottovalutare perché porta soventemente ad errori banali, ma fastidiosi da individuare e correggere.

Qui di seguito riportiamo alcuni esempi che vi illustreranno la potenza e facilità d'uso di questo sistema:

Tutte gli URI che contengono la stringa **riviste** saranno instradate verso il Controller **blog**:

```
$route['riviste'] = "blog";
```

Tutte gli URI che contengono la stringa, questa volta composta, **blog/joe** saranno instradate verso il Controller **blog**, di cui sarà eseguito il metodo **utente** con parametro **34**:

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

Nell'esempio seguente il nostro URI è composto da una stringa, seguita dalla wildcard **:any**. Quindi tutti gli URI che contengono **prodotto** e subito dopo qualsiasi carattere, saranno instradate verso il Controller **catalogo**, di cui sarà eseguito il metodo **cercaprodotto** con parametro **34**:

```
$route['prodotto/(:any)'] = "catalogo/cercaprodotto/34";
```

Questo esempio è un po' più complesso, ma mostra la flessibilità di CodeIgniter. Ogni URI che contiene la stringa `prodotto`, seguita da un valore numerico (wildcard `:num`) viene instradato verso la classe `catalogo`, di cui viene eseguito il metodo `cerca_prodotto_con_id`. Sin qui dovrebbe essere chiaro, ma è la parte che viene

subito dopo a risultare inedita e molto interessante. L'argomento del nostro metodo viene indicato con `$1`, dove il simbolo dollaro indica appunto una variabile: infatti non stiamo indicando un argomento specifico, ma uno che assume un valore differente a seconda del numero specificato nella wildcard **:num**.

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

In questo modo se specifichiamo un **URI** come `prodotto/33` CodeIgniter eseguirà il Controller-metodo `catalogo/cerca_prodotto_con_id/33`. In pratica abbiamo definito l'instradamento che si utilizza per cercare/visualizzare un prodotto in un archivio, senza dover specificare il prodotto manualmente! Una bella comodità non credete?

I più attenti si saranno posti maggiori domande sulla sintassi dell'argomento variabile `$1`. Come è facilmente intuibile, esso si riferisce al primo wildcard, ma nulla ci vieta di specificare `$2` per indicare il secondo wildcard e così via.

## 7.6 REGULAR EXPRESSION

Comunemente chiamate Regex o espressioni regolari in italiano, sono un metodo potente e altamente personalizzabile per definire le regole di routing. Di contro, imparare a gestirle pienamente non è facile. È possibile utilizzare anche le back-reference, ma in questo caso si deve utilizzare il simbolo del dollaro piuttosto che la sintassi con la doppia barra rovesciata.

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

Nell'esempio precedente, un **URI** simile a `prodotto/camicia/123` richiamerebbe il Controller `camicia` e la funzione `id_123`: `prodotto/camicia/id_123`.

Si possono anche mescolare e abbinare caratteri jolly con le espressioni regolari.

## 7.7 ROTTE RISERVATE

Esistono due rotte riservate:

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

Questa rotta indica quale Controller dovrebbe essere caricato se l'**URI** non contiene alcuno dato: in pratica, si definisce la rotta radice del sito. Nell'esempio precedente viene caricata la classe **welcome**. È importante definire una rotta radice, altrimenti, in caso di errore, verrà visualizzata una pagina predefinita con l'errore 404:

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

La rotta di qui sopra indica quale Controller deve essere caricato, se l'[URI](#) fornito dall'utente non corrisponde a nessun Controller tra quelli disponibili. Se si configura questa rotta, verrà sovrascritta la pagina di default che gestisce l'errore 404. Se invece, non si fornisce alcun valore, verrà eseguita la funzione `show_404()` che si trova nel file `error_404.php` nel percorso **/application/errors/**.

Queste, appena indicate, sono le uniche due rotte riservate. Ed esse devono avere la precedenza su qualsiasi rotta che preveda wildcard o espressioni regolari.

## 7.8 NOMI RISERVATI

Alcuni nomi riguardanti Controller, metodi, variabili e costanti non possono essere utilizzati liberamente dall'utente: essi sono riservati da CodeIgniter per definire alcuni elementi chiave del framework. È importante non utilizzare i seguenti nomi, altrimenti le proprie funzioni sovrascriveranno quelle di default, con conseguenze difficilmente prevedibili. Qui di seguito proponiamo una lista dei nomi riservati:

### CONTROLLER

```
CI_Base  
_ci_initialize  
Default  
index
```

### METODI

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

### VARIABILI

```
$config  
$mimes  
$lang
```

### COSTANTI

```
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

## 7.9 SCRIPTING

Gli engine dei template non possono eguagliare le performance del [PHP](#) nativo e la sintassi che si deve imparare per padroneggiarli di solito è solo marginalmente più facile che apprendere le basi del [PHP](#). Consideriamo questa porzione di codice:

```
<ul>
<? php foreach ($ rubrica da $ nome):>
<li><? = $ name></ li>
<php endforeach; ?>
</ul>
```

Ed ecco lo pseudo-codice utilizzato da un engine di template che svolge lo stesso compito:

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

Sì, l'esempio dell'engine di template è leggermente più pulito, ma questo si paga con delle prestazioni minori, poiché lo pseudo-codice deve essere poi convertito in [PHP](#) per funzionare. Poiché uno dei nostri obiettivi è il massimo delle prestazioni, abbiamo deciso di non rendere obbligatorio l'uso di un engine di template.

# 8

## ELENCO DELLE CLASSI

In questo capitolo verranno esaminate le Classi e i Driver presenti in CodeIgnitor che offrono funzionalità potenti e flessibili per lo sviluppo e la gestione del proprio progetto.

Lista delle classi

- classe Benchmarking
- classe Caching (driver)
- classe Calendar
- classe Cart
- classe Config
- classe Database (driver)
- classe Email
- classe Encryption
- classe File Uploading
- classe Form Validation
- classe FTP
- classe HTML Table
- classe Image Manipulation
- classe Input
- classe Javascript (driver)
- classe Loader
- classe Language
- classe Migration
- classe Output
- classe Pagination
- classe Security

- classe Session
- classe Trackback
- classe Template Parser
- classe Typography
- classe Unit Testing
- classe URI
- classe User Agent
- classe XML-RPC
- classe Zip Encoding

#### Driver

- classe Caching
- classe Database
- classe Javascript



## 8.1 CLASSE BENCHMARKING

La classe Benchmarking svolge un semplice quanto prezioso compito: calcola la differenza di tempo tra due marcatori fissati. Il valore, espresso in secondi, permette di effettuare delle stime sulla velocità del progetto sviluppato, calcolando il tempo necessario per processare una serie di istruzioni, o caricare una vista ricca di dati e immagini. Non meno importante è la funzione che misura la quantità di memoria impegnata dal progetto, che dovrà essere sempre opportunamente dimensionato all'hardware del server su cui viene eseguito. Questa classe non ha bisogno, a differenza di altre, di essere abilitata manualmente, ma al contrario risulta sempre attiva: essa viene avviata non appena si utilizza il framework e termina poco prima che la classe output invii i dati alla Vista che li mostra sul browser.

Utilizzare la classe

La classe Benchmarking può essere utilizzata con i Controller, Viste e Modelli seguendo i seguenti passi:

1. creare un punto marcatore iniziale
2. creare un punto marcatore finale
3. eseguire la funzione “elapsed time” per visualizzare i risultati

Un esempio del codice vero e proprio è il seguente:

```
$this->benchmark->mark('code_start');  
  
// spazio per il codice  
  
$this->benchmark->mark('code_end');  
  
echo $this->benchmark->elapsed_time('code_start', 'code_end');
```

Nota: le parole `code_start` e `code_end` sono arbitrarie: è possibile utilizzare qualsiasi parola, che indichi preferibilmente in maniera chiara l'inizio e la fine di un blocco di codice da analizzare. Nell'esempio di qui sotto sono state utilizzate tre parole chiave che non risultano intuitive nel definire i marcatori di inizio e fine codice. Comunque qualsiasi scelta si faccia, CodeIgniter calcolerà il tempo correttamente:

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

// spazio per il codice

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

// spazio per il codice

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

echo $this->benchmark->elapsed_time('cane', 'gatto');
echo $this->benchmark->elapsed_time('gatto', 'uccello');
echo $this->benchmark->elapsed_time('cane', 'uccello');
```

### Profilazione dei punti benchmark

Un aspetto interessante di questa classe è che si possono utilizzare i dati del benchmark condividendoli con il Profiler/vrefcap:profilazione. È sufficiente collegare in coppia i propri marcatori, facendo terminare i rispettivi nomi con le stringhe `_start` e `_end`. Altro aspetto importante è che ogni coppia di marcatori deve avere necessariamente lo stesso nome. Per esempio:

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

// spazio per il codice

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

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

// spazio per il codice

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

Maggiori informazioni sono disponibili nel capitolo dedicato alla Profilazione a pagina [96](#).

### Il tempo totale di esecuzione

Se si desidera visualizzare il tempo trascorso dal momento in cui CodeIgniter viene eseguito per la prima volta sino al momento dell'output finale verso il browser, è sufficiente porre in una delle proprie Viste la seguente istruzione:

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

Si noterà che la funzione è la stessa, utilizzata negli esempi precedenti per calcolare il tempo tra due punti, ma in questo caso, non si specifica alcun argomento. Quando i parametri sono assenti, CodeIgniter non termina il benchmark fino a quando il risultato finale non viene inviato al browser. Non importa dove si utilizzerà la chiamata di funzione, il timer continuerà a funzionare fino alla fine. Un modo alternativo per mostrare il tempo trascorso nei file di vista è quello di utilizzare questa pseudo-variabile, che evita di ricorrere al [PHP](#) puro:

```
{elapsed_time}
```

## IL CONSUMO DI MEMORIA

Il consumo di memoria invece è un altro aspetto importante che è opportuno monitorare nei progetti che coinvolgono ingenti quantità di dati. Se l'installazione del [PHP](#) sul sistema è configurata con l'opzione `--enable-memory-limit`, sarà possibile visualizzare il consumo di memoria dell'intero sistema, inserendo nelle Viste desiderate:

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

Nota: questa funzione può essere utilizzata solo nelle Viste, dove verrà visualizzata la quantità di memoria consumata dal progetto. Un metodo alternativo per ottenere lo stesso risultato, prevedo l'uso del pseudo-codice:

```
{memory_usage}
```

## 8.2 CLASSE/DRIVER CACHING

CodeIgniter offre i wrapper per alcune delle più popolari forme di caching. Tutte le modalità di caching, tranne quello basato su file, hanno bisogno che il server soddisfi determinati requisiti: se questi non vengono rispettati viene sollevato un “Fatal Exception”.

### Definizione 9: Wrapper

---

<b>Wrapper</b>	Un wrapper (dal verbo inglese to wrap, “avvolgere”) è un modulo software che ne “riveste” un altro, o detto con un’altra terminologia che lo “incapsula”. La sua funzione generalizzata è quella di fare da tramite tra l’utente ultimo (colui che usa l’interfaccia del wrapper) e il modulo rivestito (che svolge i servizi richiesti, su delega dell’oggetto wrapper). Il wrapper può essere una classe, un oggetto che ne riveste e nasconde un altro (o più di uno) celando alcuni metodi/campi oppure mostrandone di nuovi o ridefiniti. L’oggetto wrapper avrà le medesime funzionalità di quello incluso, ma modellato ad uso e consumo degli utilizzatori finali.
----------------	--

---

Il seguente esempio mostra il caricamento del driver cache Alternative PHP Cache ([APC](#)). Si noti l’uso dell’istruzione condizionale, sempre consigliata nella buona programmazione: se il sistema di caching APC non è disponibile nell’ambiente host di riferimento la condizione if/else permette di “ripiegare” su un caching basato su file. Per chi volesse approfondire il discorso legato all’ottimizzazione degli script PHP basati su [APC](#) può consultare la sezione apposita a pagina [428](#).

```
$this->load->driver('cache', array('adapter' => 'apc', 'backup' => 'file'));

if ( ! $foo = $this->cache->get('foo'))
{
    echo 'Salvataggio nella cache!<br />';
    $foo = 'foobarbaz!';

    // Salvataggio nella cache per 5 minuti
    $this->cache->save('foo', $foo, 300);
}

echo $foo;
```

## REFERENCE

- `is_supported(driver['string'])` questa funzione è chiamata automaticamente quando si accede ai driver mediante la funzione `$this->cache->get()`. Tuttavia, se vengono utilizzati i singoli driver, ci si deve assicurare di chiamare questa funzione per verificare che il driver sia supportato nell'ambiente hosting.

```
if ($this->cache->apc->is_supported())
{
    if ($data = $this->cache->apc->get('my_cache'))
    {
        // fai qualcosa...
    }
}
```

- `get(id['string'])` questa funzione tenta di recuperare un oggetto dalla cache. Se l'oggetto non esiste, la funzione restituisce FALSE.

```
$foo = $this->cache->get('my_cached_item');
```

- `save(id['string'], data['mixed'], ttl['int'])` memorizza un oggetto nella cache. Se l'operazione non va a buon fine, la funzione restituisce FALSE. Il terzo parametro opzionale (Time To Live) ha un valore di default di 60 secondi.

```
$this->cache->save('cache_item_id', 'data_to_cache');
```

- `delete(id['string'])` permette di cancellare uno specifico oggetto memorizzato nella cache. Se l'eliminazione fallisce, la funzione restituisce FALSE.

```
$this->cache->delete('cache_item_id');
```

- `clean()` questa funzione ripulisce l'intera cache. Anche in questo caso, se l'operazione fallisce, restituisce FALSE.

```
$this->cache->clean();
```

- `cache_info()` questa funzione restituisce informazioni sulla cache.

```
var_dump($this->cache->cache_info());
```

- `get_metadata(id['string'])` vengono restituite informazioni dettagliate su un oggetto specifico nella cache.

```
var_dump($this->cache->get_metadata('my_cached_item'));
```

## DRIVER

- Alternative PHP Cache (APC) Caching: è possibile accedere a tutte le funzioni elencate precedentemente senza passare uno specifico adapter al driver loader, come qui di seguito:

```
$this->load->driver('cache');  
$this->cache->apc->save('foo', 'bar', 10);
```

Per maggiori informazioni su [APC](http://php.net/apc) si faccia riferimento alla documentazione ufficiale: <http://php.net/apc>.

- File-based Caching: a differenza del caching tramite la Classe Output, quello basato su driver permette che porzioni di file Vista possano essere memorizzati nella cache. È consigliato utilizzare questa modalità di caching con molta attenzione ed effettuando ulteriori test di benchmark (si veda [8.1 a pagina 143](#)) per la propria applicazione. Purtroppo, i troppi accessi al disco per operazioni di input/output legati alla cache, invece di aumentare le prestazioni del proprio progetto, possono avere un effetto deleterio.

Tutte le funzioni sopra elencate sono accessibili senza passare uno specifico adapter al driver loader come segue:

```
$this->load->driver('cache');  
$this->cache->file->save('foo', 'bar', 10);
```

- Memcached Caching: più server Memcached possono essere specificati nel file di configurazione **memcached.php**, che si trova nella directory **/application/-config/**.

Tutte le funzioni sopra elencate sono accessibili senza passare uno specifico adapter al driver loader come segue:

```
$this->load->driver('cache');  
$this->cache->memcached->save('foo', 'bar', 10);
```

Per maggiori informazioni su Memcached si faccia riferimento alla documentazione: <http://php.net/memcached>.

- Dummy Cache: non memorizza dati, ma consente di mantenere il proprio codice di caching in ambienti che non supportano la cache scelta.

### 8.3 CLASSE CALENDAR

La classe Calendar permette di creare dinamicamente un calendario, formattato secondo un template di cui si potrà modificare ogni più piccolo aspetto. Ovviamente potremo passargli anche dei dati.

#### INIZIALIZZARE LA CLASSE

Come molte altre classi di CodeIgniter, quella di cui si sta parlando, può essere inizializzata nel proprio controller utilizzando la funzione `$this->load->library`:

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

#### VISUALIZZARE UN CALENDARIO

Una volta caricato, l'oggetto Calendar sarà disponibile con `$this->calendar`. Si può effettuare una prova inserendo nel proprio controller il codice:

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

In questo modo verrà generato un calendario con il mese/anno corrente del proprio server. Per visualizzare invece un calendario con una data personalizzata si dovranno utilizzare degli specifici argomenti, come nel seguente esempio che visualizza il mese di giugno del 2014: [19](#)

```

<<      Feb 1974      >>
Sun Mon Tue Wed Thu Fri Sat
                        1  2
3   4   5   6   7   8   9
10  11  12  13  14  15  16
17  18  19  20  21  22  23
24  25  26  27  28

```

Fig. 19: Calendario



```
$this->load->library('calendar');  
  
echo $this->calendar->generate(2014, 6);
```

## PASSARE I DATI ALLE CELLE

Per aggiungere o meglio personalizzare i dati delle celle del calendario si definisce un array associativo i cui indici si riferiscono ai giorni che si vogliono “popolare”, e i valori ai dati personalizzati:

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

In questo esempio gli indici dell’array (3, 7, 13, e 26) definiscono i giorni a cui vogliamo collegare i valori dell’array, che in questo caso corrispondono a degli [URI](#). Il risultato sarà quello di visualizzare un calendario in cui alcuni giorni saranno collegati con dei link a determinate pagine web.

Nota: solitamente i dati dell’array corrispondono a dei link. Questo esempio e i prossimi saranno basati su questa prassi, ma nulla vieta di passare altri tipi di informazioni all’array.

## CONFIGURAZIONE

Avevamo accennato al fatto che la classe Calendario potesse essere configurata in ogni suo aspetto. Le preferenze si basano su un particolare array `$pref` i cui parametri permettono di rendere il calendario molto flessibile in ogni particolare: è possibile decidere il primo giorno della settimana, il formato di visualizzazione del mese o del giorno. La sintassi utilizzata può essere meglio compresa nel codice seguente, in cui viene definito il primo giorno della settimana (sabato) e il formato del mese (completo) e del giorno (ridotto). In questo caso, quando verrà caricata la classe Calendario verrà passato anche il secondo parametro, l’array `$pref` opportunamente configurato.

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

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

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

Preferenze	Default	Opzioni	Descrizione
template	nessuno	nessuno	Una stringa contenente il template del calendario*
local_time	time()	nessuno	Il timestamp Unix che corrisponde all'ora corrente
start_day	sunday	giorni settimana	Imposta il primo giorno della settimana
month_type	long	long, short	Imposta il formato del mese. long = January, short = Jan
day_type	abr	long, short, abr	Imposta il formato dei giorni. long = Sunday, short = Sun, abr = Su
show_next_prev	FALSE	TRUE/FALSE	Imposta la visualizzazione dei link per scorrere i mesi*
next_prev_url	nessuno	URL	Imposta il basepath per i link del calendario

\*Si veda la sezione template per maggiori informazioni

## SCORRERE I MESI

Il calendario può essere “visitato” dinamicamente attraverso dei link che permettono di incrementare/decrementare i mesi.

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

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

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

Si noteranno alcune cose circa l'esempio precedente:

- è necessario impostare la `show_next_prev` su `TRUE`
- è necessario fornire al controller l'[URL](#) contenente il calendario al parametro `next_prev_url`
- è necessario fornire l'anno e il mese alla funzione che genera il calendario tramite i singoli segmenti di [URI](#) in cui appaiono (nota: la classe del calendario aggiunge automaticamente l'anno/mese all'[URL](#) base fornito)

L'esempio seguente genererà il calendario attuale attraverso il metodo `view()` definito nel controller **Pages**. Si notino i link per la navigazione tra i mesi precedente e successivo.

```
public function view($page='home')
{
    $prefs = array (
        'show_next_prev' => TRUE,
        'next_prev_url'  => 'http://127.0.0.1/index.php/pages/view/'
    );
    this->load->library('calendar', $prefs);
    echo $this->calendar->generate($this->uri->segment(3), $this->uri->segment
        (4));
}
```

<a href="#">&lt;&lt;</a>		<b>Jun 2014</b>					<a href="#">&gt;&gt;</a>
Sun	Mon	Tue	Wed	Thu	Fri	Sat	
1	2	3	4	5	6	7	
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	
22	23	24	25	26	27	28	
29	30						

Fig. 20: Calendario navigabile

## PERSONALIZZARE IL TEMPLATE

L'aspetto grafico del calendario è modificabile agendo su alcuni parametri delle seguenti pseudo variabili:

```

$prefs['template'] = '

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

{heading_row_start}<tr>{/heading_row_start}

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

{heading_row_end}</tr>{/heading_row_end}

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

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

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

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

{cal_cell_blank}&nbsp;{/cal_cell_blank}

{cal_cell_end}</td>{/cal_cell_end}
{cal_row_end}</tr>{/cal_row_end}

{table_close}</table>{/table_close}
';

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

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

```

## 8.4 CLASSE SHOPPING CART

La classe Cart (carrello) mette a disposizione dello sviluppatore delle comode funzionalità per progetti di e-commerce. Fulcro della classe sono gli elementi (per esempio i prodotti acquistati) che possono essere aggiunti alla sessione attiva sin quando l'utente si trova sul sito sviluppato con CodeIgniter. Questi elementi possono essere recuperati e visualizzati in un formato standard "shopping cart", che consente all'utente di aggiornare la loro quantità, oppure rimuoverli dal carrello.

Questa classe automatizza molti compiti legati alla sessione, ma non prevede funzionalità aggiuntive come lo shopping, l'autorizzazione delle carte di credito o altre caratteristiche che si basano sull'analisi e il processo di dati.

Prima di utilizzare la classe Cart è importante fare alcune precisazioni. La prima è che Cart si basa a sua volta sulla classe Session per salvare le informazioni del carrello in un database: quindi prima di utilizzare la classe si deve configurare la tabella del database come indicato nella documentazione delle Sessioni [8.23 a pagina 322](#) e definire le preferenze nel file `/application/config/config.php` per utilizzare una base di dati.

### INIZIALIZZARE LA CLASSE

Per inizializzare la classe Shopping Cart nel proprio costruttore del Controller è necessario usare la seguente funzione adibita al caricamento:

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

Una volta caricata, l'oggetto Cart sarà disponibile usando:

```
$this->cart
```

Nota: la classe Cart caricherà e inizierà la classe Session automaticamente, quindi a meno che non si stiano utilizzando altre sessioni nell'applicazione, non è necessario caricare manualmente la classe Session.

### AGGIUNGERE UN OGGETTO AL CARRELLO

Per aggiungere un oggetto al carrello della spesa è sufficiente passare un array con le informazioni dei prodotti alla funzione `$this->cart->insert()` come mostrato nell'esempio:

```
$data = array(
    'id'      => 'sku_123ABC',
    'qty'     => 1,
    'price'   => 39.95,
    'name'    => 'T-Shirt',
    'options' => array('Taglia' => 'L', 'Colore' => 'Rosso')
);
```

I primi quattro indici dell'array (id, qty, price, name) sono obbligatori: se ommessi tutti o solo uno di essi, i dati non verranno memorizzati nel carrello. Il quinto indice (options) è invece opzionale: è utilizzato nei casi in cui il prodotto contenga ulteriori opzioni da associare; in questo caso viene passato un nuovo array, come nell'esempio in cui si sono specificati anche la taglia e il colore dell'articolo.

**ID** ogni prodotto nel negozio deve essere caratterizzato da un identificatore univoco.

L'id, solitamente rappresentato da un numero, ha un prefisso per garantire una maggiore sicurezza. Negli esempi di questa classe verrà utilizzato il prefisso "sku"

**QTY** la quantità dell'oggetto da acquistare

**PRICE** il prezzo

**NAME** il nome dell'oggetto

**OPTIONS** un array aggiuntivo per inserire altre informazioni (questo parametro è opzionale)

#### Definizione 10: Identificatore

<b>Identificatore</b>	É un elemento che definisce l'identità di un elemento di un insieme senza che questo possa essere confuso con un altro dello stesso insieme.
-----------------------	--

Esistono, oltre a questi cinque indici, due parole riservate: **rowid** e **subtotal**. Questo significa che tali nomi non devono essere utilizzati per definire gli indici dell'array.

L'array dei prodotti può contenere dati aggiuntivi: tutto ciò che si include verrà memorizzato nella sessione. Tuttavia è meglio adottare uno standard comune tra i dati di tutti i prodotti: in questo modo la visualizzazione delle informazioni in una tabella risulterà omogenea. Il metodo **insert()** restituirà **\$rowid** (l'identificatore del prodotto) solo se il dato sarà inserito correttamente nel database.

## AGGIUNGERE PIÙ OGGETTI

Per aggiungere più prodotti al carrello bisogna affidarsi ad un array multidimensionale come nell'esempio: in questo modo tutti i dati vengono inseriti in una singola azione. Questo è utile per permettere agli utenti di scegliere tra diversi elementi collocati sulla stessa pagina.

```
$data = array(
    array(
        'id'      => 'sku_123ABC',
        'qty'     => 1,
        'price'   => 39.95,
        'name'    => 'T-Shirt',
        'options' => array('Taglia' => 'L', 'Colore' => 'Red')
    ),
    array(
        'id'      => 'sku_567ZYX',
        'qty'     => 1,
        'price'   => 9.95,
        'name'    => 'Tazza da écaff'
    ),
    array(
        'id'      => 'sku_965QRS',
        'qty'     => 1,
        'price'   => 29.95,
        'name'    => 'Occhiali'
    )
);

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

## VISUALIZZARE IL CARRELLO

Per visualizzare il carrello si utilizzano, come è facile immaginare, una Vista con un codice simile a quello riportato nell'esempio e l'Helper Form (si veda a pagina 389):

```
<?php echo form_open('path/to/controller/update/function'); ?>

<table cellpadding="6" cellspacing="1" style="width:100%" border="0">

<tr>
    <th>QTY</th>
    <th>Descrizione dell'oggetto</th>
    <th style="text-align:right">Prezzo dell'oggetto</th>
    <th style="text-align:right">totale parziale</th>
</tr>
```

```

</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>Totale</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>

```



## AGGIORNARE IL CARRELLO

L'aggiornamento del carrello con le informazioni dei prodotti in esso contenuti, avviene passando alla funzione `$this->cart->update()` un array contenente l'id della riga (Row ID) e la relativa quantità. Un aspetto utile è che, se la quantità dell'oggetto in questione è zero, l'oggetto automaticamente verrà rimosso dal carrello.

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

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

// Oppure un array multidimensionale

$data = array(
    array(
        'rowid' => 'b99ccdf16028f015540f341130b6d8ec',
        'qty'   => 3
    ),
    array(
        'rowid' => 'xw82g9q3r495893iajdh473990rikw23',
        'qty'   => 4
    ),
    array(
        'rowid' => 'fh4kdkkkaoe30njgoe92rkdckkobec333',
        'qty'   => 2
    )
);

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

Ma cosa è un **ID Row**? Si tratta di un identificatore univoco che viene generato dal codice del carrello quando un elemento viene aggiunto. La ragione per cui viene creato un ID univoco è per consentire la gestione di prodotti identici ma con piccole differenze significative. Si immagini per esempio di acquistare due birre, identiche in tutto, tranne per il fatto che una è “analcolica” e l'altra no. Un ulteriore esempio è dato da due t-shirt della stessa marca, ma con taglie differenti: l'ID del prodotto (ma anche altri attributi) risulteranno così identici proprio perché ci si riferisce al medesimo prodotto, ovvero una t-shirt. Il carrello deve quindi permettere di identificare questa differenza sottile, ma importante in modo che le due taglie (e i relativi prodotti) possano essere gestiti indipendentemente l'uno dall'altro. CodeIgniter rende possibile tutto questo creando un unico “ID Row” in base all'ID del prodotto e alle eventuali opzioni associate.

In quasi tutti i casi, l'aggiornamento del carrello sarà un'operazione che l'utente eseguirà attraverso la pagina “Visualizza carrello”. In pratica non ci si dovrà preoccupare dell'ID Row perché CodeIgniter lo celerà in un opportuno campo nascosto per

poi passarlo alla funzione demandata all'aggiornamento del carrello. Per maggiori informazioni si rimanda alla sezione "Visualizza il carrello" a pagina [157](#).

## REFERENCE

- `$this->cart->insert()` è utilizzata per aggiungere i prodotti al carrello.
- `$this->cart->update()` permette di aggiornare il carrello.
- `$this->cart->total()` visualizza il totale del carrello.
- `$this->cart->total_items()` visualizza il numero totale di oggetti nel carrello.
- `$this->cart->contents()` restituisce un array che contiene qualsiasi elemento nel carrello.
- `$this->cart->has_options(rowid)` restituisce TRUE se una particolare riga nel carrello contiene delle opzioni. Questa funzione è comoda se utilizzata in un ciclo con `$this->cart->contents()` poiché è necessario passare rowid a questa funzione, come mostrato nell'esempio "Visualizza il carrello" a pagina [157](#).
- `$this->cart->product_options(rowid)` restituisce un array di opzioni per un particolare prodotto. Questa funzione è disegnata per essere usata in un ciclo con `$this->cart->contents()` poiché è necessario passare rowid a questa funzione, come mostrato nell'esempio "Visualizza il carrello" a pagina [157](#).
- `$this->cart->destroy()` permette di eliminare il carrello. Questa funzione solitamente è richiamata quando termina il processo di acquisto da parte dell'utente e si vuole concludere l'ordine.

## 8.5 CLASSE CONFIG

La classe Config (abbreviazione di Configuration) fornisce un mezzo per recuperare le preferenze del sistema, che possono provenire dal file **/application/config/config.php** o dai file di configurazione personalizzati. Questa classe viene inizializzata automaticamente dal sistema e non è necessario farlo manualmente. Per impostazione predefinita CodeIgniter ha un solo file di configurazione principale, situato in **/application/config/config.php** in cui i singoli parametri sono memorizzati in un array chiamato **\$config**. È possibile personalizzare ogni singolo aspetto semplicemente aggiungendo o modificando i parametri di default; oppure se lo si preferisce (e in genere è una scelta sensata) si può creare un proprio file di configurazione che andrà salvato sempre nella cartella **/application/config/**.

Se si opta per quest'ultima scelta, si utilizzerà la stessa sintassi presente nel file di default: l'array si chiamerà **\$config** e i parametri rispetteranno la sintassi di base. CodeIgniter, se si rispettano queste semplici regole, capirà automaticamente come comportarsi in ogni occasione. Per esempio, in caso di parametri di default e "personalizzati" con lo stesso identico nome, CodeIgniter imporrà sempre quello corretto, evitando spiacevoli conflitti. Come precedentemente detto, il file principale di configurazione viene caricato automaticamente, quindi lo sviluppatore dovrà preoccuparsi solo della configurazione personalizzata, se è stato creato un nuovo file apposito: in questo caso, esso andrà caricato manualmente o attraverso l'impostazione l'auto-load.

### CARICAMENTO MANUALE

Analizziamo il caricamento manuale come prima scelta. La seguente istruzione, opportunamente inserita nel Controller, caricherà il file "filename" (che nel sistema avrà estensione *.php*).

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

Inoltre se si ha bisogno di suddividere i parametri di configurazione in più file, questo è possibile definendo un array per ogni singolo file di configurazione: normalmente questi file verranno uniti per definire un unico array di configurazione. Un problema che potrebbe verificarsi utilizzando questo approccio, è il verificarsi di collisioni, che si hanno quando si nominano gli indici degli array nei diversi file, nello stesso identico modo. Per evitare questo inconveniente è consigliato utilizzare un secondo parametro booleano TRUE che farà sì che ciascun file di configurazione venga memorizzato in un indice dell'array corrispondente al proprio nome. Per esempio:

```
// Memorizzato in un array con questo prototipo: $this->config['blog_settings'] = $config  
  
$this->config->load('blog_settings', TRUE);
```

Per maggiori informazioni leggere con attenzione la sezione “Fetching Config Items” su come recuperare gli elementi che definiscono la configurazione.

Esiste anche un terzo parametro, sempre booleano, che evita di visualizzare gli errori che possono scaturire quando si vuole caricare un file di configurazione che in realtà non esiste.

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

#### Auto-load

Per caricare automaticamente il proprio file di configurazione si agisce sul file **autoload.php** in **/application/config/** in cui si aggiunge il riferimento al proprio file.

## RECUPERARE PARTI DI CONFIGURAZIONI

Per recuperare un singolo elemento dal file di configurazione, si utilizza la seguente funzione, in cui l'argomento di **item** è il nome dello specifico indice (item name) dell'array **\$config**.

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

Nell'esempio seguente si recupera il parametro relativo al linguaggio (language) scelto:

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

La funzione restituisce FALSE se l'elemento che si vuole recuperare non esiste. Se si sta utilizzando il secondo parametro booleano nella funzione **\$this->config->load** per assegnare i parametri di configurazione ad uno specifico indice, allora è possibile utilizzare un metodo alternativo per recuperare i singoli elementi: basterà riportare il nome dell'indice desiderato nel secondo argomento della funzione **\$this->config->item()**, come nell'esempio:

```
// Carica un file di configurazione di nome blog_settings.php e lo assegna ad
    un indice denominato "blog_settings"
$this->config->load('blog_settings', TRUE);

// Recupera un item di configurazione chiamato site_name contenuto all'interno
    dell'array blog_settings
$site_name = $this->config->item('site_name', 'blog_settings');

// Un modo alternativo per specificare lo stesso item:
$blog_config = $this->config->item('blog_settings');
$site_name = $blog_config['site_name'];
```

Per chi avesse bisogno di impostare o cambiare dinamicamente un parametro di configurazione, può utilizzare l'istruzione:

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

Dove `item_name` è l'indice dell'array che si vuole modificare, mentre `item_value` definisce il relativo valore.

## AMBIENTI

A seconda dell'ambiente (Environment) è possibile caricare differenti file di configurazione. La costante **ENVIRONMENT** è definita nel file **index.php**. Per creare un file di configurazione per uno specifico ambiente, si dovrà creare o copiare un file di configurazione in **/application/config/ENVIRONMENT/FILENAME.php**. Per esempio per creare una configurazione **config.php** rivolta all'ambiente di produzione si dovrà:

1. creare la directory **/application/config/production/**
2. copiare l'esistente file **config.php** nel percorso precedentemente menzionato
3. modificare il file **/config/production/config.php** con i parametri desiderati

A questo punto, per caricare la nuova configurazione si imposta la costante **ENVIRONMENT** su "production". I file di configurazione così definiti potranno essere posti all'interno di directory specifiche per l'ambiente desiderato. Riassumendo, sia i file di configurazione predefiniti di CodeIgniter che quelli da noi "personalizzati", possono essere raccolti in opportune directory in modo da creare più ambienti per la produzione o per lo sviluppo.

Nota: CodeIgniter, prima di tutto, prova a caricare il file di configurazione per l'ambiente corrente. Se questo non esiste, viene caricato il file di configurazione globale (in **/application/config/**). Questo significa che non si è obbligati ad inserire tutti i file di configurazione in una sola cartella, ma solo quelli che presentano delle differenze tra i vari ambienti.

## REFERENCE

La classe Config può contare sui seguenti Helper che aiutano lo sviluppatore in numerosi compiti:

- `$this->config->site_url()` recupera l'[URL](#) e lo fornisce al proprio sito insieme con il valore "index" che si è specificato nel file di configurazione. Questa funzione è accessibile attraverso la corrispondente funzione nell'Helper URL (sezione [9.20 a pagina 422](#)).
- `$this->config->base_url()` recupera l'[URL](#) e lo fornisce al proprio sito, più un percorso opzionale come un foglio di stile o un'immagine. Questa funzione è accessibile attraverso la corrispondente funzione nell'Helper URL (sezione [9.20 a pagina 422](#)).
- `$this->config->system_url()` recupera l'[URL](#) e lo fornisce alla directory di sistema.

## 8.6 CLASSE DATABASE

CodeIgniter è dotato di una classe di database completa, molto veloce ed astratta: supporta sia le strutture tradizionali che i pattern Active Record.

## INTRODUZIONE

Qui di seguito verranno descritti alcuni esempi su come viene utilizzata la classe Database. Per informazioni dettagliate si rimanda alla lettura delle singole sezioni che descrivono ogni funzione.

## QUERY STANDARD CON RISULTATI MULTIPLI (OBJECT)

Verrà mostrato un codice di una query standard con risultati multipli (versione Object). La funzione **result()** restituisce un array di oggetti. Esempio **\$row->title**.

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

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

echo 'Totale dei risultati: ' . $query->num_rows();
```

## QUERY STANDARD CON RISULTATI MULTIPLI (ARRAY)

La query standard con risultati multipli, ma che utilizza l'array per i propri dati, prevede una sintassi leggermente differente. In questo caso la funzione **result\_array()** restituisce un array di indici di array, come per esempio **\$row['title']**:

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

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

Quando si esegue una query solitamente non vengono prodotti risultati su schermo: questo rende difficile capire se il comportamento della query sia corretto. La funzione **num\_rows()** aiuta in tal senso perché fornisce il numero di righe prodotto dalla query, ovvero i risultati ottenuti.



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

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

La query standard con un singolo risultato presenta la funzione **row()** che restituisce un oggetto:

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

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

La query standard con un singolo risultato può fare uso anche di un array. In questo caso la funzione **row\_array()** restituisce un array:

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

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

## INSERIMENTO DEI DATI IN MODALITÀ STANDARD

```
$sql = "INSERT INTO mytable (title, name)
      VALUES (". $this->db->escape($title) .", " . $this->db->escape($name) . ")";

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

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

## QUERY ACTIVE RECORD

Il pattern Active Record fornisce un sistema per semplificare le interrogazioni al database. La funzione **get()** per esempio restituisce tutti i risultati disponibili in una tabella. La classe Active Record contiene un nutrita serie di funzioni per lavorare agevolmente con un database.

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

## INSERIMENTO DEI DATI CON ACTIVE RECORD

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

## CONFIGURAZIONE DEL DATABASE

La configurazione del database avviene inserendo i dati nel file **/application/-config/database.php** come per esempio lo username e la password per stabilire una connessione con la propria base di dati. Il prototipo del file di configurazione è il seguente:

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

Il file di configurazione si basa su un array multi-dimensionale piuttosto che su uno più semplice. Il motivo di questa scelta è quello di permettere la memorizzazione di più insiemi di valori di connessione. Se per esempio si eseguono più ambienti (sviluppo, produzione, test, ecc) con un'unica installazione del framework, sarà possibile impostare un gruppo di connessione per ciascuno di essi, e in seguito passare tra i gruppi in base alle necessità. Ad esempio, per impostare un ambiente di "test" si esamini il seguente esempio:

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

Nota: l'array multidimensionale ha come valore "test".

Quindi per utilizzare questa impostazione globalmente, si modifica il parametro seguente sempre nel file config:

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

Il nome "test" è arbitrario: quindi è possibile impostare qualsiasi nome. Per impostazione predefinita si è usata la parola "default" per la connessione primaria, ma anche questo può essere rinominato in qualcosa di più rilevante per il proprio progetto.

## ACTIVE RECORD

La classe Active Record si abilita globalmente o meno agendo sul parametro di configurazione \$active\_record nel file di configurazione del database. Se non si utilizza la classe Active Record, si dovrà impostare il parametro sul valore FALSE: questo permetterà al sistema di consumare meno risorse quando le classi database saranno inizializzate. Se invece si vuole sfruttare questa peculiarità di CodeIgniter sarà sufficiente impostare il valore su TRUE:

```
$active_record = TRUE;
```

Nota: Alcune classi di CodeIgniter come per esempio Sessions, richiedono che Active Record sia abilitato per accedere ad alcune sue funzionalità.

## LISTA DEI VALORI

- hostname. L'indirizzo del server in cui si trova il database, spesso identificato con localhost o 127.0.0.1
- username. Il nome (login) con cui effettuare l'accesso
- password. La parola segreta per accedere al database
- database. Il nome del database a cui ci si vuole connettere
- dbdriver. La tipologia di database utilizzato: mysql, postgres, odbc, ecc. Il nome deve essere specificato scrivendolo in minuscolo
- dbprefix. Indica un prefisso opzionale da aggiungere al nome della tabella quando si eseguono delle query Active Record. Questo permette a diverse installazioni di CodeIgniter di condividere lo stesso database
- pconnect. Viene utilizzato per indicare delle connessioni persistenti o meno: valore booleano TRUE/FALSE

- `db_debug`. Viene utilizzato per visualizzare o meno gli errori del database: valore booleano TRUE/FALSE
- `cache_on`. Abilita o meno la cache delle query. Si veda anche la classe Database Caching
- `cachedir`. Il percorso assoluto della directory della query cache database
- `char_set`. Il set di caratteri utilizzato con il database
- `dbcollat`. Le regole di confronto tra caratteri utilizzate nella comunicazione con il database. Questo parametro per MySQL e MySQLi è usato solo come backup se il proprio server funziona con una versione del PHP inferiore alla v.5.2.3. oppure se MySQL è inferiore alla v.5.0.7 (e le query di creazione della tabella sono realizzate con DB Forge). Esiste una incompatibilità nel PHP con la funzione `mysql_real_escape_string()` che rende il proprio sito vulnerabile alle infezioni di tipo SQL se si usa un set di caratteri multi-byte e se si usa una versione inferiore di MySQL e PHP. I siti che utilizzano come set di caratteri per il database come Latin-1 o UTF-8 e collation non sono vulnerabili a questa minaccia
- `swap_pre`. Il prefisso di default della tabella che può essere scambiato con **dbprefix**. Questo parametro è utile per le applicazioni distribuite in cui le query devono essere scritte manualmente e devono avere un prefisso personalizzabile dall'utente finale
- `autoinit`. Con questo parametro si stabilisce se si deve impostare una connessione automatica al database quando viene caricata la libreria. Se è viene utilizzato il valore FALSE la connessione avverrà prima di eseguire la prima query
- `stricton`. Stabilisce se forzare la connessione "Strict Mode", consigliato per garantire il rigoroso rispetto di SQL durante lo sviluppo di un'applicazione
- `port`. Il numero di porta della base di dati: si utilizza aggiungendo una linea nel file di configurazione relativa all'array del database

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

A seconda di quale piattaforma di database si utilizza (MySQL, Postgres, ecc) non saranno necessari tutti i valori precedentemente indicati. Ad esempio, quando si utilizza SQLite non si dovrà fornire un nome utente o la password, e il nome del database sarà il percorso del file di database. Le informazioni di cui sopra si riferiscono ad un sistema basato su MySQL.

## INIZIAMO

Incominceremo con una introduzione sulla classe Database: spiegheremo come utilizzarla grazie a degli esempi speriamo comprensibili.

Per caricare e inizializzare il database, esistono due strade: connessione automatica oppure manuale. La prima permette di caricare e istanziare la classe database per ogni pagina caricata. È sufficiente aggiungere la parola database all'array library nel file **/application/config/autoload.php**. Se invece le pagine che richiedono una connessione alla base dei dati sono limitate, si può optare per una connessione manuale. Sulla base dei parametri forniti (si veda [2.2 a pagina 15](#)) si utilizzerà l'istruzione:

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

Nota: Se la funzione di cui sopra non contiene alcuna informazione nel primo parametro, essa si collegherà al gruppo specificato nel file di configurazione del database. Per molte persone, questo è il metodo preferito di utilizzo.

## PARAMETRI DISPONIBILI

1. i valori di connessione del database passate tramite un array o una stringa Domain Name System ([DNS](#))
2. TRUE/FALSE. Per restituire l'ID di collegamento (si veda nella pagina successiva)
3. TRUE/FALSE. Per abilitare la classe Active Record. Per default questo valore è TRUE

## CONNESSIONE MANUALE

Il primo parametro di questa funzione può opzionalmente essere utilizzato per specificare un particolare gruppo di database dal file di configurazione, oppure si possono anche inviare i valori di connessione per un database che non è specificato nel file di configurazione. Esempi:

Per scegliere uno specifico gruppo dal file di configurazione si può utilizzare la seguente istruzione, dove `group_name` è il nome del gruppo di connessione specificato nel file config.

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

Per connettersi manualmente al database si può passare un array con i valori:

```
$config['hostname'] = "localhost";
$config['username'] = "myusername";
$config['password'] = "mypassword";
$config['database'] = "mydatabase";
$config['dbdriver'] = "mysql";
$config['dbprefix'] = "";
$config['pconnect'] = FALSE;
$config['db_debug'] = TRUE;
$config['cache_on'] = FALSE;
$config['cachedir'] = "";
$config['char_set'] = "utf8";
$config['dbcollat'] = "utf8_general_ci";

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

Oppure si possono inviare i valori del proprio database come un [DNS](#) caratterizzato da questo prototipo:

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

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

Per sovrascrivere i valori di config di default quando ci si connette con una stringa DNS, è necessario aggiungere le variabili config come una stringa query:

```
$dsn = 'dbdriver://username:password@hostname/database?char_set=utf8&dbcollat=
utf8_general_ci&cache_on=true&cachedir=/path/to/cache';

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

## CONNESSIONE A PIÙ DATABASE

La connessione con database multipli è possibile grazie ad una serie di istruzioni come:

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

Dove `group_one` e `group_two` indicano gli specifici gruppi a cui ci si desidera connettere. Il secondo parametro, nell'esempio impostato su `TRUE`, fa sì che la funzione restituisca un database object. Quando ci si collega in questo modo si

utilizza il nome dell'oggetto piuttosto che la sintassi utilizzata in questa guida. In altre parole, invece di impartire comandi con:

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

si utilizza:

```
$DB1->query();  
$DB1->result();
```

## MANTENERE LA CONNESSIONE

Per mantenere la connessione “viva” prima che si verifichi un timeout (per esempio a causa di istruzioni [PHP](#) che richiedono molto tempo) è possibile utilizzare la funzione **connect()** che effettua un ping del server, prima di inviare ulteriori query al server:

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

## CHIUDERE UNA CONNESSIONE

Una connessione può invece essere chiusa esplicitamente attraverso il seguente comando. Solitamente CodeIgniter si occupa dell'operazione autonomamente.

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



## QUERY

In questa sezione verranno trattate le query ovvero quelle istruzioni che permettono di “interrogare” i database per ottenere/modificare una grande mole di informazioni.

`$this->db->query()` è l’istruzione basilare per eseguire una query. La sintassi dell’istruzione è:

```
$this->db->query('INSERISCI QUI LA TUA QUERY');
```

La funzione **query()** restituisce un oggetto quando vengono eseguite le query di tipo read, utilizzate per visualizzare i risultati (per esempio **SELECT**). Quando invece vengono eseguite le query di tipo write, la funzione semplicemente restituisce il valore booleano TRUE oppure FALSE, a seconda del successo o meno dell’operazione. Quando si recuperano i dati, tipicamente si può assegnare la query alla propria variabile come nel seguente:

```
$query = $this->db->query('INSERISCI QUI LA TUA QUERY');
```

`$this->db->simple_query()` è la versione semplificata della funzione `$this->db->query()`. Essa restituisce solo i valori TRUE oppure FALSE a seconda del successo o meno della query. Questa funzione perciò non restituisce una lista di risultati, né imposta le query timer, compila i dati bind, o memorizza le proprie query per il debug: viene semplicemente utilizzata per inviare/sottoporre una query. Questa funzione viene implementata raramente.

## PREFISSO DEL DATABASE

Se si è configurato manualmente un prefisso per il database e lo si vuole anteporre ad un nome di tabella per utilizzarlo per esempio in una query nativa SQL, è possibile utilizzare il seguente comando:

```
$this->db->dbprefix('tablename');  
// restituisce prefisso_tablename
```

Se per qualsiasi ragione è necessario cambiare il prefisso, senza creare una nuova connessione, si può adoperare la funzione:

```
$this->db->set_dbprefix('newprefix');  
  
$this->db->dbprefix('tablename');  
// restituisce nuovoprefisso_tablename
```

## PROTEGGERE GLI IDENTIFICATORI

In molti database, è consigliabile proteggere la tabella e i record, per esempio con i backtick in MySQL (si veda la definizione in questa pagina). Con Active Record, le query sono automaticamente protette, tuttavia se si ha bisogno di proteggere manualmente un identificatore è possibile utilizzare:

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

### Definizione 11: Backtick

---

**Backtick** L'operatore di esecuzione backtick è costituito dagli apici inversi (""). Si noti che non si tratta di doppi apici o apostrofi. L'istruzione contenuta all'interno dei backtick verrà eseguito come se fosse un comando di shell: il risultato dovrà essere assegnato ad una variabile e visualizzato eventualmente con il comando echo. L'uso dell'operatore backtick è identico alla funzione `shell_exec()` e diversamente da altri linguaggi non possono essere utilizzati all'interno di stringhe delimitate da doppi apici. Per finire, l'operatore backtick viene disabilitato quando la modalità sicura è attiva oppure se `shell_exec()` è disabilitata.

---

Questa funzione si può anche aggiungere al prefisso della propria tabella, ipotizzando che un prefisso sia indicato nel proprio file di configurazione. Per abilitare l'uso del prefisso è necessario impostare su TRUE il secondo parametro:

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

## ESCAPE DELLE QUERY

È una buona pratica di sicurezza effettuare l'escape dei dati prima di inviarli al proprio database. CodeIgniter ha tre metodi utili a tale scopo.

- `$this->db->escape()` questa funzione determina il tipo di dati in modo che si possa fare l'escape solo dei dati stringa. Essa aggiunge le singole virgolette attorno al dato:

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

- `$this->db->escape_str()` questa funzione effettua l'escape dei dati che gli vengono passati indipendentemente dal tipo:

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

- `$this->db->escape_like_str()` questa funzione deve essere usata quando le stringhe contengono la condizione LIKE e i wildcard ( %, \_).

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

## QUERY BINDING

I Binding consentono di semplificare la sintassi di query lasciando che il sistema inserisca le query per noi. Si consideri il seguente esempio:

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

I “punto di domanda” nella query vengono sostituiti automaticamente con i valori presenti nel secondo parametro dell'array della funzione `query()`. Il secondo vantaggio nell'utilizzare Bind è che i valori sono automaticamente controllati, permettendo così query più sicure. In pratica, non ci si deve più preoccupare di effettuare l'escape manualmente perché sarà l'engine di CodeIgniter a farlo automaticamente.

## I RISULTATI DELLE QUERY

Ci sono diverse strade per generare i risultati delle query: qui analizzeremo tutte le funzioni messe a disposizione da CodeIgniter:

- `result()` questa funzione restituisce il risultato di una query come un array di oggetti o un array vuoto in caso di fallimento; solitamente viene utilizzata in un loop `foreach` come il seguente:

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

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

La funzione è un alias di `result_object()`. Se si esegue una query che potrebbe non produrre il risultato, è consigliabile verificare prima il risultato:

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

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

Si può passare una stringa alla funzione `result()` che rappresenta una classe per istanziare ogni risultato oggetto (nota: questa classe deve essere caricata):

```
$query = $this->db->query("SELECT * FROM users;");

foreach ($query->result('User') as $row)
{
    echo $row->name; // chiama gli attributi
    echo $row->reverse_name(); // o i metodi definiti nella classe User
}
```

- `result_array()` questa funzione restituisce il risultato della query come un array o un array vuoto quando non viene prodotto alcun risultato. Solitamente questa funzione viene utilizzata in un loop foreach come:

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

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

- `row()` restituisce come risultato una singola riga. Se la query ha più di una riga, sarà processata solo la prima. Il risultato è restituito con un oggetto. Un esempio:

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

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

    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

Se si desidera che venga restituita una specifica riga tra i vari risultati, è possibile indicarla nel primo parametro sotto forma di numero intero:

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

È anche possibile aggiungere un secondo parametro con una stringa, ovvero il nome della classe per istanziare la riga:

```
$query = $this->db->query("SELECT * FROM users LIMIT 1;");

$query->row(0, 'User')
echo $row->name; // chiama gli attributi
echo $row->reverse_name(); // o i metodi definiti nella classe User
```

- `row_array()` è una funzione simile a `row()` eccetto per il fatto che restituisce un array. Per esempio:

```
$query = $this->db->query("LA TUA QUERY");  
  
if ($query->num_rows() > 0)  
{  
    $row = $query->row_array();  
  
    echo $row['title'];  
    echo $row['name'];  
    echo $row['body'];  
}
```

Se si desidera che venga restituita una specifica riga, è possibile specificarne il numero nel primo parametro:

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

In aggiunta ci si può spostare tra i risultati in avanti/indietro/all'inizio/alla fine utilizzando questi metodi:

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

Per impostazione predefinita questi metodi restituiscono un oggetto a meno di mettere la parola "array" nel parametro:

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

## FUNZIONI HELPER: I RISULTATI

- `$query->num_rows()` fornisce il numero di righe restituite dalla query. In questo esempio, `$query` è la variabile a cui viene assegnato il risultato dell'interrogazione:

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

- `$query->num_fields()` viene restituito il numero di colonne (attributi) della query. È necessario essere sicuri di chiamare la funzione utilizzando il risultato della query memorizzato in **\$query**:

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

- `$query->free_result()` questa funzione libera la memoria associata al risultato e cancella il risultato della risorsa individuata dall'ID. Normalmente [PHP](#) libera automaticamente la memoria al termine dell'esecuzione dello script, tuttavia, se si eseguono numerose interrogazioni in un particolare script, è consigliabile eliminare subito questi risultati. Per esempio:

```
$query = $this->db->query('SELECT title FROM my_table');  
  
foreach ($query->result() as $row)  
{  
    echo $row->title;  
}  
$query->free_result(); // L'oggetto $query non è ùpi disponibile  
  
$query2 = $this->db->query('SELECT name FROM some_table');  
  
$row = $query2->row();  
echo $row->name;  
$query2->free_result(); // L'oggetto $query2 non è ora disponibile
```

## FUNZIONI HELPER QUERY

- `$this->db->insert_id()` il numero ID di inserimento quando si esegue un record viene aggiunto nel database.
- `$this->db->affected_rows()` visualizza il numero di righe interessate quando vengono processate le query di tipo write (INSERT, UPDATE, ecc.).

Nota: in MySQL DELETE FROM TABLE restituisce o (zero) righe. La classe di database ha un piccolo hack che permette di restituire il giusto numero di righe interessate. Per impostazione predefinita, questo hack è abilitato, ma può essere disattivato nel file driver di database

- `$this->db->count_all()` permette di determinare il numero di righe in una specifica tabella. Si deve indicare il nome della tabella come primo parametro:

```
echo $this->db->count_all('my_table');  
  
// Produce un intero come 25
```

- `$this->db->platform()` visualizza il tipo di database utilizzato (MySQL, MS SQL, Postgres, etc).

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

- `$this->db->version()` visualizza la versione del database utilizzato:

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

- `$this->db->last_query()` restituisce l'ultima query eseguita (la stringa query, non il risultato):

```
$str = $this->db->last_query();  
  
// Produce: SELECT * FROM qualche tabella...
```

- `$this->db->insert_string()` questa funzione semplifica il processo di scrittura degli inserimenti nel database. Viene restituita una stringa di inserimento nel formato SQL.



```
$data = array('name' => $name, 'email' => $email, 'url' => $url);  
$str = $this->db->insert_string('table_name', $data);
```

Il primo parametro è il nome della tabella, il secondo è un array associativo che contiene i dati da inserire. L'esempio di qui sopra produce:

```
INSERT INTO table_name (name, email, url) VALUES ('Rick', 'rick@example.  
com', 'example.com')
```

Nota: i valori subiscono automaticamente l'escape, aspetto che rende le query sicure.

- `$this->db->update_string()` questa funzione semplifica l'aggiornamento del database. Viene restituita un stringa di aggiornamento formattata SQL. Per esempio:

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

Il primo parametro è il nome della tabella, mentre il secondo è un array associativo con i dati da aggiornare. Il terzo parametro è la clausola WHERE. L'esempio di qui sopra produce:

```
UPDATE table_name SET name = 'Rick', email = 'rick@example.com', url = '  
example.com' WHERE author_id = 1 AND status = 'active'
```

Nota: i valori subiscono automaticamente l'escape, aspetto che rende le query sicure.

## 8.7 CLASSE ACTIVE RECORD

CodeIgniter usa una versione modificata del pattern Active Record Database per agevolare il recupero, l'inserimento e la modifica delle informazioni attraverso i metodi di una specifica classe. In molti casi sono necessarie solo una o due linee di codice per eseguire una operazione sul database. CodeIgniter non richiede che ogni tabella della base di dati abbia il proprio file di classe poiché fornisce un'interfaccia maggiormente semplificata. Al di là della semplicità, un grande vantaggio nell'utilizzare le funzionalità Active Record è la possibilità di creare applicazioni indipendenti dal database utilizzato, attraverso l'astrazione. Ciò consente inoltre di generare query più sicure, in quanto l'escape sui valori viene svolto automaticamente dal sistema. Se comunque si intende scrivere la propria query autonomamente, si può tranquillamente disabilitare questa classe attraverso il file di configurazione del database. Questo permette alla libreria core del database, tra le altre cose, di consumare meno risorse.

---

### Definizione 12: Active Record

---

#### Active Record

Si tratta di un modulo implementato seguendo il pattern descritto da Martin Fowler (<http://www.martinfowler.com/>) nel suo libro del 2003: *Patterns of enterprise application architecture*. Addison-Wesley. ISBN 978-0-321-12742-6. Secondo questo pattern esiste una relazione molto stretta fra tabella e classe, colonne e attributi della classe. Ecco alcuni punti importanti:

- una tabella del database relazionale è gestita con una classe
  - una riga della tabella può essere definita da una singola istanza della classe
  - una nuova istanza della classe crea una nuova riga all'interno della tabella, mentre modificare l'istanza produce un aggiornamento della riga
- 

---

### Definizione 13: Persistenza

---

#### Persistenza

Una base di dati è persistente, cioè ha un tempo di vita superiore al tempo di esecuzione delle applicazioni che la utilizzano.

---

## ACTIVE RECORD: I PRINCIPI

Le colonne delle tabelle sono rappresentate come attributi della classe: ogni istanza quindi può effettuare qualsiasi operazione sui valori dei propri attributi e renderli persistenti. In pratica, creando un'istanza di un oggetto (che rappresenta un record) si possono modificare gli attributi (i campi o le colonne) e rendere persistente i dati (con le istruzioni INSERT o UPDATE), lasciando che Active Record faccia il resto.

Le convenzioni sui nomi delle tabelle, classi, colonne e attributi permettono di semplificare la produzione di codice SQL, partendo dalla definizione del modello:

- per ogni modello, una tabella possiede lo stesso nome al plurale e scritto tutto in minuscolo. Se il modello si chiama Blog, la tabella prenderà il nome di blogs
- i nomi dei modelli composti da più parole sono rappresentati con rappresentazione CamelCase (l'iniziale maiuscola); la tabella corrispondente avrà un nome separato da trattini bassi (underscore)
- ogni colonna della tabella è rappresentato da un attributo con lo stesso nome
- in ogni tabella vi è un identificatore univoco rappresentato da una colonna id (intero positivo)

## SELEZIONE

Le seguenti istruzioni si basano sull'istruzione SELECT dell'SQL, per selezionare parte o tutte le informazioni contenute nella base di dati.

Nota: se si sta utilizzando PHP 5 è possibile utilizzare il metodo di concatenamento per una sintassi più compatta. Maggiori informazioni sono disponibili alla fine di questo capitolo.

- `$this->db->get()` esegue la query di selezione e restituisce il risultato. Può essere utilizzata per recuperare tutti i record da una tabella:

```
$query = $this->db->get('mytable');  
  
// Produce: SELECT * FROM mytable
```

Il secondo e il terzo parametro del codice seguente limitano la lista dei risultati:

```
$query = $this->db->get('mytable', 10, 20);  
  
// Produce: SELECT * FROM mytable LIMIT 20, 10 (questo in MySQL. Altri  
database hanno una sintassi diversa)
```

Si noterà che la funzione di cui sopra viene assegnata ad una variabile denominata `$query`, che può essere utilizzata per mostrare i risultati:

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

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

Per maggiori informazioni si rimanda alla funzione dei risultati.

- `$this->db->get_where()` è identica alla funzione sopra descritta: consente di aggiungere la condizione WHERE nel secondo parametro, invece di usare la funzione `db->where()`:

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

Nota: `get_where()` era precedentemente nota come `getwhere()`, è ora stata rimossa.

- `$this->db->select()` permette di scrivere la parte SELECT della query:

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

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

// Produce: SELECT title, content, date FROM mytable
```

Nota: se si selezionano tutti i dati (\*) di una tabella non è necessario utilizzare questa funzione. Se omissso, CodeIgniter presuppone che si desidera selezionare \*. La funzione accetta un secondo parametro opzionale: se lo si imposta su FALSE, CodeIgniter non cercherà di proteggere i vostri campi o la tabella con apici inversi. Questo è utile se si ha bisogno di una select composta:

```
$this->db->select('(SELECT SUM(payments.amount) FROM payments WHERE
    payments.invoice_id=4') AS amount_paid', FALSE);
$query = $this->db->get('mytable');
```

- `$this->db->select_max()` scrive una parte della query "SELECT MAX(attributo)". Opzionalmente è possibile includere un secondo parametro per rinominare il campo del risultato:

```
$this->db->select_max('age');  
$query = $this->db->get('members');  
// Produce: SELECT MAX(age) as age FROM members  
  
$this->db->select_max('age', 'member_age');  
$query = $this->db->get('members');  
// Produce: SELECT MAX(age) as member_age FROM members
```

- `$this->db->select_min()` scrive una parte della query “SELECT MIN(attributo)” e come con `select_max()` è possibile includere opzionalmente un parametro per rinominare l’attributo risultante.

```
$this->db->select_min('age');  
$query = $this->db->get('members');  
// Produce: SELECT MIN(age) as age FROM members
```

- `$this->db->select_avg()` scrive una parte della query “SELECT AVG(attributo)” e come con `select_max()` è possibile includere opzionalmente un parametro per rinominare l’attributo risultante.

```
$this->db->select_avg('age');  
$query = $this->db->get('members');  
// Produce: SELECT AVG(age) as age FROM members
```

- `$this->db->select_sum()` scrive una parte della query “SELECT SUM(attributo)” e come con `select_max()` è possibile includere opzionalmente un parametro per rinominare l’attributo risultante.

```
$this->db->select_sum('age');  
$query = $this->db->get('members');  
// Produce: SELECT SUM(age) as age FROM members
```

- `$this->db->from()` scrive la parte FROM della propria query:

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

Nota: come indicato in precedenza, la parte della query FROM può anche essere specificata nella funzione `this->db->get()`: è quindi possibile utilizzare il metodo preferito.

- `$this->db->join()` permette di scrivere la parte della query JOIN:

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

Chiamate a funzioni multiple possono essere effettuate se si ha bisogno di più join in una query. Se è necessario un particolare tipo di JOIN è possibile specificarlo tramite il terzo parametro della funzione. Le opzioni sono: left, right, outer, inner, left outer, e right outer.

```
$this->db->join('comments', 'comments.id = blogs.id', 'left');  
  
// Produce: LEFT JOIN comments ON comments.id = blogs.id
```

- `$this->db->where()` questa funziona abilita la configurazione della clausola WHERE usando uno dei quattro metodi seguenti. Tutti i valori passati alla funzione effettuano automaticamente l'escape, producendo query sicure:

1. Chiave semplice/metodo valore

```
$this->db->where('name', $name);  
  
// Produce: WHERE name = 'Joe'
```

Si noti che il segno di uguaglianza viene aggiunto automaticamente. Se si utilizzano più chiamate di funzione, queste saranno concatenate insieme con la clausola AND:

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

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

2. Chiave personalizzata/metodo valore È possibile includere un operatore nel primo parametro in ordine per controllare il confronto:

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

3. Array associativo

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

È possibile includere i propri operatori usando il seguente metodo:

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

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

#### 4. Stringhe personalizzate Le clausole possono essere scritte manualmente:

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

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

L'istruzione `$this->db->where()` accetta un terzo parametro opzionale: se questo viene impostato su `FALSE`, CodeIgniter non proteggerà l'attributo o il nome della tabella con gli apici inversi.

```
$this->db->where('MATCH (field) AGAINST ("value")', NULL, FALSE);
```

- `$this->db->or_where()` questa funzione è identica a quella sopra descritta, tranne per il fatto che istanze multiple possono essere collegate tramite l'operatore OR.

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

// Produce: WHERE name != 'Joe' OR id > 50
```

Nota: `or_where()` era conosciuta precedentemente come `orWhere()`: ora quest'ultima funzione è stata rimossa.

- `$this->db->where_in()` genera una query con campo WHERE associato a IN ('elemento', 'elemento'); può essere associata anche all'operatore AND, se necessario:

```
$names = array('Frank', 'Todd', 'James');
$this->db->where_in('username', $names);
// Produce: WHERE username IN ('Frank', 'Todd', 'James')
```

- `$this->db->or_where_in()`; genera una query con campo WHERE associato a IN ('elemento', 'elemento'); può essere associata anche all'operatore OR, se necessario:



```
$names = array('Frank', 'Todd', 'James');  
$this->db->or_where_in('username', $names);  
// Produce: OR username IN ('Frank', 'Todd', 'James')
```

- `$this->db->where_not_in()` genera una query con campo WHERE associato a NOT IN ('elemento', 'elemento'); può essere associata anche all'operatore AND, se necessario:

```
$names = array('Frank', 'Todd', 'James');  
$this->db->or_where_not_in('username', $names);  
// Produce: OR username NOT IN ('Frank', 'Todd', 'James')
```

- `$this->db->like()` questa funzione abilita l'uso della clausole LIKE, utile per effettuare delle ricerche/comparazioni. Tutti i valori passati a questa funzione hanno l'escape automatico.

1. Chiave semplice/metodo valore

```
$this->db->like('title', 'match');  
  
// Produce: WHERE title LIKE '%match%'
```

Se si utilizzano chiamate a funzioni multiple, queste saranno concatenate assieme con l'operatore AND:

```
$this->db->like('title', 'match');  
$this->db->like('body', 'match');  
  
// Produce: WHERE title LIKE '%match%' AND body LIKE '%match%'
```

Se si desidera controllare dove è posto il wildcard (

```
$this->db->like('title', 'match', 'before');  
// Produce: WHERE title LIKE '%match'  
  
$this->db->like('title', 'match', 'after');  
// Produce: WHERE title LIKE 'match%'  
  
$this->db->like('title', 'match', 'both');  
// Produce: WHERE title LIKE '%match%'
```

Se non si vuole usare il wildcard (

```
$this->db->like('title', 'match', 'none');  
// Produce: WHERE title LIKE 'match'
```

## 2. Metodo array associativo

```
$array = array('title' => $match, 'page1' => $match, 'page2' =>  
    $match);  
  
$this->db->like($array);  
  
// WHERE title LIKE '%match%' AND page1 LIKE '%match%' AND page2  
    LIKE '%match%'
```

- `$this->db->or_like()` questa funzione è identica ad una già esaminata, eccetto per il fatto che istanze multiple possono essere collegate tramite l'operatore OR:

```
$this->db->like('title', 'match');  
$this->db->or_like('body', $match);  
  
// WHERE title LIKE '%match%' OR body LIKE '%match%'
```

Nota: `or_like()` era prima conosciuta come `orlike()` che è stata ora rimossa.

- `$this->db->not_like()` questa funzione è identica a `like()` eccetto per il fatto che genera una dichiarazione NOT LIKE.

```
$this->db->not_like('title', 'match');  
  
// WHERE title NOT LIKE '%match%'
```

- `$this->db->or_not_like()` è identica a `not_like()` eccetto per il fatto che istanze multiple sono collegabili tramite OR:

```
$this->db->like('title', 'match');  
$this->db->or_not_like('body', 'match');  
  
// WHERE title LIKE '%match%' OR body NOT LIKE '%match%'
```

- `$this->db->group_by()` permette di definire la parte GROUP BY della query:

```
$this->db->group_by("title");  
  
// Produce: GROUP BY title
```

È anche possibile passare un array di valori multipli come:

```
$this->db->group_by(array("title", "date"));  
  
// Produce: GROUP BY title, date
```

Nota: `group_by()` è conosciuta anche come `groupby()` ma quest'ultima è stata ora rimossa.

- `$this->db->distinct()` viene aggiunta la parola chiave DISTINCT alla query.

```
$this->db->distinct();  
$this->db->get('table');  
  
// Produce: SELECT DISTINCT * FROM table
```

- `$this->db->having()` permette di scrivere la parte HAVING della query. Per quanto riguarda la sintassi, esistono due possibilità a seconda del numero di argomenti:

```
$this->db->having('user_id = 45');  
// Produce: HAVING user_id = 45  
  
$this->db->having('user_id', 45);  
// Produce: HAVING user_id = 45
```

Se si utilizza un database per cui si effettua automaticamente l'escape delle query, è possibile evitare che il contenuto venga reso sicuro (escaping) passando un terzo argomento facoltativo con il valore FALSE.

```
$this->db->having('user_id', 45);  
// Produce: HAVING `user_id` = 45 in molti database come MySQL  
$this->db->having('user_id', 45, FALSE);  
// Produce: HAVING user_id = 45
```

- `$this->db->or_having()` è identica alla funzione `having()` dalla quale differisce per il fatto che le clausole sono separate da OR.
- `$this->db->order_by()` consente di impostare una clausola ORDER BY. Il primo parametro contiene il nome dell'attributo che si desidera ordinare. Il secondo parametro imposta la direzione del risultato: le opzioni sono asc (crescente) o desc (decrescente), o random (casuale).

```
$this->db->order_by("title", "desc");  
  
// Produce: ORDER BY title DESC
```

È possibile anche passare come primo parametro la propria stringa:

```
$this->db->order_by('title desc, name asc');  
  
// Produce: ORDER BY title DESC, name ASC
```

Oppure si possono realizzare chiamate multiple alla funzione se si ha bisogno di più campi:

```
$this->db->order_by("title", "desc");  
$this->db->order_by("name", "asc");  
  
// Produce: ORDER BY title DESC, name ASC
```

Nota: `order_by()` è conosciuto anche come `orderby()`, ma quest'ultimo è stato rimosso.

Nota: l'ordine random non è attualmente supportato dai driver Oracle o MSSQL: per questi il valore di default sarà ASC.

- `$this->db->limit()` permette di limitare il numero di righe (record) che la query restituisce:

```
$this->db->limit(10);  
  
// Produce: LIMIT 10
```

Il secondo parametro permette di settare l'offset del risultato:

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

// Produce: LIMIT 20, 10 (in MySQL. Gli altri database possono avere una
// sintassi leggermente differente)
```

- `$this->db->count_all_results()` determina il numero di righe di una particolare query Active Record. Le query accetteranno delle funzioni limitatrici come `where()`, `or_where()`, `like()`, `or_like()`, ecc. Per esempio:

```
echo $this->db->count_all_results('my_table');
// Produce un intero come 25

$this->db->like('title', 'match');
$this->db->from('my_table');
echo $this->db->count_all_results();
// Produce un intero come 17
```

- `$this->db->count_all()` determina il numero di righe di una tabella. È necessario fornire come primo parametro il nome della tabella. Ad esempio:

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

// Produce un intero come 25
```

## INSERIMENTO

- `$this->db->insert()` genera una stringa di inserimento basata sui dati che si forniscono, e quindi esegue la query. È possibile passare alla funzione un array oppure un oggetto; in questo esempio, verrà fornito un array:

```
$data = array(  
    'title' => 'Il mio titolo' ,  
    'name' => 'Il mio nome' ,  
    'date' => 'La mia data'  
);  
  
$this->db->insert('mytable', $data);  
  
// Produce: INSERT INTO mytable (title, name, date) VALUES ('Il mio  
    titolo', 'Il mio nome', 'La mia data')
```

Il primo parametro conterrà il nome della tabella, mentre il secondo è un array associativo di valori. Qui di seguito l'esempio che utilizza un oggetto:

```
/*  
    class Myclass {  
        var $title = 'Il mio titolo';  
        var $content = 'Il mio contenuto';  
        var $date = 'La mia data';  
    }  
*/  
  
$object = new Myclass;  
  
$this->db->insert('mytable', $object);  
  
// Produce: INSERT INTO mytable (title, content, date) VALUES ('Il mio  
    titolo', 'Il mio contenuto', 'La mia data')
```

Il primo parametro conterrà il nome della tabella, mentre il secondo è un oggetto. Tutti i valori sono resi sicuri (escaping) automaticamente, cosa che produce query prive di rischi.

- `$this->db->insert_batch()` genera una stringa di inserimento basata sui dati che verranno forniti, quindi esegue la query. È possibile passare alla funzione un array o un oggetto. Qui di seguito si mostra un esempio utilizzando un array: il primo parametro contiene il nome della tabella, mentre il secondo è un array associativo di valori:

```
$data = array(
    array(
        'title' => 'Il mio titolo' ,
        'name' => 'Il mio nome' ,
        'date' => 'La mia data'
    ),
    array(
        'title' => 'Un'altro titolo' ,
        'name' => 'Un altro nome' ,
        'date' => 'Un'altra data'
    )
);

$this->db->insert_batch('mytable', $data);

// Produce: INSERT INTO mytable (title, name, date) VALUES ('Il mio
           titolo', 'Il mio nome', 'La mia data'), ('Un'altro titolo', 'Un altro
           nome', 'Un'altra data')
```

Tutti i valori sono resi sicuri (escaping) automaticamente, cosa che produce query pulite.

- `$this->db->set()` questa funzione permette di impostare i valori per gli inserimenti o gli aggiornamenti. Può essere usata al posto di un array di dati. Per esempio:

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

// Produce: INSERT INTO mytable (name) VALUES ('{$name}')
```

Se si utilizzano più chiamate di funzione, queste saranno organizzate correttamente a seconda che si stia facendo un inserimento o un aggiornamento:

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

La funzione **set()** accetta un terzo parametro opzionale (`$escape`): se settato su `FALSE` non verrà effettuato l'escaping dei caratteri. Per meglio comprendere la differenza, qui di seguito si mostra **set()** con e senza il parametro di escape:

```
$this->db->set('field', 'field+1', FALSE);  
$this->db->insert('mytable');  
// Produce: INSERT INTO mytable (field) VALUES (field+1)  
  
$this->db->set('field', 'field+1');  
$this->db->insert('mytable');  
// Produce INSERT INTO mytable (field) VALUES ('field+1')
```

È possibile passare anche un array associativo alla funzione:

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

oppure un oggetto:

```
/*  
    class Myclass {  
        var $title = 'Il mio titolo';  
        var $content = 'Il mio contenuto';  
        var $date = 'La mia data';  
    }  
*/  
  
$object = new Myclass;  
  
$this->db->set($object);  
$this->db->insert('mytable');
```

## AGGIORNAMENTO

- `$this->db->update()` questa funzione genera e aggiorna la stringa quindi esegue la query sulla base dei dati forniti. Si può passare alla funzione un array oppure un oggetto. Per esempio, utilizzando un array si avrà:



```
$data = array(
    'title' => $title,
    'name' => $name,
    'date' => $date
);

$this->db->where('id', $id);
$this->db->update('mytable', $data);

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

oppure inviando un oggetto:

```
/*
    class Myclass {
        var $title = 'Il mio titolo';
        var $content = 'Il mio contenuto';
        var $date = 'La mia data';
    }
*/

$object = new Myclass;

$this->db->where('id', $id);
$this->db->update('mytable', $object);

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

Nota: viene effettuato l'escape di tutti i valori, cosa che rende le query sicure.

Si noterà che l'uso dell'istruzione `$this->db->where()` consente di impostare la clausola WHERE. Opzionalmente si possono passare queste informazioni direttamente alla funzione di aggiornamento sotto forma di una stringa:

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

oppure come un array:

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

È inoltre possibile utilizzare la funzione `$this->db->set()` descritta in precedenza nelle operazioni che riguardano gli aggiornamenti.

- `$this->db->update_batch()` questa funzione genera e aggiorna la stringa ed esegue la query sulla base dei dati forniti. Si può passare alla funzione un array oppure un oggetto. Per esempio, utilizzando un array si avrà:

```
$data = array(
    array(
        'title' => 'Il mio titolo' ,
        'name' => 'Il mio nome 2' ,
        'date' => 'La mia data 2'
    ),
    array(
        'title' => 'Un'altro titolo' ,
        'name' => 'Un altro nome 2' ,
        'date' => 'Un'altra data 2'
    )
);

$this->db->update_batch('mytable', $data, 'title');

// Produce:
// UPDATE `mytable` SET `name` = CASE
// WHEN `title` = 'Il mio titolo' THEN 'Il mio nome 2'
// WHEN `title` = 'Un'altro titolo' THEN 'Un altro nome 2'
// ELSE `name` END,
// `date` = CASE
// WHEN `title` = 'Il mio titolo' THEN 'La mia data 2'
// WHEN `title` = 'Un'altro titolo' THEN 'Un'altra data 2'
// ELSE `date` END
// WHERE `title` IN ('Il mio titolo','Un'altro titolo')
```

Il primo parametro è il nome della tabella, il secondo è un array associativo di valori, mentre il terzo è la chiave WHERE.

Nota: viene effettuato l'escape di tutti i valori, cosa che rende le query sicure.

## CANCELLAZIONE

- `$this->db->delete()` genera e cancella la stringa SQL ed esegue la query.

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

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

Il primo parametro è il nome della tabella, il secondo è la clausola WHERE. É possibile anche utilizzare le funzioni `where()` oppure `or_where()` invece di passare i dati al secondo parametro della funzione.

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

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

Un array di nomi di tabella possono essere passati alla funzione `delete()` se si desiderano cancellare i dati provenienti da più di una tabella.

```
$tables = array('table1', 'table2', 'table3');
$this->db->where('id', '5');
$this->db->delete($tables);
```

Se si desiderano cancellare tutti i dati da una tabella, allora è consigliabile utilizzare la funzione `truncate()` oppure `empty_table()`.

- `$this->db->empty_table()` genera e cancella la stringa SQL ed esegue la query.

```
$this->db->empty_table('mytable');

// Produce
// DELETE FROM mytable
```

- `$this->db->truncate()` genera e tronca la stringa SQL quindi esegue la query.

```
$this->db->from('mytable');  
$this->db->truncate();  
// or  
$this->db->truncate('mytable');  
  
// Produce:  
// TRUNCATE mytable
```

Nota: se il comando TRUNCATE non è disponibile, allora `truncate()` sarà eseguita come “DELETE FROM table”.

## METODO DI CONCATENAMENTO

Questo metodo (che funziona solo con PHP 5) permette di semplificare la sintassi collegando più funzioni. Si consideri il seguente esempio:

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

## ACTIVE RECORD CACHING

Anche se non si tratta di un caching “vero”, Active Record permette di salvare (o meglio memorizzare nella cache) alcune parti delle query per riutilizzarle successivamente quando il proprio script verrà eseguito nuovamente. Normalmente quando una chiamata Active Record viene eseguita, tutte le informazioni memorizzate sono resettate per la chiamata successiva. Con la memorizzazione nella cache è possibile evitare questo ripristino, e riutilizzare facilmente le informazioni. Le chiamate cache sono cumulative: se vengono fatte due chiamate **select()** (memorizzate nella cache) e successivamente due chiamate **select()** (non memorizzate nella cache), questo si tradurrà in quattro chiamate **select()**. Ci sono tre funzioni caching disponibili:

- `$this->db->start_cache()`
- `$this->db->stop_cache()`
- `$this->db->flush_cache()`

```
$this->db->start_cache();
$this->db->select('field1');
$this->db->stop_cache();

$this->db->get('tablename');

//Genera: SELECT `field1` FROM (`tablename`)

$this->db->select('field2');
$this->db->get('tablename');

//Genera: SELECT `field1`, `field2` FROM (`tablename`)

$this->db->flush_cache();

$this->db->select('field2');
$this->db->get('tablename');

//Genera: SELECT `field2` FROM (`tablename`)
```

Nota: le seguenti istruzioni possono essere memorizzate nella cache: select, from, join, where, like, group\_by, having, order\_by, set.

## TRANSAZIONI

L'astrazione del database di CodeIgniter permette di utilizzare le transazioni con tutti quei database che supportano i tipi di tabelle transaction-safe: in MySQL, è necessario utilizzare le tabelle InnoDB o BDB piuttosto che il più comune MyISAM, mentre la maggior parte delle altre piattaforme di database supportano le operazioni di transazione nativamente.

Se non si ha familiarità con l'argomento trattato si consiglia di trovare una buona risorsa online per apprendere a sfruttarne le potenzialità. Le informazioni qui di seguito riportate, infatti presuppongono una conoscenza di base sull'argomento, comunque non complesso e molto utile in diversi contesti. CodeIgniter utilizza un approccio alle transazioni che è molto simile a quello della classe database ADODB. È stato scelto questo approccio perché semplifica notevolmente il processo di funzionamento delle transazioni: nella maggior parte dei casi tutto ciò che serve sono due righe di codice.

Le transazioni hanno richiesto una buona dose di lavoro per essere implementate poiché tengono traccia delle query e determinano cosa debba essere sottoposto a commit o rollback in base al successo o il fallimento delle proprie query. Ciò è particolarmente difficile da gestire con le query nidificate: per questo CodeIgniter implementa un sistema di transazione intelligente che svolge automaticamente le operazioni senza l'intervento dello sviluppatore (è possibile anche gestire le transazioni manualmente se lo si preferisce, ma questo non comporterà alcun beneficio).

### Definizione 14: Transazione

---

**Transazione**

È una sequenza di operazioni normalmente implementate da un DBMS o da un gestore di transazioni che ha il compito di processare una sequenza di operazioni. Queste possono generare un errore (per un bug, crash del server, operazioni illecite), oppure concludersi con successo. Nel primo caso verrà eseguita una istruzione di **rollback** che abortisce la transazione, mentre in caso di successo viene eseguita l'istruzione **commit** che conferma la transazione.

---

## ESEGUIRE LE TRANSAZIONI

Per eseguire le proprie query usando le transazioni si utilizzeranno due funzioni di inizio e fine sequenza:

- `$this->db->trans_start()`
- `$this->db->trans_complete()`

```
$this->db->trans_start();  
    $this->db->query('UNA QUERY SQL...');  
$this->db->query('ALTRA QUERY...');  
$this->db->query('E...ANCORA UNA QUERY...');  
$this->db->trans_complete();
```

É possibile eseguire diverse query tra le funzioni start/complete su cui si potrà avere un commit o un rollback in base al successo o meno di una determinata query.

## MODALITÀ STRICT

CodeIgniter, per impostazione predefinita, esegue le transazioni in modalità Strict: se si eseguono più gruppi di transazioni, ed una di queste fallisce, allora tutti i gruppi subiscono un rollback. Se invece la modalità Strict è disabilitata, ogni gruppo viene gestito indipendentemente e il fallimento di uno non avrà conseguenze sugli altri gruppi. La modalità strict può essere disabilitata nel seguente modo:

```
$this->db->trans_strict(FALSE);
```

## GESTIONE DEGLI ERRORI

Se si è impostato il file `/config/database.php` per avere i report sugli errori, allora sarà possibile visualizzare i messaggi di errore quando un commit non viene correttamente eseguito. Se il debug invece è disabilitato, gli errori verranno gestiti nel seguente modo:

```
$this->db->trans_start();  
$this->db->query('UNA QUERY...');  
$this->db->query('ALTRA QUERY...');  
$this->db->trans_complete();  
  
if ($this->db->trans_status() === FALSE)  
{  
    // genera un errore, oppure usare la funzione log_message() per i log dei  
    propri errori  
}
```

## ABILITARE LE TRANSAZIONI

Le transazioni sono abilitate automaticamente nel momento in cui si usa la funzione `$this->db->trans_start()`. Se si desidera invece disabilitarle, è necessario usare la funzione `$this->db->trans_off()`. Quando le transazioni sono disabilite, verrà eseguito un commit automatico delle query.

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

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

## MODALITÀ TEST

Il sistema delle transazioni si può opzionalmente impostare nella modalità test, con cui si avrà sempre un rollback delle query, anche nel caso in cui non si avrà alcun errore. Per utilizzare questa modalità, basta semplicemente impostare il primo parametro della funzione `$this->db->trans_start()` su `TRUE`:

```
$this->db->trans_start(TRUE); // Query will be rolled back
$this->db->query('UNA QUERY SQL...');
$this->db->trans_complete();
```

## ESEGUIRE LE TRANSAZIONI MANUALMENTE

L'esecuzione manuale delle transazioni si attiva mediante le seguenti funzioni, stando attenti ad adoperare correttamente `$this->db->trans_begin()` invece della funzione `$this->db->trans_start()`:



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

$this->db->query('UNA QUERY SQL...');
$this->db->query('ALTRA QUERY SQL...');
$this->db->query('E INFINE UNA QUERY SQL...');

if ($this->db->trans_status() === FALSE)
{
    $this->db->trans_rollback();
}
else
{
    $this->db->trans_commit();
}
```

## TABELLA METADATI

Per ottenere informazioni sulle proprie tabelle, CodeIgniter fornisce due preziose funzioni:

- `$this->db->list_tables()` restituisce un array che contiene i nomi di tutte le tabelle nel database a cui si è connessi. Per esempio:

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

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

- `$this->db->table_exists()` alcune volte è di aiuto conoscere se una determinata tabella esiste prima di eseguire un'operazione su di essa. Questa funzione restituisce un valore booleano TRUE/FALSE:

```
if ($this->db->table_exists('table_name'))
{
    // altro codice...
}
```

Si sostituisca il parametro `table_name` con il nome della tabella di cui si vuole verificare l'esistenza.

## CAMPI METADATI

- `$this->db->list_fields()` restituisce un array che contiene i nomi dei campi. Questa query può essere chiamata in due modi:

1. È possibile fornire il nome della tabella e richiamarla `$this->db->`

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

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

2. I nomi dei campi associati ad una qualsiasi query che si sta eseguendo, possono essere raccolti chiamando la funzione dall'oggetto risultato della query:

```
$query = $this->db->query('SELECT * FROM some_table');  
  
foreach ($query->list_fields() as $field)  
{  
    echo $field;  
}
```

- `$this->db->field_exists()` alcune volte è di aiuto conoscere se un particolare campo esiste prima di eseguire un'azione: questa funzione restituisce un valore booleano TRUE/FALSE. Si sostituisca il campo `field_name` con il nome dell'attributo che si desidera verificare.

```
if ($this->db->field_exists('field_name', 'table_name'))  
{  
    // altro codice...  
}
```

- `$this->db->field_data()` restituisce un array di oggetti che contiene informazioni sul campo selezionato come per esempio, il tipo di attributo, la lunghezza massima del campo, ecc. Questa funzione può non essere disponibile in tutti i database:

```
$fields = $this->db->field_data('table_name');  
  
foreach ($fields as $field)  
{  
    echo $field->name;  
    echo $field->type;  
    echo $field->max_length;  
    echo $field->primary_key;  
}
```

Se si è già eseguita una query, è possibile utilizzare un oggetto (risultato della query) invece di fornire il nome della tabella.

```
$query = $this->db->query("LA TUA QUERY");  
$fields = $query->field_data();
```

I seguenti metadati sono disponibili nella funzione se supportati dal proprio database:

1. `name`: nome della colonna (attributo)
2. `max_length`: massima lunghezza della colonna
3. `primary_key`: fornisce il valore "1" se l'attributo è una chiave primaria (primary key)
4. `type`: il tipo di attributo

## CHIAMATA DELLE FUNZIONI CUSTOM

La funzione `$this->db->call_function()` abilita le possibilità di invocare metodi [PHP](#) che non sono compresi nativamente in CodeIgniter. Per esempio, se si desidera richiamare la funzione `mysql_get_client_info()` che non è nativamente supportata da CodeIgniter, è possibile scrivere:

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

È necessario fornire il nome della funzione senza il prefisso `mysql_` nel primo parametro, poiché questo viene aggiunto automaticamente. Questo accorgimento permette di eseguire le stesse funzioni in differenti piattaforme di database. Ovviamente non tutte le chiamate alle funzioni sono identiche tra le diverse piattaforme, per cui ci sono dei limiti all'utilizzo pratico di questa funzione in termini di portabilità.

Tutti i parametri necessari per la funzione che si sta chiamando saranno aggiunti al secondo parametro.

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

Spesso è necessario fornire l'ID di connessione al database o l'ID risultato del database. L'ID di connessione può essere ottenuto con la seguente istruzione:

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

Mentre l'ID risultato si recupera sotto forma di oggetto:

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

## CLASSE DATABASE CACHING

La classe Database Caching carica in una memoria temporanea (la cache) le proprie query sotto forma di file di testo, in modo da diminuire il carico di lavoro sul database.

Nota: questa classe è inizializzata automaticamente dai driver del database quando il “caching” è abilitato. Non si deve pertanto mai abilitare questa classe manualmente.

## ABILITARE IL CACHING

Vi sono tre fasi da seguire per utilizzare la funzionalità di caching:

- si crea una directory con i permessi di scrittura nel proprio server dove i file di cache saranno memorizzati
- si definisce il percorso della cartella di cache nel file **/application/config/database.php**
- si abilita la funzionalità di caching globalmente tramite il file **/application/config/database.php** oppure manualmente come descritto successivamente

Una volta eseguiti questi passi, il caching delle query avverrà automaticamente ogni volta che viene caricata una pagina contenente appunto le query di database.

## COME FUNZIONA IL CACHING

Il sistema di caching delle query di CodeIgniter avviene dinamicamente quando le pagine vengono visualizzate. Se la memorizzazione nella cache è attivata, la prima volta che viene caricata una pagina web, l’oggetto risultato della query viene serializzato (si veda la definizione 15) e memorizzato in un file di testo sul server: la prossima volta che la pagina verrà nuovamente caricata, il file di cache verrà utilizzato senza dover accedere nuovamente al database. In questo modo l’utilizzo del database può essere efficacemente ridotto a zero per tutte quelle le pagine memorizzate precedentemente nella cache.

### Definizione 15: Serializzare

---

**Serializzare**

Indica quell’operazione che genera una versione archiviabile di un valore. Viene utilizzata sovente per memorizzare o passare valori a PHP senza perderne la struttura e il tipo.

---

Solo le query tipo lettura (SELECT) possono essere memorizzate nella cache, poiché queste sono le uniche che producono un risultato. Le query di tipo scrittura

(INSERT, UPDATE, ecc) poiché non generano un risultato, non verranno memorizzate nella cache. I file di cache inoltre non scadono: tutte le query che sono state memorizzate, rimarranno tali fino a quando non le si elimina. Il sistema consente comunque di pulire la cache associata alle singole pagine, oppure di eliminare l'intera collezione di file di cache. Solitamente le funzioni di pulizia descritte di seguito sono utilizzate per eliminare i file della cache solo dopo determinati eventi, come quando si aggiungono nuove informazioni al database.

## PRESTAZIONI

L'aumento delle prestazioni utilizzando il caching dipende da molti fattori legati al proprio database. Se il progetto sviluppato si basa su una base di dati altamente ottimizzata in cui le interrogazioni sono ridotte, allora il sistema di caching non porterà rilevanti vantaggi. Al contrario se il database è soggetto a molte richieste, i pregi di questa soluzione saranno tangibili. Si tenga sempre presente che il caching cambia semplicemente il modo in cui le informazioni vengono recuperate, trasformando un'operazione sul database in una sul file-system. In alcuni ambienti server-cluster, per esempio, la cache può essere dannosa in quanto le operazioni sul file system possono divenire molto intense. Su singoli server in ambienti condivisi, il caching sarà probabilmente utile. Purtroppo non esiste una risposta semplice alla domanda se è opportuno utilizzare la cache, poiché questa scelta dipende dall'ambiente in cui lavora il server e dalla natura del progetto sviluppato.

CodeIgniter memorizza il risultato di ogni query nel proprio specifico file di cache. Insieme ai file di cache sono ulteriormente organizzati in sotto-cartelle corrispondenti alle funzioni del controller. Per essere precisi, le sotto-cartelle sono denominate in modo identico ai primi due segmenti dell'[URI](#) (il nome della classe controller e il nome della funzione). Ad esempio, supponiamo di avere un controller chiamato "blog" con una funzione chiamata "commenti" che contiene tre query. Il sistema di caching creerà una cartella di cache denominata **blog+commenti**, nella quale scriverà tre file di cache. Se si utilizzano le query dinamiche che cambiano in base alle informazioni dell'[URI](#) (quando si utilizza l'impaginazione, per esempio), ogni istanza della query produrrà un proprio file di cache. È possibile, quindi, ottenere più file di cache delle rispettive query che li hanno prodotti.

## GESTIRE IL FILE DI CACHE

Poiché i file di cache non scadono, sarà necessario definire delle routine di eliminazione nella propria applicazione. Ad esempio, se si considera un blog che permetta all'utente di commentare i post, ogni volta che un nuovo commento verrà memorizzato, si dovranno eliminare i file di cache associati al controller che gestisce i commenti. In caso contrario, il server si troverà con innumerevoli file inutili e capaci di limitare le prestazioni generali. Qui di seguito sono descritte due funzioni che consentono di eliminare i file.

## QUANDO IL CACHING É INUTILE

Infine, dobbiamo sottolineare che l'oggetto risultato che viene memorizzato nella cache è una versione semplificata del risultato (oggetto) completo. Per questo motivo, alcune delle funzioni di query non sono utilizzabili. Le seguenti funzioni non sono disponibili quando si utilizza un oggetto memorizzato nella cache:

- `num_fields()`
- `field_names()`
- `field_data()`
- `free_result()`

Inoltre, le due risorse del database (`result_id` e `conn_id`) non sono disponibili quando vi è il caching, poiché le risorse dei risultati si riferiscono solo alle operazioni runtime.

## REFERENCE DELLE FUNZIONI DI CACHING

- `$this->db->cache_on()` / `$this->db->cache_off()` abilita o disabilita manualmente il caching. Può rivelarsi utile quando si desidera che certe query non vengano memorizzate. Per esempio:

```
// Abilita il caching
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM mytable");

// Disabilita il caching per la prima query
$this->db->cache_off();
$query = $this->db->query("SELECT * FROM members WHERE member_id = '
    $current_user'");

// Abilita nuovamente il caching
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM another_table");
```

- `$this->db->cache_delete()` cancella i file di cache associati ad una particolare pagina: ciò è utile per esempio, dopo l'aggiornamento del database. Il sistema di caching salva i file di cache nella cartella corrispondente all'[URI](#) della pagina che si sta visualizzando. Per esempio nel caso di una pagina come **example.com/index.php/blog/commenti**, il sistema di caching salverà i propri dati in una cartella chiamata "blog+commenti". Per cancellare questo file di cache si utilizza l'istruzione:



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

Se non si usa alcun parametro, l'URI corrente sarà usato per determinare quale file cancellare.

- `$this->db->cache_delete_all()` elimina tutti i file di cache. Per esempio:

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

## CLASSE DATABASE FORGE

Si tratta di una classe che fornisce numerose funzionalità per gestire la propria base di dati, permettendo azioni come la creazione e l'eliminazione di una tabella, aggiungere una chiave o un campo.

## INIZIALIZZARE LA CLASSE

Prima di tutto, per inizializzare la classe Forge, il driver di database deve essere già in esecuzione, poiché tale classe si basa su di esso. La seguente istruzione carica la classe:

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

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

Una volta inizializzato si potrà accedere alle funzioni utilizzando il `$this->dbforge`:

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

- `$this->dbforge->create_database('db_name')` permette di creare il database specificato come primo parametro; restituisce TRUE/FALSE in base al successo o meno della funzione.

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

- `$this->dbforge->drop_database('db_name')` elimina il database specificato nel primo parametro; restituisce TRUE/FALSE in base al successo o meno della funzione.

```
if ($this->dbforge->drop_database('my_db'))  
{  
    echo 'Il database è stato cancellato!';  
}
```

## CREARE E CANCELLARE LE TABELLE

Quando si creano le tabelle, CodeIgniter fornisce un meccanismo per aggiungere campi, chiavi o modificare le colonne.

## AGGIUNGERE I CAMPI

I campi vengono creati mediante un array associativo all'interno del quale è necessario includere una chiave "type" che si riferisce al tipo di dati del campo (ad esempio, INT, VARCHAR, TEXT, ecc). Alcuni tipi di dati (ad esempio VARCHAR) richiedono anche una chiave con un vincolo (constraint).

```
$fields = array(  
    'users' => array(  
        'type' => 'VARCHAR',  
        'constraint' => '100',  
    ),  
);  
  
// sarà convertito in "users VARCHAR(100)" quando verrà aggiunto il campo.
```

Inoltre possono essere usati le seguenti chiavi/valori:

- unsigned/true: per generare UNSIGNED (senza segno) nella definizione del campo
- default/value: per generare un valore di default nella definizione del campo
- null/true: per generare NULL nella definizione del campo. Senza questo, il campo verrà impostato su NOT NULL
- auto\_increment/true: genera un flag auto\_increment sul campo. Si noti che il campo deve essere di un tipo supportato, come per esempio quello integer (intero)

```
$fields = array(  
    'blog_id' => array(  
        'type' => 'INT',  
        'constraint' => 5,  
        'unsigned' => TRUE,  
        'auto_increment' => TRUE  
    ),  
    'blog_title' => array(  
        'type' => 'VARCHAR',  
        'constraint' => '100',  
    ),  
    'blog_author' => array(  
        'type' => 'VARCHAR',  
        'constraint' => '100',  
        'default' => 'King of Town',  
    ),  
    'blog_description' => array(  
        'type' => 'TEXT',  
        'null' => TRUE,  
    ),  
);
```

Una volta definiti i campi, è possibile aggiungerli utilizzando la funzione `$this->dbforge->add_field($fields)`, seguita da una chiamata alla funzione `create_table()`.

- `$this->dbforge->add_field()` questa funzione aggiunge i campi che saranno accettati dall'array precedentemente descritto.

## PASSAGGIO DI STRINGHE

Se si sa esattamente quale campo si vuole creare, è possibile passare una stringa alla funzione `add_field()` con la corretta definizione.

```
$this->dbforge->add_field("label varchar(100) NOT NULL DEFAULT 'default label'");
```

Le chiamate multiple alla funzione `add_field()` sono cumulative.

## CREARE UN CAMPO ID

Esiste una istruzione particolare per creare i campi id. Quando si specifica un argomento "id" viene automaticamente creato un campo chiave primaria auto incrementale di tipo `INT(9)`.

```
$this->dbforge->add_field('id');  
// produce: id INT(9) NOT NULL AUTO_INCREMENT
```

## AGGIUNGERE UNA CHIAVE

Ogni tabella possiede generalmente una chiave. Questa si assegna con la funzione `$this->dbforge->add_key('field')`. Un secondo parametro opzionale impostato su `TRUE` definirà l'argomento come chiave primaria. Si noti che `add_key()` deve essere seguita da una chiamata a `create_table()`.

Eventuali colonne multiple di chiavi non primarie devono essere specificate come un array. Un esempio di output è il seguente codice MySQL:

```
$this->dbforge->add_key('blog_id', TRUE);  
// produce: PRIMARY KEY `blog_id` (`blog_id`)  
  
$this->dbforge->add_key('blog_id', TRUE);  
$this->dbforge->add_key('site_id', TRUE);  
// produce: PRIMARY KEY `blog_id_site_id` (`blog_id`, `site_id`)  
  
$this->dbforge->add_key('blog_name');  
// produce: KEY `blog_name` (`blog_name`)  
  
$this->dbforge->add_key(array('blog_name', 'blog_label'));  
// produce: KEY `blog_name_blog_label` (`blog_name`, `blog_label`)
```

## CREARE UNA TABELLA

Dopo che i campi e le chiavi sono state dichiarati è possibile creare una nuova tabella con:

```
$this->dbforge->create_table('table_name');  
// produce: CREATE TABLE table_name
```

Un secondo parametro opzionale impostato su `TRUE` aggiunge una clausola `IF NOT EXISTS` nella definizione:

```
$this->dbforge->create_table('table_name', TRUE);  
// produce: CREATE TABLE IF NOT EXISTS table_name
```

## CANCELLARE UNA TABELLA

Per eseguire una istruzione sql DROP TABLE:

```
$this->dbforge->drop_table('table_name');  
// produce: DROP TABLE IF EXISTS table_name
```

## RINOMINARE UNA TABELLA

Per rinominare una tabella:

```
$this->dbforge->rename_table('old_table_name', 'new_table_name');  
// produce: ALTER TABLE old_table_name RENAME TO new_table_name
```

## MODIFICARE LE TABELLE

- `$this->dbforge->add_column()` la funzione `add_column()` è utilizzata per modificare una tabella esistente. Essa accetta lo stesso array specificato precedentemente tramite il quale si possono aggiungere un numero infinito di campi:

```
$fields = array(  
    'preferences' => array('type' => 'TEXT')  
);  
$this->dbforge->add_column('table_name', $fields);  
  
// produce: ALTER TABLE table_name ADD preferences TEXT
```

- `$this->dbforge->drop_column()` è usata per cancellare una colonna da una tabella:

```
$this->dbforge->drop_column('table_name', 'column_to_drop');
```

- `$this->dbforge->modify_column()` l'utilizzo di questa funzione è identica a `add_column()`, tranne che modifica una colonna esistente piuttosto che aggiungerne una nuova. Per modificarne il nome è possibile aggiungere una chiave "name" nel campo che definisce l'array.

```
$fields = array(  
    'old_name' => array(  
        'name' => 'new_name',  
        'type' => 'TEXT',  
    ),  
);  
$this->dbforge->modify_column('table_name', $fields);  
  
// produce: ALTER TABLE table_name CHANGE old_name new_name TEXT
```

## CLASSE DATABASE UTILITY

Questa classe contiene numerose funzioni che aiutano a gestire il proprio database per ottimizzare, riparare le tabelle o lo stesso database.

### INIZIALIZZARE LA CLASS UTILITY

Per inizializzare la classe Utility, i driver del proprio database devono essere precedentemente caricati, poiché la classe si basa su di essi. Per caricare la classe si usa la seguente istruzione:

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

Una volta inizializzata la classe si potrà accedere alle funzioni utilizzando l'oggetto `$this->dbutil`:

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

- `$this->dbutil->list_databases()` restituisce un array di nomi di database:

```
$dbs = $this->dbutil->list_databases();  
  
foreach ($dbs as $db)  
{  
    echo $db;  
}
```

- `$this->dbutil->database_exists()` alcune volte è necessario conoscere se un database esiste. Questa funzione, che è case sensitive restituisce un valore booleano TRUE/FALSE. Invece l'argomento della funzione, `database_name` definisce la tabella che si sta cercando.

```
if ($this->dbutil->database_exists('database_name'))  
{  
    // altro codice  
}
```

- `$this->dbutil->optimize_table('table_name')` questa funzione permette di ottimizzare una tabella usando come primo parametro il nome della tabella. Viene restituito TRUE/FALSE a seconda del successo o meno della funzione. È



bene precisare che non tutte le piattaforme supportano questa ottimizzazione: questa funzione è disponibile infatti solo con i database MySQL/MySQLi.

```
if ($this->dbutil->optimize_table('table_name'))
{
    echo 'L'operazione è stata effettuata!';
}
```

- `$this->dbutil->repair_table('table_name')` permette di riparare una tabella utilizzando nella funzione, come primo parametro, il nome della tabella: viene restituito TRUE/FALSE a seconda del successo o meno dell'operazione. È bene precisare che non tutte le piattaforme supportano questa ottimizzazione: questa funzione è disponibile infatti solo con i database MySQL/MySQLi.

```
if ($this->dbutil->repair_table('table_name'))
{
    echo 'L'operazione è stata effettuata!';
}
```

- `$this->dbutil->optimize_database()` permette di ottimizzare il database quando si è connessi alla classe DB. Restituisce un array che contiene messaggi di stato DB oppure FALSE in caso di fallimento. È bene precisare che non tutte le piattaforme supportano questa ottimizzazione: questa funzione è disponibile infatti solo con i database MySQL/MySQLi.

```
$result = $this->dbutil->optimize_database();

if ($result !== FALSE)
{
    print_r($result);
}
```

- `$this->dbutil->csv_from_result($db_result)` permette di generare un file Comma-Separated Value (CSV) da un risultato di query. Il primo parametro della funzione deve contenere l'oggetto della propria query:

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

$query = $this->db->query("SELECT * FROM mytable");

echo $this->dbutil->csv_from_result($query);
```

Il secondo e il terzo parametro permettono di impostare il delimitatore e il carattere di ritorno a capo (per default il tasto tab è il delimitatore, mentre “\n” è usato come carattere di newline). Per esempio:

```
$delimiter = ",";
$newline = "\r\n";

echo $this->dbutil->csv_from_result($query, $delimiter, $newline);
```

Nota: si presti attenzione al fatto che questa funzione non scrive il file [CSV](#) per noi: semplicemente definisce il layout; per scrivere il file è necessario utilizzare l'Helper File (si veda la sezione [9.8 a pagina 386](#)).

- `$this->dbutil->xml_from_result($db_result)` viene generato un file eXtensible Markup Language ([XML](#)) dal risultato di una query. Il primo parametro si aspetta il risultato di una query, mentre il secondo può contenere un array opzionale di parametri di configurazione. Per esempio:

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

$query = $this->db->query("SELECT * FROM mytable");

$config = array (
    'root'    => 'root',
    'element' => 'element',
    'newline' => "\n",
    'tab'     => "\t"
);

echo $this->dbutil->xml_from_result($query, $config);
```

Nota: questa funzione non scrive fisicamente il file [XML](#), ma si occupa di definirne semplicemente il layout. Se si ha bisogno di questa specifica funzionalità ci si dovrà riferire l'Helper file (si veda la sezione [9.8 a pagina 386](#)).

- `$this->dbutil->backup()` permette di effettuare un backup dell'intero database o di alcune tabelle. Il backup viene compresso in un formato a scelta tra Zip o Gzip ed è utilizzabile solo nel database MySQL.

Nota: a causa del tempo di esecuzione limitato per le operazioni [PHP](#), il backup di database di grandi dimensioni potrebbe non essere possibile. In questo caso si può effettuare una copia di sicurezza dal server SQL tramite riga di comando. Ecco alcuni esempi:

```
// Caricamento della classe DB
$this->load->dbutil();

// Backup dell'intero database e assegnazione ad una variabile
$backup =& $this->dbutil->backup();

// Caricamento dell'Helper File e scrittura del file sul server
$this->load->helper('file');
write_file('/path/to/mybackup.gz', $backup);

// Caricamento dell'Helper Download e invio del file sul server
$this->load->helper('download');
force_download('mybackup.gz', $backup);
```

## LE PREFERENZE DEL BACKUP

Si possono definire le preferenze del backup inviando un array di valori come primo parametro alla funzione. Per esempio:

```
$prefs = array(
    'tables'      => array('table1', 'table2'), // Array delle
    // tabelle di cui fare una copia
    'ignore'     => array(),                  // Lista delle tabelle da
    // omettere dal backup
    'format'     => 'txt',                    // gzip, zip, txt
    'filename'   => 'mybackup.sql',          // Nome del file (solo per
    // i file zip)
    'add_drop'   => TRUE,                     // Per aggiungere le
    // istruzioni DROP TABLE al backup
    'add_insert' => TRUE,                     // Per aggiungere i dati
    // INSERT al backup
    'newline'    => "\n"                     // Specifica il carattere
    // Newline usato nel backup
);

$this->dbutil->backup($prefs);
```

## DESCRIZIONE DELLE PREFERENZE

Preferenze	Val.Default	Opzioni	Descrizione
tables	array vuoto	nessuno	un array di tabelle
ignore	array vuoto	nessuno	un array di tabelle
format	gzip	gzip, zip, txt	formato di esportazione
filename	data attuale	nessuno	nome del file di backup
add_drop	TRUE	TRUE/FALSE	include le dichiarazioni DROP TABLE nel file di esportazione SQL
add_insert	TRUE	TRUE/FALSE	include le dichiarazioni INSERT TABLE nel file di esportazione SQL
newline	\n	'\n ', '\r ', '\r \n '	carattere newline da utilizzare nel file di esportazione SQL

## 8.8 CLASSE EMAIL

La classe di CodeIgniter per le email è caratterizzata da robuste funzionalità capaci di soddisfare la maggior parte delle esigenze:

- protocolli multipli: mail, sendmail, SMTP
- recipienti multipli
- CC e BCC
- HTML e email testuali
- allegati
- a capo automatico
- priorità
- modalità batch BCC: le grandi liste e-mail possono essere suddivise in piccoli parti BCC
- strumenti Email Debug

### INVIO DELL'EMAIL

L'invio di posta elettronica non solo è semplice, ma anche veloce ed immediato. È possibile configurare inline ogni impostazione o definirle in un apposito file di configurazione. L'esempio seguente dimostra la facilità con cui si può inviare una e-mail.

Nota: si presuppone che l'e-mail sia spedita da uno dei propri controller.

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

$this->email->from('your@example.com', 'Your Name');
$this->email->to('someone@example.com');
$this->email->cc('another@another-example.com');
$this->email->bcc('them@their-example.com');

$this->email->subject('Test');
$this->email->message('Test della classe Email');

$this->email->send();

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

## IMPOSTARE LE PREFERENZE

Ci sono diciassette diversi parametri preposti a definire la classe per l'invio dei messaggi di posta elettronica. Questi vengono impostati manualmente come descritto precedentemente, o automaticamente tramite le preferenze memorizzate nel file di configurazione. Le preferenze sono impostate passando un array di valori alla funzione che inizializza l'email. Ecco un esempio:

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

Nota: la maggior parte delle preferenze hanno valori predefiniti che verranno utilizzati se non diversamente impostati.

## IMPOSTAZIONE CON IL FILE CONFIG

Se si preferisce non impostare le preferenze utilizzando il metodo di cui sopra, si può invece utilizzare un file di configurazione. É sufficiente creare un nuovo file chiamato **email.php**, aggiungendo al suo interno l'array **\$config**. Il file salvato nel percorso **/config/email.php** sarà utilizzato automaticamente e non sarà necessario utilizzare la funzione `$this->email->initialize()`.

## PREFERENZE DELL'EMAIL

Qui di seguito si riporta una lista di tutte le impostazioni e dei valori che permettono di impostare la classe secondo le proprie esigenze:

Preferenze	Val.Default	Opzioni	Descrizione
useragent protocol	CodeIgniter mail	Nessuno mail, sendmail, smtp	user agent il protocollo di invio email
mailpath	/usr/sbin/sendmail	Nessuno	Path Server per Send- mail
smtp_host	Nessuno	Nessuno	Indirizzo server SMTP
smtp_user	Nessuno	Nessuno	Username SMTP
smtp_pass	Nessuno	Nessuno	Password SMTP
smtp_port	25	Nessuno	Porta SMTP
smtp_timeout	5 (sec)	Nessuno	Timeout SMTP
wordwrap	TRUE	TRUE/FALSE	Abilita il ritorno a capo
wrapchars	76	Nessuno	Imposta il carattere del ritorno a capo
mailtype	text	text/html	Formato di presenta- zione dell'email
charset	utf-8	Nessuno	Formato dei caratteri (utf-8, iso-8859-1, etc.)
validate	FALSE	TRUE/FALSE	Abilita la validazione dell'email
priority	3	1, 2, 3, 4, 5	Priorità dell'email. 1 = la più alta. 5 = la più bassa. 3 = normale
crlf	\n	\r \n, \n, \r	Carattere di ritorno a capo (si usi r\n per RFC 822)
newline	\n	\r \n, \n, \r	Carattere di ritorno a capo (si usi r\n per RFC 822)
bcc_batch_mode	FALSE	TRUE/FALSE	Abilita la modalità BCC Batch
bcc_batch_size	200	Nessuno	Numero di email nel BCC Batch

## REFERENCE

`$this->email->from()` imposta l'indirizzo email e il nome della persona a cui inviare l'email.

```
$this->email->from('you@example.com', 'Il tuo nome');
```

`$this->email->reply_to()` imposta il campo "rispondi a" (reply to), se questa informazione non è fornita dal campo "da" (from):

```
$this->email->reply_to('you@example.com', 'Il tuo nome');
```

`$this->email->to()` imposta l'indirizzo di destinazione dell'email che può essere unico o multiplo (più indirizzi):

```
$this->email->to('someone@example.com');
```

```
$this->email->to('one@example.com, two@example.com, three@example.com');
```

```
$list = array('uno@example.com', 'due@example.com', 'tre@example.com');  
$this->email->to($list)
```

`$this->email->cc()` imposta il campo Carbon Copy (**CC**); come nel campo "to" può essere un indirizzo singolo o indirizzi multipli separati da una virgola o definiti da un array.

`$this->email->bcc()` imposta il campo blind carbon copy (**BCC**); come nel campo "to" può essere un indirizzo singolo o multipli indirizzi separati da una virgola o definiti da un array.

`$this->email->subject()` definisce l'oggetto (subject) dell'email.

```
$this->email->subject('Questo è il soggetto');
```

`$this->email->message()` definisce il corpo dell'email.

```
$this->email->message('Questo è il testo');
```

`$this->email->set_alt_message()` imposta il corpo di un messaggio alternativo. Si tratta di una stringa testuale (facoltativa) che sarebbe bene utilizzare quando si inviano email in formato **HTML** non sempre visualizzabile correttamente. Permette quindi di specificare un messaggio alternativo senza formattazione **HTML** che viene aggiunto alla stringa di intestazione per le persone che non accettano o non visualizzano le e-mail in tale formato. Se non si imposta il proprio messaggio alternativo, CodeIgniter estrarrà il testo dalla e-mail **HTML** privandolo dei suoi tag.

```
$this->email->set_alt_message('Un messaggio alternativo');
```



`$this->email->clear()` inizializza tutte le variabili delle email ad uno stato vuoto (azzerà i valori). Questa funzione viene utilizzata se l'invio delle email viene utilizzato in un ciclo: questo permette ai dati di essere resettati tra le varie fasi dei cicli in modo da assumerne di nuovi.

```
foreach ($list as $name => $address)
{
    $this->email->clear();

    $this->email->to($address);
    $this->email->from('your@example.com');
    $this->email->subject('Qui le tue informazioni '.$name);
    $this->email->message('Ciao '.$name.' Qui le tue informazioni richieste.');
```

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

Se il parametro sarà settato su `TRUE` ogni allegato (attach) sarà resettato:

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

`$this->email->send()` è la funzione per l'invio delle email, che restituisce un valore booleano `TRUE/FALSE` a seconda del successo o il fallimento della funzione. È quindi preferibile utilizzarla all'interno di una istruzione condizionale:

```
if ( ! $this->email->send() )
{
    // Genera un errore
}
```

`$this->email->attach()` abilita l'invio di allegati all'email: il primo parametro della funzione indica il percorso/nome dell'allegato. È necessario inserire il percorso del file (path) e non un [URL](#). Se si desidera utilizzare più allegati, richiamare la funzione più volte:

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

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

`$this->email->print_debugger()` restituisce una stringa che contiene ogni messaggio del server, come l'header, il testo del messaggio. Questa funziona viene utilizzata per il debug.

### Word Wrapping

Se si abilita il “ritorno a capo” (raccomandato per conformarsi a RFC 822) e si dispone di un link molto esteso nella propria e-mail, questo potrebbe essere spezzato su più righe, rendendo non efficace il collegamento. CodeIgniter consente di ignorare manualmente il ritorno a capo all’interno di una parte del messaggio come in questo caso:

```
{unwrap}http://example.com/a_long_link_that_should_not_be_wrapped.html{/unwrap}
```

Se quindi si vuole preservare l’integrità dei link o di qualsiasi codice particolare si posiziona il testo che non si vuole spezzare tra i delimitatori: {unwrap} {/unwrap}

## 8.9 CLASSE ENCRYPTION

La classe Encryption fornisce una crittografia dei dati a due vie basata sulla libreria Mcrypt che, a partire dalla versione 2.2.0 di CodeIgniter, deve essere necessariamente installata; in caso contrario la classe Encrypt non sarà utilizzabile.

### Definizione 16: Mcrypt

---

**Mcrypt**

È una libreria che fornisce un alto grado di sicurezza adeguato per la memorizzazione persistente delle informazioni per esempio di un database. La sua implementazione supporta diversi algoritmi e cifrari come DES, TripleDES, Blowfish (default), 3-WAY, SAFER-SK64, SAFER-SK128, TWO-FISH, TEA, RC2 e GOST in CBC, OFB, CFB e ECB. Inoltre, sono supporti RC6 e IDEA che però sono considerate “non-free”. CFB/OFB per impostazione predefinita sono a 8 bit.

---

### Definizione 17: Crittografia

---

**Crittografia**

È la branca della crittologia che tratta delle “scritture nascoste”, ovvero dei metodi per rendere un messaggio “offuscato” in modo da non essere comprensibile/intelligibile a persone non autorizzate a leggerlo. Maggiori informazioni: <http://www.php.net/manual/en/book.mcrypt.php>

---

## IMPOSTARE LA PROPRIA CHIAVE

Una chiave (key) è un pezzo di informazione che controlla il processo di crittografia e permette ad una stringa criptata di essere decodificata. La chiave scelta costituisce l'unico mezzo per decodificare i dati precedentemente cifrati con la stessa chiave: non solo la si deve scegliere con cura, ma non deve mai essere cambiata se si intende utilizzarla per memorizzare dati persistenti. Appare quindi ovvio che si deve custodire con estrema attenzione: se qualcuno accedesse alla chiave, tutti i dati del progetto potranno essere facilmente decodificati. Per trarre il massimo vantaggio dell'algoritmo di crittografia, la chiave dovrebbe essere lunga 32 caratteri (128 bit) e costituire una stringa totalmente casuale, con numeri e lettere maiuscole e minuscole. La chiave non deve mai essere una stringa di testo semplice. La suddetta chiave può essere sia memorizzata nel file `/application/config/config.php`, oppure si può progettare un proprio meccanismo di memorizzazione e passare la chiave dinamicamente quando sono necessarie le operazioni di codifica/decodifica. Per salvare la chiave nel file `/application/config/config.php`, si imposti il seguente parametro:

```
$config['encryption_key'] = "LA PROPRIA CHIAVE";
```

## LUNGHEZZA DEL MESSAGGIO

È importante sapere che i messaggi codificati con la funzione di crittografia saranno di circa 2,6 volte più lunghi del messaggio originale. Ad esempio, se si effettua la crittografia della stringa “my super secret data”, che è di 21 caratteri di lunghezza, vi ritroverete con una stringa codificata di circa 55 caratteri. Non bisogna sottovalutare questo aspetto, perché anche se lo spazio di memorizzazione non è più un problema insormontabile, esistono delle limitazioni difficilmente aggirabili: per esempio, per la memorizzazione dei cookie sono disponibili solo 4KB di informazioni.

## INIZIALIZZARE LA CLASSE

La classe di crittografia viene inizializzata nel Controller utilizzando la funzione `$this->load->library`:

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

Una volta caricato, l'oggetto della libreria Encrypt sarà disponibile con `$this->encrypt`.

## REFERENCE

- `$this->encrypt->encode()` esegue la crittografia dei dati e li restituisce come una stringa. Per esempio:

```
$msg = 'My secret message';  
  
$encrypted_string = $this->encrypt->encode($msg);
```

Opzionalmente si può passare la chiave di cifratura tramite il secondo parametro, se non si vuole utilizzare quello presente nel file di configurazione:

```
$msg = 'My secret message';  
$key = 'super-secret-key';  
  
$encrypted_string = $this->encrypt->encode($msg, $key);
```

- `$this->encrypt->decode()` decrittografa una stringa codificata. Esempio:

```
$encrypted_string = 'APANtByIGI1BpVXZTJgcsAG8GZl8pdwwa84';  
$plaintext_string = $this->encrypt->decode($encrypted_string);
```

Opzionalmente si può passare la chiave di cifratura tramite il secondo parametro, se non si vuole utilizzare quella presente nel file di configurazione:

```
$msg = 'My secret message';  
$key = 'super-secret-key';  
  
$encrypted_string = $this->encrypt->decode($msg, $key);
```

- `$this->encrypt->set_cipher()` permette di impostare un cifrario Mcrypt. Di default viene utilizzato `MCRYPT_RIJNDAEL_256`. Per esempio:

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

Si prega di visitare [php.net](http://php.net/mcrypt) (<http://php.net/mcrypt>) per un elenco di cifrari disponibili.

Per i più curiosi, o meglio per quelli più saggi, è consigliato verificare manualmente se il server supporta Mcrypt attraverso la seguente istruzione:

```
echo ( ! function_exists('mcrypt_encrypt')) ? 'No!' : 'Si!';
```

`$this->encrypt->set_mode()` permette di impostare una modalità Mcrypt. Di default viene utilizzata `MCRYPT_MODE_CBC`. Ecco un semplice esempio:

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

Si consiglia di visitare [php.net \(http://php.net/mcrypt\)](http://php.net/mcrypt) per un elenco delle modalità disponibili.

- `$this->encrypt->sha1()` è la funzione di codifica SHA1 a cui si fornisce una stringa che restituisce un hash 160 bit. Nota: SHA1, come MD5 non è decodificabile. Per esempio:

```
$hash = $this->encrypt->sha1('Una stringa');
```

Molte installazioni di [PHP](#) hanno di default il supporto a SHA1, quindi se tutto ciò che serve è questa codifica hash, allora è decisamente più semplice e veloce utilizzare la funzione nativa:

```
$hash = sha1('Una stringa');
```

Se invece il server non supporta SHA1, ci si può rivolgere alla funzione:

- `$this->encrypt->encode_from_legacy($orig_data, $legacy_mode = MCRYPT_MODE_ECB, $key = '')` consente di ricodificare i dati originariamente crittografati con CodeIgniter 1.x per essere compatibili con la libreria presente in CodeIgniter 2.x. È necessario utilizzare questo metodo solo se i dati crittografati sono memorizzati in modo persistente, come in un file o in un database e su di un server che supporta Mcrypt. “Light” utilizza la crittografia come i dati di sessione o le transazioni crittografate e non richiede alcun intervento da parte dello sviluppatore. Tuttavia, le sessioni crittografate esistenti verranno distrutte in quanto i dati crittografati prima della versione 2.x non possono essere decodificati.

Perché utilizzare solo un metodo per ricodificare i dati invece di mantenere i metodi legacy sia per la codifica che per la decodifica? Gli algoritmi nella libreria Encryption sono stati migliorati in CodeIgniter 2.x, sia sotto il profilo delle prestazioni che sotto quello della sicurezza e si sconsiglia di utilizzare i vecchi metodi. Ovviamente si può estendere la libreria di crittografia, se lo si desidera e sostituire i nuovi metodi con quelli precedenti e mantenere la compatibilità con i dati crittografati da CodeIgniter 1.x, ma questa è una decisione che uno sviluppatore dovrebbe ponderare con cautela.

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

Parametro	Val.Default	Descrizione
<code>\$orig_data</code>	n/a	i dati originali crittografati da CodeIgniter 1.x
<code>\$legacy_mode</code>	<code>MCRYPT_MODE_ECB</code>	la modalità Mcrypt è stata utilizzata precedentemente per generare i dati originali crittografati. In CodeIgniter 1.x la modalità predefinita era <code>MCRYPT_MODE_ECB</code>
<code>\$key</code>	n/a	la chiave crittografica: essa è specificata solitamente nel file config specificato poco sopra

## 8.10 CLASSE UPLOAD FILE

Questa classe fornisce diversi parametri per aiutare lo sviluppatore nel caricare i file sul server. Tra le preferenze possiamo contare molte parametri che circoscrivono il tipo e la dimensione del file.

### LE FASI

Il processo di caricamento di un file avviene attraverso le seguenti fasi:

- viene visualizzato un form attraverso cui l'utente seleziona il file e lo carica sul server(upload)
- quando il form viene eseguito, il file viene caricato sul server nel percorso specificato
- successivamente il file viene validato per essere sicuri che risponda ai parametri corretti
- una volta caricato con successo, verrà visualizzato un messaggio per confermare l'esecuzione corretta dell'operazione

### CREARE UN FORM DI UPLOAD

Si apra il proprio editor di testo preferito, quindi si crei un form con il nome `upload_form.php` e lo si salvi nel percorso `/applications/views/`. Per esempio:

```
<html>
<head>
<title>Upload Form</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>
```



Per creare il tag di apertura del form si è utilizzato un Helper Form. L'upload dei file richiede un "form multipart", in modo che l'Helper si occupi della sintassi corretta per noi. È presente anche una variabile **\$error** che mostra i messaggi di errore nel caso in cui l'utente faccia qualcosa di sbagliato.

## PAGINA DI SUCCESSO

Si crei un nuovo form con il nome `upload_success.php` e lo si salvi nel percorso `/applications/views/`. Per esempio:

```
<html>
<head>
<title>Upload Form</title>
</head>
<body>

<h3>Il file è stato caricato con successo</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', 'Carica un altro file!'); ?></p>

</body>
</html>
```

## IL CONTROLLER

Sempre utilizzando un editor di testo, nel percorso `/applications/controllers/` si definisca il file **upload.php** con il codice seguente:

```

<?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 CARTELLA DI UPLOAD

Ovviamente è necessaria una cartella che sia adibita a contenere i file caricati. Questa deve essere creata nella directory root di CodeIgniter con il nome **uploads** e i permessi di scrittura, lettura, esecuzione (777) abilitati.

É possibile provare il form appena creato attraverso un [URL](#) come il seguente:

```
example.com/index.php/upload/
```

Provando ad effettuare l'upload di un file immagine (come jpeg, gif o png), se tutto è corretto, la cartella di destinazione alla fine dell'operazione di upload conterrà i file selezionati.

## REFERENCE

Come in altre classi di CodeIgniter, è possibile inizializzare la classe Upload utilizzando l'istruzione:

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

Una volta caricata, l'oggetto della libreria sarà disponibile con `$this->upload`

## IMPOSTARE LE PREFERENZE

Le impostazioni della classe possono essere definite nel controller modificando i seguenti parametri:

```
$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);

// In alternativa è possibile impostare le preferenze richiamando la funzione
// di inizializzazione. Utile se si carica automaticamente la classe con:
$this->upload->initialize($config);
```

## ELENCO DEI PARAMETRI

Ecco un elenco di tutte le impostazioni e dei valori utilizzabili:

Parametro	Val.Default	Opzioni	Descrizione
upload_path	Nessuno	Nessuno	Il percorso della cartella in cui l'upload viene effettuato. La cartella che deve avere i permessi di scrittura e può avere un path assoluto o relativo
allowed_types	Nessuno	Nessuno	I tipi MIME corrispondenti ai tipi di file che possono essere caricati. Di solito l'estensione del file può essere utilizzato come tipo mime. Separare più tipi con un pipe
file_name	Nessuno	Nome del file	Se impostato CodeIgniter rinomina il file caricato con questo nome. L'estensione prevista nel nome del file deve essere un tipo di file consentito
overwrite	FALSE	TRUE/FALSE	Se impostato su true, allora se un file con lo stesso nome di quello che si sta caricando esiste, verrà sovrascritto. Se impostato su false, un verrà aggiunto al nome del file un numero
max_size	0	Nessuno	La dimensione massima (in kilobyte) prevista per il file da caricare. Impostare a zero questo valore per non avere alcun limite. Nota: La maggior parte delle installazioni di PHP hanno un limite, specificato nel file php.ini. Di solito questo è di 2 MB (o 2048 KB) per impostazione predefinita
max_width	0	Nessuno	La massima width (larghezza) in pixel del file. Impostare a zero questo valore per non avere alcun limite
max_height	0	Nessuno	La massima height (altezza) in pixel del file. Impostare a zero questo valore per non avere alcun limite
max_filename	0	Nessuno	La lunghezza massima (length) del nome del file. Impostare a zero questo valore per non avere alcun limite
encrypt_name	FALSE	TRUE/FALSE	Se impostato su TRUE il nome del file verrà convertito in una stringa crittografata casuale. Questo può essere utile se si desidera che il file possa essere salvato con un nome che non possa essere individuate dalla persona che ha effettuato l'upload
remove_spaces	TRUE	TRUE/FALSE	Se impostato su TRUE, eventuali spazi nel nome del file verranno convertiti in underscore (sottolineatura). Questa è una opzione raccomandata

## PREFERENZE TRAMITE CONFIG

Se si preferisce non impostare le preferenze utilizzando il metodo di qui sopra, si può invece utilizzare il file di configurazione **upload.php** in cui verrà aggiunto l'array `\$config`. Si salvi il file di configurazione nel percorso `/config/upload.php` ed esso verrà utilizzato automaticamente dal sistema. Questo metodo non deve essere utilizzato con `$this->upload->initialize`.

Sono disponibili le seguenti funzioni:

- `$this->upload->do_upload()` esegue l'upload in base alle preferenze selezionate. Per impostazione predefinita, la routine di caricamento si aspetta che il file provenga da un campo di form denominato `userfile`, e il form deve essere di tipo `multipart`:

```
<form method="post" action="some_action" enctype="multipart/form-data" />
```

```
$field_name = "some_field_name";  
$this->upload->do_upload($field_name)
```

Se si desidera impostare il proprio campo per il form, lo si può fare semplicemente passando il suo valore alla funzione `do_upload`:

```
$field_name = "some_field_name";  
$this->upload->do_upload($field_name)
```

- `$this->upload->display_errors()` recupera eventuali messaggi di errore se `do_upload()` restituisce `FALSE`. La funzione non visualizza automaticamente l'errore, ma il suo valore restituito può essere assegnato ad una variabile e utilizzata come si preferisce.

Per impostazione predefinita la funzione di cui sopra spezza ogni errore con un ritorno a capo con il tag `<p>`. È comunque possibile modificare questo delimitatore:

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

- `$this->upload->data()` si tratta di una funzione helper che restituisce un array con tutti i dati relativi al file caricato. Qui di seguito si mostra il prototipo dell'array:

```

Array
(
    [file_name]    => mypic.jpg
    [file_type]    => image/jpeg
    [file_path]    => /path/to/your/upload/
    [full_path]    => /path/to/your/upload/jpg.jpg
    [raw_name]     => mypic
    [orig_name]    => mypic.jpg
    [client_name]  => mypic.jpg
    [file_ext]     => .jpg
    [file_size]    => 22.2
    [is_image]     => 1
    [image_width]  => 800
    [image_height] => 600
    [image_type]   => jpeg
    [image_size_str] => width="800" height="200"
)

```

## PARAMETRI DELL'ARRAY

Qui di seguito una descrizione di tutti gli elementi dell'array.

Indice	Descrizione
<code>file_name</code>	il nome del file da caricare compreso di estensione
<code>file_type</code>	il tipo di Mime del file
<code>file_path</code>	il percorso assoluto del file sul server
<code>full_path</code>	il percorso assoluto del file sul server comprensivo del nome del file
<code>raw_name</code>	il nome del file senza estensione
<code>orig_name</code>	Il nome originario del file. È utile nel caso si sia crittografato il nome del file
<code>client_name</code>	Il nome del file fornito dallo user agent client, prima di qualsiasi modifica sul nome del file
<code>file_ext</code>	l'estensione del file comprensivo del simbolo punto (.)
<code>file_size</code>	la dimensione del file in kilobyte
<code>is_image</code>	indica se il file è un'immagine. Se è un'immagine il valore è 1, altrimenti è 0
<code>image_width</code>	larghezza dell'immagine (pixel)
<code>image_height</code>	altezza dell'immagine (pixel)
<code>image_type</code>	formato dell'immagine. Solitamente è l'estensione senza il simbolo punto (.)
<code>image_size_str</code>	una stringa che contiene la larghezza e l'altezza. È utile quando si deve utilizzare un tag con l'immagine

## 8.11 CLASSE VALIDAZIONE DEI FORM

CodeIgniter fornisce una classe che aiuta il lavoro dello sviluppatore quando si tratta di validare i dati inviati al server, minimizzando quindi la scrittura di codice adibito a tale scopo.

### INTRODUZIONE

Prima di spiegare l'approccio utilizzato dal framework per la convalida dei dati, cerchiamo di descrivere lo scenario ideale:

1. viene visualizzato un modulo
2. questo viene compilato e inviato al server
3. se il contenuto non è valido o si è dimenticato di inserire un elemento obbligatorio, il form viene visualizzato nuovamente con i dati e con un messaggio di errore che descrive il problema
4. questo processo continua fino a quando non si compila un form interamente valido

Sul lato ricevente, lo script deve:

1. verificare la presenza dei dati richiesti
2. verificare che i dati siano del tipo corretto, e che soddisfino i criteri impostati. Ad esempio se viene inviato un "nome utente" che deve essere convalidato, questo deve contenere solo caratteri consentiti, deve avere una lunghezza minima e non superare la lunghezza massima. Il nome utente non può essere un "nome utente" già esistente o una parola riservata
3. effettuare l'escape dei dati per la sicurezza
4. pre-formattare i dati in caso di necessità
5. preparare i dati per l'inserimento nel database

Anche se non c'è niente di terribilmente complesso, il processo precedentemente descritto, di solito richiede una notevole quantità di codice adibito alla visualizzazione dei messaggi di errore, e alle varie strutture di controllo all'interno dei form [HTML](#). La validazione dei form anche se semplice, è quasi sempre un lavoro lungo e noioso da implementare.

### TUTORIAL

Quello che segue è una guida per l'implementazione della classe Form Validation di CodeIgniter. Al fine di attuare la validazione dei form si avranno bisogno di tre cose:

1. un file View contenente un form
2. un file View con un messaggio di “successo” (da visualizzare in caso di validazione corretta del form)
3. una funzione di controllo per ricevere ed elaborare i dati inviati

Creiamo queste tre cose, utilizzando come esempio un form per l’iscrizione di un nuovo utente.

## IL FORM

Utilizzando un editor di testo, si crea il file **myform.php** nel percorso **/applications/views/**. Esso conterrà il codice del nostro form:

```
<html>
<head>
<title>My Form</title>
</head>
<body>

<?php echo validation_errors(); ?>

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

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

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

<h5>Password Confirm</h5>
<input type="text" name="passconf" value="" size="50" />

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

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

</form>

</body>
</html>
```



## LA PAGINA DI SUCCESSO

Utilizzando un editor di testo, si definisce il file **formsuccess.php** nel percorso **/applications/views/** che informerà l'utente del corretto invio (e quindi validazione) del form.

```
<html>
<head>
<title>My Form</title>
</head>
<body>

<h3>Il tuo form è stato inviato correttamente!</h3>

<p><?php echo anchor('form', 'Try it again!'); ?></p>

</body>
</html>
```

## IL CONTROLLER

Utilizzando un editor di testo, si crea il file **form.php** nel percorso **/applications/controllers/**.

```
<?php

class Form extends CI_Controller {

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

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

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

Molto semplicemente è possibile testare quando appena descritto provando il seguente [URL](#):

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

Se al momento si invia il modulo, il form verrà caricato nuovamente: questo avviene perché non si è ancora stabilita alcuna regola di convalida. Infatti, poiché la classe Form Validation non sa come comportarsi, restituisce FALSE per impostazione predefinita. La funzione **run()** restituisce TRUE solo se le regole sono state rispettate senza alcuna eccezione.

## SPIEGAZIONE

Qualcuno avrà notato diverse cose sulle pagine di qui sopra:

Il file **myform.php** è un form standard con un paio di eccezioni:

1. esso usa un Helper Form per definire un form per l'inserimento dei dati. Anche se non è strettamente necessario, poiché è possibile utilizzare lo standard [HTML](#), l'utilizzo dell'helper genera l'azione dell'[URL](#) automaticamente, in base all'[URL](#) impostato nel file di configurazione. Questo rende l'applicazione maggiormente portabile nel caso in cui gli indirizzi dovessero cambiare
2. all'inizio del form appare la seguente chiamata di funzione `<?php echo validation_errors(); ?>` Questa funzione restituisce ogni messaggio di errore

prodotto dal validatore. Se non vi è alcun errore, verrà restituita una stringa vuota.

Il controller **form.php** ha la sola funzione predefinita **index()** che inizializza la classe di validazione e carica l'Helper Form (vedi a pagina 389) e l'Helper URL (vedi a pagina 422) utilizzati per visualizzare i file. Viene inoltre eseguita la routine di validazione e, in base al successo o meno dell'operazione, la pagina di "successo" viene caricata. in caso contrario viene visualizzato nuovamente il form.

## IMPOSTAZIONE DELLE REGOLE DI VALIDAZIONE

CodeIgniter permette di impostare molte regole per la validazione dei campi di un form. Per impostare le regole si utilizzerà la funzione `set_rules()`:

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

La funzione di cui sopra è caratterizzato da tre parametri in ingresso:

1. il nome del campo: il nome esatto dato al campo del form
2. un nome "umano" per il campo, ovvero un nome che faccia intuire all'utente immediatamente il contenuto del campo: questa informazione è utile perché il nome del form verrà utilizzato all'interno di eventuali messaggi di errori
3. le regole di validazione per il campo del form

Questo è un esempio che mostra il codice da aggiungere al Controller **form.php**:

```
$this->form_validation->set_rules('username', 'Username', 'required');  
$this->form_validation->set_rules('password', 'Password', 'required');  
$this->form_validation->set_rules('passconf', 'Password Confirmation', '  
    required');  
$this->form_validation->set_rules('email', 'Email', 'required');
```

Il Controller a questo punto apparirà come:

```
<?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', 'Username', 'required');
        $this->form_validation->set_rules('password', 'Password', 'required');
        $this->form_validation->set_rules('passconf', 'Password Confirmation', '
        required');
        $this->form_validation->set_rules('email', 'Email', 'required');

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

Se si invia il form con i campi vuoti compariranno su schermo i messaggi di errore, altrimenti, se tutti i dati sono corretti verrà visualizzata la pagina di successo.

## IMPOSTAZIONE DELLE REGOLE USANDO UN ARRAY

Se si preferisce definire le regole di validazione in una sola azione, è possibile utilizzare un array che verrà passato alla funzione apposita:

```
$config = array(
    array(
        'field' => 'username',
        'label' => 'Username',
        'rules' => 'required'
    ),
    array(
        'field' => 'password',
        'label' => 'Password',
        'rules' => 'required'
    ),
    array(
        'field' => 'passconf',
        'label' => 'Password Confirmation',
        'rules' => 'required'
    ),
    array(
        'field' => 'email',
        'label' => 'Email',
        'rules' => 'required'
    )
);

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

## REGOLE IN CASCATA

Si possono impostare regole multiple associandole mediante pipe (carattere |). Si osservi il terzo parametro:

```
$this->form_validation->set_rules('username', 'Username', 'required|min_length[5]|max_length[12]|is_unique[users.username]');
$this->form_validation->set_rules('password', 'Password', 'required|matches[passconf]');
$this->form_validation->set_rules('passconf', 'Password Confirmation', 'required');
$this->form_validation->set_rules('email', 'Email', 'required|valid_email|is_unique[users.email]');
```

Il codice precedente definisce le seguenti regole:

1. il campo username non può essere più corto di 5 caratteri e più lungo di 12 caratteri
2. il campo password deve contenere un valore identico al campo di conferma password

3. il campo email deve soddisfare le regole alla base degli indirizzi di posta elettronica

Si invii il form senza i dati corretti e si vedranno nuovi messaggi di errore che corrispondono alle regole appena impostate. Ci sono numerose altre regole disponibili.

## PREPPING DATA

Oltre alle funzioni di validazione appena viste, è possibile preparare i propri dati in vari modi. Per esempio, è possibile settare le regole:

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

Nel precedente esempio i campi sono processati con la funzione trim, le password sono convertite in MD5, il nome utente passa per la funzione xss\_clean, che rimuove i dati potenzialmente dannosi.

Qualsiasi funzione PHP nativa che accetta un parametro può essere usato come regola come nel caso di htmlspecialchars, trim, MD5, etc.

Nota: generalmente si desidera usare le funzioni di tipo prepping dopo le regole di validazione, in modo che, se vi è un errore, i dati originali potranno essere visualizzati nuovamente nel form.

### Definizione 18: Trim

---

#### Trim

La funzione TRIM in SQL viene utilizzata per rimuovere il prefisso o il suffisso specificato da una stringa. Il modello più comunemente rimosso sono gli spazi vuoti. La funzione viene chiamata diversamente a seconda del tipo di database.

---

## RIPOPOLARE IL FORM

Finora si è avuto a che fare solo con gli errori: è tempo di ripopolare i campi del form con i dati inviati al server. CodeIgniter offre diverse funzioni di supporto che permettono questa operazione, una delle quali, verrà utilizzata frequentemente:

```
set_value('field name')
```

Si apra la Vista **myform.php** e si aggiorni il valore in ogni campo, usando la funzione `set_value()`. Non ci si deve dimenticare di includere il nome di ogni campo nelle funzioni `set_value()`

```
<html>
<head>
<title>My Form</title>
</head>
<body>

<?php echo validation_errors(); ?>

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

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

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

<h5>Conferma Password</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="Submit" /></div>

</form>

</body>
</html>
```

Ora è possibile ricaricare la propria pagina per testare le modifiche appena introdotte. Si compili il form facendo in modo che si verifichi un errore: si vedrà che il form viene ricaricato con i campi del form che saranno ripopolati. Per maggiori informazioni si faccia riferimento alla sezione Reference che contiene diverse funzioni per ripopolare i menu select, i pulsanti radio e i checkbox. Se come nome di un campo del form si utilizza un array, questo dovrà essere inviato anche come array all'interno della funzione. Per esempio:

```
<input type="text" name="colors[]" value="<?php echo set_value('colors[]'); ?>" size="50" />
```

## CALLBACK: LE FUNZIONI DI VALIDAZIONE

Il sistema di validazione supporta i callback: questo permette di estendere la classe per soddisfare qualsiasi esigenza particolare. Ad esempio, se è necessario eseguire una query del database per vedere se l'utente ha scelto un nome utente univoco, si può creare una funzione di callback con questo preciso scopo. Nel controller si modifichi la regola username con questa:

```
$this->form_validation->set_rules('username', 'Username', 'callback_username_check');
```

Quindi si aggiunga la nuova funzione chiamata `username_check` al proprio controller. Questo è quello che dovrebbe contenere il file del Controller:



```
<?php

class Form extends CI_Controller {

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

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

        $this->form_validation->set_rules('username', 'Username', '
        callback_username_check');
        $this->form_validation->set_rules('password', 'Password', 'required');
        $this->form_validation->set_rules('passconf', 'Password Confirmation', '
        required');
        $this->form_validation->set_rules('email', 'Email', 'required|is_unique[
        users.email]');

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

    public 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;
        }
    }
}

?>
```

Caricando nuovamente il form e utilizzando come nome utente la parola “test” si vedrà che il campo data del form verrà passato alla funzione di callback per essere elaborata. Per invocare un callback si inserisca il nome della funzione in una regola, con il prefisso callback\_. Se si ha bisogno di un parametro aggiuntivo nella funzione

di callback, basta aggiungerlo dopo il nome della funzione tra parentesi quadre, come in: `callback_foo[bar]`, quindi questo sarà passato come secondo argomento della funzione di callback.

Nota: è anche possibile elaborare i dati del form che sono passati alla funzione callback a cui viene restituita. Se il callback restituisce qualcosa di diverso da un valore booleano TRUE/FALSE si presume che i dati provengano da un nuovo form.

## CONFIGURARE I MESSAGGI DI ERRORE

Tutti i messaggi di errore nativi sono localizzati nel seguente file di linguaggio: `/language/english/form_validation_lang.php`. Se si vuole configurare un proprio messaggio si deve editare questo file, oppure utilizzare la seguente funzione:

```
$this->form_validation->set_message('rule', 'Error Message');
```

Dove la parola **rule** corrisponde al nome di una particolare regola e il testo **Error Message** a quello che si desidera venga visualizzato. Se si include nella stringa di errore `%s`, questa verrà sostituita con il nome “umano” del campo settato nella propria regola. Per esempio nella funzione di callback di qui sopra, il messaggio di errore viene impostato passandolo alla funzione:

```
$this->form_validation->set_message('username_check')
```

Un'altra possibilità è quella di sovrascrivere ogni messaggio di errore che si trova nel file del linguaggio. Per esempio per cambiare il messaggio relativo alla regola “required” (obbligatorio) è possibile operare nel seguente modo:

```
$this->form_validation->set_message('required', 'il proprio messaggio di errore');
```

## TRADURRE I NOMI DEI CAMPI

Se si vogliono memorizzare i nomi “umani” passati alla funzione `set_rules` in un file di linguaggio e quindi rendere il nome in grado di essere tradotto, si utilizzi il prefisso **lang** davanti al nome del campo “umano” come nell'esempio:

```
$this->form_validation->set_rules('first_name', 'lang:first_name', 'required');
```

Quindi si memorizzi il nome in un proprio array del file di linguaggio (senza alcun prefisso):

```
$lang['first_name'] = 'First Name';
```

Nota: se si memorizzano gli elementi del proprio array in un file di linguaggio, questo non sarà automaticamente caricato da CI, ma dovrà essere eseguito mediante il proprio Controller, utilizzando:

```
$this->lang->load('file_name');
```

Per maggiori informazioni si rimanda all'esame approfondito della classe Language a pagina [308](#).

## CAMBIARE I DELIMITATORI DEGLI ERRORI

La classe Form Validation come valore predefinito aggiunge un tag paragrafo <p> attorno ad ogni messaggio di errore visualizzato. È possibile modificare questo delimitatore globalmente oppure per un singolo messaggio.

- modifica globale. In questo caso si aggiunge alla propria funzione Controller, dopo aver caricato la classe Form Validation, la seguente istruzione:

```
$this->form_validation->set_error_delimiters('<div class="error">', '</div>');
```

In questo esempio, si sono impostati come nuovi delimitatori i tag <div>.

- modifica individuale. Ciascuna delle due funzioni di errore mostrate in questa guida sono forniti di singoli delimitatori come segue:

```
<?php echo form_error('field name', '<div class="error">', '</div>'); ?>
```

oppure

```
<?php echo validation_errors('<div class="error">', '</div>'); ?>
```

## VISUALIZZARE I SINGOLI ERRORI

Se si preferisce visualizzare un messaggio di errore affianco ad ogni campo del form piuttosto che in una lista, è necessario appoggiarsi alla funzione `form_error()`. Si modifichi il proprio form come segue e si effettui una prova:

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

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

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

<h5>Email Address</h5>
<?php echo form_error('email'); ?>
<input type="text" name="email" value="<?php echo set_value('email'); ?>" size
      ="50" />
```

Se non ci sono errori, ovviamente nulla verrà visualizzato, in caso contrario comparirà il valore errato. Un aspetto a cui prestare attenzione è che quando si utilizza un array come nome di un campo del form, è necessario specificarlo anche nella funzione:

```
<?php echo form_error('options[size]'); ?>
<input type="text" name="options[size]" value="<?php echo set_value("options[
size]"); ?>" size="50" />
```

Per maggiori informazioni si rimanda alla sezione a pagina [262](#).

## SALVARE LE PROPRIE REGOLE

Una caratteristica utile della classe Form Validation permette di memorizzare tutte le proprie regole in un file config. Inoltre è possibile organizzare queste regole in gruppi che possono essere caricati automaticamente quando un controller/metodo sono chiamati, oppure manualmente, quando necessari. Le regole di validazione possono essere salvate facilmente in un file denominato `form_validation.php` nel percorso `/application/config/`. In questo file sarà definito un array di nome (obbli-

gatorio!) **\$config** con la definizione delle proprie regole. Un semplice esempio è mostrato nell'esempio seguente:

```
$config = array(  
    array(  
        'field' => 'username',  
        'label' => 'Username',  
        'rules' => 'required'  
    ),  
    array(  
        'field' => 'password',  
        'label' => 'Password',  
        'rules' => 'required'  
    ),  
    array(  
        'field' => 'passconf',  
        'label' => 'Password Confirmation',  
        'rules' => 'required'  
    ),  
    array(  
        'field' => 'email',  
        'label' => 'Email',  
        'rules' => 'required'  
    )  
);
```

Le regole di validazione così definite vengono caricate automaticamente e utilizzate quando si chiama la funzione **run()**. Per quanto riguarda invece la loro organizzazione in gruppi, è necessario inserire le regole in sotto array. Si consideri il seguente esempio in cui vengono definiti due gruppi di regole chiamate “signup” e “email”:

```

$config = array(
    'signup' => array(
        array(
            'field' => 'username',
            'label' => 'Username',
            'rules' => 'required'
        ),
        array(
            'field' => 'password',
            'label' => 'Password',
            'rules' => 'required'
        ),
        array(
            'field' => 'passconf',
            'label' => 'PasswordConfirmation',
            'rules' => 'required'
        ),
        array(
            'field' => 'email',
            'label' => 'Email',
            'rules' => 'required'
        )
    ),
    'email' => array(
        array(
            'field' => 'emailaddress',
            'label' => 'EmailAddress',
            'rules' => 'required|valid_email'
        ),
        array(
            'field' => 'name',
            'label' => 'Name',
            'rules' => 'required|alpha'
        ),
        array(
            'field' => 'title',
            'label' => 'Title',
            'rules' => 'required'
        ),
        array(
            'field' => 'message',
            'label' => 'MessageBody',
            'rules' => 'required'
        )
    )
);

```

A questo punto se si desidera richiamare solo un determinato gruppo di regole, è necessario passare il relativo nome alla funzione **run()**. Per esempio se si volesse

chiamare il solo gruppo signup si procederà nel seguente modo:

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

Una possibilità alquanto interessante è quella di associare un gruppo di regole ad una funzione del proprio Controller in modo da richiamare automaticamente le regole di validazione desiderate. Il metodo si basa sul chiamare il gruppo di regole nello stesso identico modo della coppia “controller” e “metodo” che si desiderano utilizzare. Per esempio, se si possiede un controller di nome **Member** e di un metodo chiamato **signup** ecco come apparirà il relativo prototipo:

```
<?php
class Member extends CI_Controller {

    function signup()
    {
        $this->load->library('form_validation');

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

Nel proprio file config, il proprio gruppo di regole prenderà il nome di **member/-signup**:

```
$config = array(
    'member/signup' => array(
        array(
            'field' => 'username',
            'label' => 'Username',
            'rules' => 'required'
        ),
        array(
            'field' => 'password',
            'label' => 'Password',
            'rules' => 'required'
        ),
        array(
            'field' => 'passconf',
            'label' => 'PasswordConfirmation',
            'rules' => 'required'
        ),
        array(
            'field' => 'email',
            'label' => 'Email',
            'rules' => 'required'
        )
    )
);
```

Quando un gruppo di regole si chiama come un controller/metodo, esso verrà utilizzato automaticamente invocando la funzione **run()** dalla classe/metodo.

## UTILIZZARE UN ARRAY PER I NOMI DEI CAMPI

La classe Form Validation supporta l'utilizzo degli array nei nomi dei campi del form. Si consideri il seguente codice:

```
<input type="text" name="options[]" value="" size="50" />
```

Se come in questo caso si utilizza un array, obbligatoriamente si dovrà specificare il nome esatto dell'array anche nella funzione. Per esempio per definire un gruppo di regole per l'esempio sopra riportato, si dovrà usare:

```
$this->form_validation->set_rules('options[]', 'Options', 'required');
```

oppure per visualizzare un errore:



```
<?php echo form_error('options[]'); ?>
```

o ancora per ripopolare il campo:

```
<input type="text" name="options[]" value="<?php echo set_value('options[]');  
?>" size="50" />
```

É anche possibile utilizzare un array multidimensionale come nome del campo. Per esempio:

```
<input type="text" name="options[size]" value="" size="50" />
```

o anche:

```
<input type="text" name="sports[nba][basketball]" value="" size="50" />
```

Come rimarcato nel primo esempio, è assolutamente obbligatorio utilizzare l'esatto nome dell'array nelle funzioni helper:

```
<?php echo form_error('sports[nba][basketball]'); ?>
```

Se si usano i checkbox (o altri tipi di campi) che hanno opzioni multiple, non bisogna dimenticare di inserire delle parentesi quadre vuote (che non racchiudono alcun valore), in modo che tutte le selezioni vengano aggiunte all'array POST:

```
<input type="checkbox" name="options[]" value="red" />  
<input type="checkbox" name="options[]" value="blue" />  
<input type="checkbox" name="options[]" value="green" />
```

oppure, se si usa un array multidimensionale:

```
<input type="checkbox" name="options[color][]" value="red" />  
<input type="checkbox" name="options[color][]" value="blue" />  
<input type="checkbox" name="options[color][]" value="green" />
```

Quando si utilizza una funzione helper, è necessario includere le parentesi quadre come segue:

```
<?php echo form_error('options[color][]'); ?>
```

## REFERENCE DELLE REGOLE

Qui di seguito una lista di tutte le regole native di CodeIgniter disponibili per l'uso:

Regola	Parametro	Descrizione	Esempio
<code>required</code>	No	Restituisce FALSE se l'elemento del form è vuoto	
<code>matches</code>	Si	Restituisce FALSE se l'elemento del form non corrisponde a quello nel parametro	<code>matches[form_item]</code>
<code>is_unique</code>	Si	Restituisce FALSE se l'elemento form non è unico nel tabella e campo indicato nel parametro	<code>is_unique[table.field]</code>
<code>min_length</code>	Si	Restituisce FALSE se l'elemento form ha meno caratteri rispetto a quelli indicati nel parametro	<code>min_length[6]</code>
<code>max_length</code>	Si	Restituisce FALSE se l'elemento form ha più caratteri rispetto a quelli indicati nel parametro	<code>max_length[12]</code>
<code>exact_length</code>	Si	Restituisce FALSE se l'elemento form non ha lo stesso numero di caratteri indicati nel parametro	<code>exact_length[8]</code>
<code>greater_than</code>	Si	Restituisce FALSE se l'elemento form ha un valore numerico inferiore a quello indicato nel parametro o non è un valore numerico	<code>greater_than[8]</code>
<code>less_than</code>	Si	Restituisce FALSE se l'elemento form ha un valore numerico superiore a quello indicato nel parametro o non è un valore numerico	<code>less_than[8]</code>
<code>alpha</code>	No	Restituisce FALSE se l'elemento form ha caratteri differenti da quelli alfabetici	
<code>alpha_numeric</code>	No	Restituisce FALSE se l'elemento form ha caratteri differenti da quelli alfanumerici	
<code>alpha_dash</code>	No	Restituisce FALSE se l'elemento form ha caratteri differenti da quelli alfabetici, underscore o dash	
<code>numeric</code>	No	Restituisce FALSE se l'elemento form ha caratteri differenti da quelli numerici	
<code>integer</code>	No	Restituisce FALSE se l'elemento form ha un tipo di dati differente da un intero	
<code>decimal</code>	Si	Restituisce FALSE se l'elemento form non è identico a quello indicato nel parametro	
<code>is_natural</code>	No	Restituisce FALSE se l'elemento form qualcosa di diverso da un numero naturale: 0, 1, 2, 3, etc.	
<code>is_natural_no_zero</code>	No	Restituisce FALSE se l'elemento form contiene qualcosa di diverso da un numero naturale, ma non zero: 1, 2, 3, etc.	
<code>valid_email</code>	No	Restituisce FALSE se l'elemento form non contiene un indirizzo email valido	
<code>valid_emails</code>	No	Restituisce FALSE se qualsiasi valore inserito in una lista, e separati da virgole, non corrisponde ad un indirizzo email valido	
<code>valid_ip</code>	No	Restituisce FALSE l'IP fornito non è valido. Accetta un parametro opzionale IPv4 oppure IPv6 per indicare un formato specifico	
<code>valid_base64</code>	No	Restituisce FALSE se la stringa fornita qualcosa di diverso da caratteri Base64	

Nota: queste regole possono essere richiamate come singole funzioni. Per esempio:

```
$this->form_validation->required($string);
```

É anche possibile utilizzare qualsiasi funzione nativa [PHP](#) che si basa su un parametro.

## REFERENCE PREPPING

Ecco una lista di funzioni prepping disponibili all'uso:

Nome	Parametro	Descrizione
xss_clean	No	Esegue i dati attraverso la funzione XSS filtering descritta nella classe Input
prep_for_form	No	Converte i caratteri speciali come quelli dell'HTML per essere visualizzati nei campi di un form senza interruzioni
prep_url	No	Aggiunge http:// all'URL se manca
strip_image_tags	No	Elimina il codice HTML dal tag immagine, per ottenere il solo URL
encode_php_tags	No	Converte i tag PHP in entità

É anche possibile utilizzare qualsiasi funzione nativa [PHP](#) che si basa su un parametro come **trim**, **htmlspecialchars**, **urlencode**, ecc.

## REFERENCE DELLE FUNZIONI

Le seguenti funzioni sono da utilizzarsi all'interno dei controller.

- `$this->form_validation->set_rules()` permette di definire un gruppo di regole di validazione come descritto qui sopra.
- `$this->form_validation->run()` esegue le routine di validazione. Restituisce un valore booleano TRUE in caso di successo oppure FALSE in caso contrario. Opzionalmente è consentito passare il nome del gruppo di validazione tramite una funzione.
- `$this->form_validation->set_message()` consente di definire un messaggio di errore predefinito.

## REFERENCE HELPER

Qui di seguito si elencano una serie di helper che possono essere utilizzati nelle proprie Viste che contengono form. Si noti che si tratta di funzioni procedurali che quindi non richiedono di anteporre `$this->form_validation`.

- `form_error()` visualizza un messaggio di errore associato ad un campo specifico fornito alla funzione. I delimitatori degli errori possono essere opzionalmente specificati. Per esempio:

```
<?php echo form_error('username'); ?>
```

- `validation_errors()` visualizza tutti gli errori come una stringa. I delimitatori degli errori possono essere opzionalmente specificati. Per esempio:

```
<?php echo validation_errors(); ?>
```

- `set_value()` consente di definire il valore di un input o di una textarea in un form. È necessario fornire il nome del campo come primo parametro della funzione. Il secondo (opzionale) parametro consente di definire il valore di default del form. Per esempio:

```
<input type="text" name="quantity" value="<?php echo set_value('quantity', '0'); ?>" size="50" />
```

Il precedente form visualizzerà “0” (zero) quando sarà caricato per la prima volta.

- `set_select()` se si utilizza un menu `<select>` questa funzione visualizzerà gli elementi (del menu) che sono stati selezionati. Il primo parametro deve contenere il nome del menu select, mentre il secondo il valore di ogni elemento. Il terzo (sempre opzionale) parametro consente di definire quale elemento del menu sia quello predefinito (si utilizzino i valori booleani TRUE/FALSE). Per esempio:

```
<select name="myselect">
<option value="one" <?php echo set_select('myselect', 'one', TRUE); ?> >
    One</option>
<option value="two" <?php echo set_select('myselect', 'two'); ?> >Two</
    option>
<option value="three" <?php echo set_select('myselect', 'three'); ?> >
    Three</option>
</select>
```

- `set_checkbox()` visualizza un checkbox nello stato in cui viene inviato al server. Il primo parametro deve contenere il nome del checkbox, mentre il secondo parametro, il suo valore. Il terzo parametro (opzionale) consente di definire un elemento del checkbox predefinito (si utilizzino i valori booleani TRUE/FALSE). Per esempio:

```
<input type="checkbox" name="mycheck[]" value="1" <?php echo set_checkbox(
    'mycheck[]', '1'); ?> />
<input type="checkbox" name="mycheck[]" value="2" <?php echo set_checkbox(
    'mycheck[]', '2'); ?> />
```

- `set_radio()` i pulsanti radio saranno visualizzati nello stato in cui vengono inviati al server. Questa funzione è identica a `set_checkbox()` vista precedentemente:

```
<input type="radio" name="myradio" value="1" <?php echo set_radio('
    myradio', '1', TRUE); ?> />
<input type="radio" name="myradio" value="2" <?php echo set_radio('
    myradio', '2'); ?> />
```

## 8.12 CLASSE FTP

La classe di CodeIgniter `acFTP` è dedicata alla gestione dei trasferimenti tra host remoti: i file che si trovano su un altro server possono essere così spostati, rinominati e cancellati. La classe in oggetto inoltre include una funzione di mirroring che permette ad una directory locale di essere ricreata in remoto tramite [FTP](#).

Nota: attualmente i protocolli “SFTP” e “SSL FTP” non sono supportati.

La classe è inizializzata tramite il proprio controller utilizzando la funzione `$this->load->library`:

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

Una volta caricato, l'oggetto FTP sarà disponibile utilizzando `$this->ftp`.

### ESEMPI DI UTILIZZO

In questo esempio viene aperta una connessione verso il server [FTP](#) e un file locale viene letto e uploadato in modalità American Standard Code for Information Interchange ([ASCII](#)). I permessi del file devono essere impostati su 755.

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

$config['hostname'] = 'ftp.example.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['debug'] = TRUE;

$this->ftp->connect($config);

$this->ftp->upload('/local/path/to/myfile.html', '/public_html/myfile.html', '
    ascii', 0775);

$this->ftp->close();
```

Il codice seguente mostra come una lista di file possa essere recuperata dal server.

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

$config['hostname'] = 'ftp.example.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['debug'] = TRUE;

$this->ftp->connect($config);

$list = $this->ftp->list_files('/public_html/');

print_r($list);

$this->ftp->close();
```

In questo esempio viene effettuato un mirror di una directory locale sul server.

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

$config['hostname'] = 'ftp.example.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['debug'] = TRUE;

$this->ftp->connect($config);

$this->ftp->mirror('/path/to/myfolder/', '/public_html/myfolder/');

$this->ftp->close();
```

## REFERENCE DELLE FUNZIONI

- `$this->ftp->connect()` consente di connettersi e loggarsi ad un server FTP. Le impostazioni della connessione sono passate alla funzione mediante un array, oppure memorizzate in un file config. Qui di seguito si mostra come impostare le preferenze di connessione manualmente:



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

$config['hostname'] = 'ftp.example.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['port']     = 21;
$config['passive']  = FALSE;
$config['debug']    = TRUE;

$this->ftp->connect($config);
```

Se si preferisce memorizzare le impostazioni di connessione FTP in un file config, è sufficiente creare un nuovo file chiamato **ftp.php** e aggiungere al suo interno un array **\$config**. Questo file andrà salvato nel percorso **/config/ftp.php** e verrà automaticamente utilizzato.

Le opzioni disponibili per la connessione sono:

- hostname. È l'indirizzo FTP che viene solitamente espresso nel formato: ftp.esempio.com
  - username. Il nome di login FTP
  - password. La password FTP
  - port. Il numero di porta (il valore di default è 21)
  - debug. Abilita il debug per visualizzare i messaggi di errore (valori: TRUE/FALSE)
  - passive. Abilita la modalità passiva che è quella predefinita (valori: TRUE/FALSE)
- **\$this->ftp->upload()** consente di caricare un file sul proprio server. È necessario fornire un percorso locale ed uno remoto. Opzionalmente si può definire la modalità di upload ed i permessi. Per esempio:

```
$this->ftp->upload('/local/path/to/myfile.html', '/public_html/myfile.
html', 'ascii', 0775);
```

Le opzioni per la modalità di upload sono: **ascii**, **binary** e **auto** (valore di default). Se viene utilizzato il valore **auto**, la modalità sarà impostata sulla base dell'estensione del file locale. I permessi possono essere passati in base ottale con quattro cifre.

- **\$this->ftp->download()** è utilizzata per scaricare un file dal server remoto. È necessario fornire un percorso locale e uno remoto e, opzionalmente, definire la modalità. Questa funzione restituisce **FALSE** se il download non viene eseguito correttamente (include anche il caso in cui **PHP** non ha i permessi per scrivere un file localmente). Per esempio:

```
$this->ftp->download('/public_html/myfile.html', '/local/path/to/myfile.html', 'ascii');
```

Le opzioni per la modalità di upload sono: `ascii`, `binary` e `auto` (valore di default). Se viene utilizzato il valore `auto`, la modalità sarà impostata sulla base dell'estensione del file locale.

- `$this->ftp->rename()` consente di rinominare un file. Si deve fornire il nome/percorso del file attuale e il nuovo nome/percorso.

```
// Rinomina green.html in blue.html  
$this->ftp->rename('/public_html/foo/green.html', '/public_html/foo/blue.html');
```

- `$this->ftp->move()` sposta un file definendo il percorso del file attuale e quello di destinazione. Se il nome del file specificato nella "destinazione" è differente da quello iniziale, il file viene rinominato.

```
// Sposta blog.html da "joe" in "fred"  
$this->ftp->move('/public_html/joe/blog.html', '/public_html/fred/blog.html');
```

- `$this->ftp->delete_file()` è utilizzata per cancellare un file definendo il suo percorso iniziale e quello di destinazione (con il nome del file):

```
is->ftp->delete_file('/public_html/joe/blog.html');
```

`$this->ftp->delete_dir()` cancella una directory e ogni cosa al suo interno. È necessario fornire il percorso di origine della directory in oggetto con uno slash (/) finale:

```
$this->ftp->delete_dir('/public_html/path/to/folder/');
```

Nota: È molto importante prestare la massima attenzione nell'utilizzare questa funzione. La sua natura ricorsiva consente di cancellare qualsiasi elemento all'interno del percorso specificato, comprese le sottocartelle e tutti i file. Assicurarsi che il percorso sia corretto utilizzando la funzione `list_files()`.

- `$this->ftp->list_files()` consente di ottenere una lista di file sul server remoto sotto forma di un array. Si deve fornire il percorso della directory in oggetto:

```
$list = $this->ftp->list_files('/public_html/');  
  
print_r($list);
```

- `$this->ftp->mirror()` permette la creazione di un mirror. Questa funzione legge ricorsivamente una directory locale e tutto quello che contiene (comprese eventuali sottocartelle) e crea una sua copia sul server remoto. È necessario fornire il percorso della sorgente e della destinazione:

```
$this->ftp->mirror('/path/to/myfolder/', '/public_html/myfolder/');
```

- `$this->ftp->mkdir()` crea una directory sul server remoto. Si deve fornire il percorso finale della directory in oggetto compreso di uno slash (/) finale. I permessi, definiti in base ottale, possono essere passati come secondo parametro alla funzione:

```
// Crea una directory di nome "bar"  
$this->ftp->mkdir('/public_html/foo/bar/', DIR_WRITE_MODE);
```

- `$this->ftp->chmod()` consente di definire i permessi dei file. Si deve fornire il percorso del file o della directory di cui si vogliono modificare i permessi di lettura/scrittura/esecuzione:

```
// Chmod "bar" impostato sui permessi 777  
$this->ftp->chmod('/public_html/foo/bar/', DIR_WRITE_MODE);
```

- `$this->ftp->close()` questa funzione termina la connessione al server. È consigliabile utilizzarla sempre quando si conclude l'operazione di upload.

## 8.13 CLASSE HTML TABLE

Questa classe aiuta nella definizione e implementazione delle tabelle tramite un array di dati, oppure una lista di risultati forniti da un database. La classe è inizializzata tramite il proprio controller utilizzando la funzione **\$this->load->library**:

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

Una volta caricata, l'oggetto della libreria Table è disponibile utilizzando **\$this->table**. Nel seguente codice viene creata una tabella partendo da un array multidimensionale. Si noti che il primo indice dell'array diventerà la testata della tabella; per chi lo desiderasse è possibile definire una testata personalizzata utilizzando la funzione `set_heading()` descritta nel reference della classe.

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

$data = array(
    array('Name', 'Color', 'Size'),
    array('Fred', 'Blue', 'Small'),
    array('Mary', 'Red', 'Large'),
    array('John', 'Green', 'Medium')
);

echo $this->table->generate($data);
```

L'esempio successivo mostra una tabella creata a partire dal risultato di una query di un database. La classe Table genera automaticamente la testata della tabella; inoltre è possibile definire una testata personalizzata utilizzando la funzione `set_heading()` descritta nel reference della classe.

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

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

echo $this->table->generate($query);
```

Una tabella creata utilizzando singoli parametri, invece si presenta con il codice:

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

$this->table->set_heading('Name', 'Color', 'Size');

$this->table->add_row('Fred', 'Blue', 'Small');
$this->table->add_row('Mary', 'Red', 'Large');
$this->table->add_row('John', 'Green', 'Medium');

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

Ecco lo stesso esempio, tranne che invece di singoli parametri, vengono utilizzati gli array:

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

$this->table->set_heading(array('Name', 'Color', 'Size'));

$this->table->add_row(array('Fred', 'Blue', 'Small'));
$this->table->add_row(array('Mary', 'Red', 'Large'));
$this->table->add_row(array('John', 'Green', 'Medium'));

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

## CAMBIARE L'ASPETTO DELLA TABELLA

La classe Table definisce nei minimi dettagli il layout della tabella. Qui di seguito viene descritto il prototipo del template:

```

$tpl = 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>'
);

$this->table->set_template($tpl);

```

Si può notare come compaiano due blocchi di righe (row): questo permette di creare righe dai colori alternati, oppure di mettere in evidenza degli elementi che si alternano con ogni interazione. Non è quindi necessario fornire un template completo. Se si ha bisogno di cambiare solo alcuni aspetti del layout della tabella, si possono semplicemente ridefinire questi elementi. Per esempio, se si vuole solo personalizzare il tag di apertura della tabella:

```

$tpl = array ( 'table_open' => '<table border="1" cellpadding="2" cellspacing="1" class="mytable">' );

$this->table->set_template($tpl);

```

## REFERENCE

- `$this->table->generate()` restituisce una stringa che contiene la tabella generata. Accetta un parametro opzionale che può essere un array o un risultato oggetto di un database.
- `$this->table->set_caption()` permette di aggiungere una didascalia (caption) alla tabella:

```
$this->table->set_caption('Colors');
```

- `$this->table->set_heading()` definisce il titolo della tabella. È possibile definire un array o singoli parametri:

```
$this->table->set_heading('Name', 'Color', 'Size');
```

```
$this->table->set_heading(array('Name', 'Color', 'Size'));
```

`$this->table->add_row()` aggiunge una riga alla tabella. È possibile definire un array o singoli parametri:

```
$this->table->add_row('Blue', 'Red', 'Green');
```

```
$this->table->add_row(array('Blue', 'Red', 'Green'));
```

Se si vogliono definire gli attributi di un tag di una ben definita cella, si può utilizzare un array. La chiave associativa **data** definisce per l'appunto l'informazione della cella. Tutte le altre coppie "chiave => valore" sono aggiunte come un attributo al tag, nel formato "key='val'":

```
$cell = array('data' => 'Blue', 'class' => 'highlight', 'colspan' => 2);
$this->table->add_row($cell, 'Red', 'Green');

// Genera
// <td class='highlight' colspan='2'>Blue</td><td>Red</td><td>Green</td>
```

- `$this->table->make_columns()` questa funzione prende in ingresso un array (monodimensionale) e restituisce un array multidimensionale con una profondità uguale al numero di colonne desiderato. Questo permette ad un singolo array con diversi elementi di essere visualizzato in una tabella con un numero fisso di colonne. Si consideri il seguente esempio:

```

$list = array('one', 'two', 'three', 'four', 'five', 'six', 'seven', '
    eight', 'nine', 'ten', 'eleven', 'twelve');

$new_list = $this->table->make_columns($list, 3);

$this->table->generate($new_list);

// Genera una tabella con questo prototipo

<table border="0" cellpadding="4" cellspacing="0">
<tr>
<td>one</td><td>two</td><td>three</td>
</tr><tr>
<td>four</td><td>five</td><td>six</td>
</tr><tr>
<td>seven</td><td>eight</td><td>nine</td>
</tr><tr>
<td>ten</td><td>eleven</td><td>twelve</td></tr>
</table>

```

`$this->table->set_template()` consente di definire il proprio template. Si può modificare un intero template oppure solo una sua parte:

```

$tpl = array ( 'table_open' => '<table border="1" cellpadding="2"
    cellspacing="1" class="mytable">' );

$this->table->set_template($tpl);

```

- `$this->table->set_empty()` definisce un valore predefinito da utilizzare in ogni cella “vuota” della tabella. Si potrebbe, ad esempio, impostare un spazio unificatore:

```

$this->table->set_empty("&nbsp;");

```

- `$this->table->clear()` consente di pulire il titolo della tabella e le informazioni contenute nelle righe. Questa funzione è particolarmente utile se si ha la necessità di visualizzare più tabelle con differenti dati. Chiamando questa funzione subito dopo ogni tabella, questa verrà svuotata dei contenuti. Per esempio:



```

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

$this->table->set_heading('Name', 'Color', 'Size');
$this->table->add_row('Fred', 'Blue', 'Small');
$this->table->add_row('Mary', 'Red', 'Large');
$this->table->add_row('John', 'Green', 'Medium');

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

$this->table->clear();

$this->table->set_heading('Name', 'Day', 'Delivery');
$this->table->add_row('Fred', 'Wednesday', 'Express');
$this->table->add_row('Mary', 'Monday', 'Air');
$this->table->add_row('John', 'Saturday', 'Overnight');

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

```

`$this->table->function` consente di specificare una funzione nativa [PHP](#) oppure un oggetto array da applicare ai dati di tutte le celle:

```

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

$this->table->set_heading('Name', 'Color', 'Size');
$this->table->add_row('Fred', '<strong>Blue</strong>', 'Small');

$this->table->function = 'htmlspecialchars';
echo $this->table->generate();

```

Nell'esempio precedente, tutti i dati delle celle verranno eseguite con la funzione del [PHP htmlspecialchars\(\)](#) come risulta dall'istruzione:

```

<td>Fred</td><td>&lt;strong&gt;Blue&lt;/strong&gt;</td><td>Small</td>

```

## 8.14 CLASSE IMAGE MANIPULATION

Questa classe è di prezioso aiuto nelle operazioni sui formati grafici fornendo importanti strumenti per la manipolazione delle immagini. Ecco un elenco sommario, che verrà esaminato nel dettaglio nel corso del capitolo:

- ridimensionamento
- creazione di miniature
- ritaglio
- rotazione
- filigrana (watermark)

Sono supportate tutte le tre maggiori librerie, ovvero GD/GD2, NetPBM, e ImageMagick anche se la funzione di filigrana non è disponibile con la GD/GD2. Inoltre, anche se altre librerie sono supportate, GD è sempre necessaria affinché lo script per calcolare le proprietà dell'immagine funzioni correttamente. L'elaborazione delle immagini, comunque, verrà eseguita con la libreria specificata.

La classe viene inizializzata nel proprio controller con il metodo `$this->load->library`

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

Una volta che la libreria è caricata, sarà pronta per l'uso, semplicemente richiamando la funzione `$this->image_lib`.

Indipendentemente dal tipo di operazione che si desidera eseguire (ridimensionamento, ritaglio, rotazione, o filigrana), il processo per eseguirle è identico. Si impostano alcune preferenze corrispondenti all'azione che si intende eseguire, quindi si chiama una delle quattro funzioni di elaborazione disponibili. Ad esempio, per creare una miniatura dell'immagine:

```
$config['image_library'] = 'gd2';
$config['source_image'] = '/path/to/image/mypic.jpg';
$config['create_thumb'] = TRUE;
$config['maintain_ratio'] = TRUE;
$config['width'] = 75;
$config['height'] = 50;

$this->load->library('image_lib', $config);

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

Il codice chiama la funzione `image_resize()` per visualizzare un'immagine chiamata "mypic.jpg" situata nella cartella `source_image`, quindi crea una miniatura di 75x50 pixel utilizzando la funzione `image_library` di GD2. Sin quando è abilitata

l'opzione `maintain_ratio` i parametri `width` e `height` dell'immagine saranno inaccessibili in modo da garantire le corrette proporzioni. La miniatura dell'immagine si chiamerà `mypic_thumb.jpg`.

Nota: per essere autorizzati a compiere qualsiasi elaborazione con la classe `Image`, la cartella contenente i file grafici deve avere abilitati i permessi di scrittura. Inoltre l'elaborazione delle immagini può richiedere una considerevole quantità di memoria del server per alcune operazioni. Se si verificano errori di memoria durante l'elaborazione delle immagini può essere necessaria limitare la loro dimensione massima, e/o regolare adeguatamente i limiti di memoria [PHP](#).

## FUNZIONI DI ELABORAZIONE IMMAGINE

CodeIgniter rende disponibile cinque funzioni adibite all'elaborazione delle immagini.

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

Queste funzioni restituiscono un valore booleano a seconda del successo o meno dell'operazione impostata. In caso di insuccesso (valore `FALSE`) è possibile recuperare il messaggio di errore utilizzando la funzione:

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

Una buona abitudine è quella di utilizzare le funzioni di elaborazione delle immagini all'interno di istruzioni condizionali che permettano di visualizzare l'errore in caso di insuccesso. Per esempio:

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

Nota: Si può specificare la formattazione [HTML](#) da applicare agli errori, specificando i tag di apertura/chiusura nella funzione, nel seguente modo:

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

## PREFERENZE

Le preferenze descritte di seguito permettono di definire le operazioni di elaborazione dell'immagine nei minimi dettagli. Si presti comunque attenzione al fatto che non tutte le impostazioni sono disponibili per ogni funzione. Per esempio la preferenza sugli assi  $x/y$  è utilizzabile solo per il ritaglio dell'immagine. Allo stesso modo le impostazioni in altezza e larghezza non hanno alcun effetto se applicate all'operazione di ritaglio dell'immagine. La colonna con l'attributo "disponibilità" indica che la funzione supporta le preferenze specificate:

### Legenda

- R: ridimensionamento dell'immagine

- C: ritaglio dell'immagine

- X: rotazione dell'immagine

- W: filigrana nell'immagine

Preferenze	Default	Opzioni	Descrizione	Disponibilità
<code>image_library</code>	GD2	GD, GD2, ImageMagick, NetPBM	Imposta la libreria di immagini da utilizzare.	R, C, X, W
<code>library_path</code>	Nessuno	Nessuno	Imposta il percorso del server alla libreria ImageMagick o NetPBM. Se si utilizza una di queste è necessario fornire il percorso.	R, C, X
<code>source_image</code>	Nessuno	Nessuno	Imposta l'immagine di origine nome/path. Il percorso deve essere un percorso di server relativo o assoluto, non un URL.	R, C, S, W
<code>dynamic_output</code>	FALSE	TRUE/FALSE	Determina se il nuovo file immagine deve essere scritto su disco o generato dinamicamente. Nota: Se si sceglie l'impostazione dinamica, verrà visualizzata solo un'immagine alla volta, e non sarà posizionata sulla pagina. Produce semplicemente l'immagine raw (grezza) inviata dinamicamente al browser, insieme con le intestazioni dell'immagine.	R, C, X, W
<code>quality</code>	90%	1 - 100%	Imposta la qualità dell'immagine. Più alta è la qualità, più grande è la dimensione del file.	R, C, X, W
<code>new_image</code>	Nessuno	Nessuno	Imposta l'immagine di destinazione nome/path. Si potrà utilizzare questa opzione durante la creazione di una copia dell'immagine. Il percorso deve essere un percorso di server relativo o assoluto, non un URL.	R, C, X, W
<code>width</code>	Nessuno	Nessuno	Imposta la larghezza dell'immagine.	R, C
<code>height</code>	Nessuno	Nessuno	Imposta l'altezza dell'immagine.	R, C
<code>create_thumb</code>	FALSE	TRUE/FALSE	Indica la funzione di elaborazione delle immagini per creare una miniatura.	R
<code>thumb_marker</code>	<code>_thumb</code>	Nessuno	Specifica l'indicatore di miniatura che sarà inserito appena prima dell'estensione del file: per esempio <code>mypic.jpg</code> diventa <code>mypic_thumb.jpg</code>	R
<code>maintain_ratio</code>	TRUE	TRUE/FALSE	Specifica se mantenere le proporzioni originali durante il ridimensionamento o utilizzare i valori principali.	R, C
<code>master_dim</code>	auto	auto, width, height	Specifica cosa utilizzare come asse principale durante il ridimensionamento o la creazione della miniatura. Se la dimensione originale dell'immagine non consente un perfetto ridimensionamento, questa impostazione determina l'asse che dovrebbe essere utilizzata come valore. "Auto" imposta l'asse automaticamente in base al fatto che l'immagine sia più alta che larga, o viceversa.	R
<code>rotation_angle</code>	Nessuno	90, 180, 270, vrt, hor	Specifica l'angolo di rotazione delle immagini. Si noti che PHP le ruota in senso antiorario, quindi una rotazione di 90 gradi verso destra deve essere specificata come 270.	X
<code>x_axis</code>	Nessuno	Nessuno	Imposta la coordinata X in pixel per il ritaglio (crop) dell'immagine. Ad esempio, un'impostazione di 30 permetterà di ritagliare un'immagine di 30 pixel partendo da sinistra.	C
<code>y_axis</code>	Nessuno	Nessuno	Imposta la coordinata Y in pixel per il ritaglio dell'immagine. Ad esempio, un'impostazione di 30 permetterà di ritagliare un'immagine di 30 pixel partendo dall'alto.	C

## IMPOSTAZIONE DELLE PREFERENZE

Se si preferisce definire le impostazioni alternativamente attraverso un file config, si può creare un file `image_lib.php` definendo al suo interno un array `$config`. Il file così creato andrà salvato nel percorso `config/image_lib.php` e verrà automaticamente utilizzato. Non è necessario utilizzare la funzione `$this->image_lib->initialize` se si definiscono le preferenze nel file config.

- `$this->image_lib->resize()` questa funzione svolge il compito di ridimensionare l'immagine creando una sua copia (con o senza ridimensionamento), oppure definendo una sua miniatura. In pratica non vi sono differenze tra creare una copia dell'immagine o creare una sua miniatura, ad eccezione del fatto che quest'ultima avrà un simbolo con il nome (per esempio `mypic_thumb.jpg`).

Tutte le preferenze elencate nella tabella precedente sono disponibili, ad eccezione di queste tre: `rotation_angle`, `x_axis` e `y_axis`.

Se si utilizza la funzione di ridimensionamento per creare una miniatura (preservando le dimensioni originali) è necessario impostare le preferenze su `TRUE`:

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

Se si utilizza la funzione di ridimensionamento per creare una copia dell'immagine (preservando le dimensioni originali) si deve definire un nuovo percorso per la nuova immagine:

```
$config['new_image'] = '/path/to/new_image.jpg';
```

- Se si indica solo il nome della nuova immagine, questa verrà inserita nella stessa directory dell'immagine originale
- Se si indica solo il percorso, la nuova immagine sarà posta nel percorso di destinazione con lo stesso nome dell'immagine originale
- Se è presente sia il nuovo nome che il percorso, la nuova immagine sarà posta nel path specificato con il nome indicato

Se nessuna delle due opzioni sopra elencate (`create_thumb` e `new_image`) vengono utilizzate, verrà eseguita la funzione di ridimensionamento.

- `$this->image_lib->crop()` la funzione di ritaglio funziona nello stesso modo di quella per il ridimensionamento tranne per il fatto che essa richiede di definire l'asse X e Y in pixel per specificare dove ritagliare l'immagine. Per esempio:

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

Tutte le impostazioni definite nella tabella precedente sono disponibili ad eccezione delle: `rotation_angle`, `width`, `height`, `create_thumb`, `new_image`.

Qui di seguito vi è un esempio di come viene compiuta l'operazione su di una immagine. Si noti comunque che senza visualizzare l'immagine è arduo definire le coordinate per il ritaglio e questo rende la funzione poco pratica. É quindi consigliato utilizzarla con il modulo `gallery` presente in `ExpressionEngine`: in esso è presente una interfaccia utente JavaScript che permette di selezionare l'area di ritaglio.

- `$this->image_lib->rotate()` la funzione richiede l'angolo di rotazione dell'immagine che si può impostare con l'istruzione:

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

Esistono cinque gradi di rotazione utilizzabili:

- 90 gradi in senso orario
- 180 gradi in senso orario
- 270 gradi in senso orario
- hor: ribalta l'immagine orizzontalmente
- vrt: ribalta l'immagine verticalmente

Qui di seguito si mostra il codice di esempio che prevede la rotazione di una immagine:

```
$config['image_library'] = 'netpbm';  
$config['library_path'] = '/usr/bin/';  
$config['source_image'] = '/path/to/image/mypic.jpg';  
$config['rotation_angle'] = 'hor';  
  
$this->image_lib->initialize($config);  
  
if ( ! $this->image_lib->rotate()  
{  
    echo $this->image_lib->display_errors();  
}
```

- `$this->image_lib->clear()` questa funziona resetta (pulisce) tutti i valori utilizzati nel modificare una immagine. Essa viene utilizzata se più immagini vengono elaborate tramite un ciclo:

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

## FILIGRANA

Esistono due tipi di filigrana utilizzabili: una che prevede un testo, e l'altra che utilizza una immagine. Questa operazione richiede obbligatoriamente la libreria GD/GD2.

- text: la filigrana è un messaggio testuale per cui si utilizza il font True Type desiderato oppure un qualsiasi formato testo supportato dalla libreria GD. Se si utilizza la versione True Type, la propria installazione GD deve essere compilata con il supporto ai font True Type
- overlay: in questo caso la filigrana è composta da una nuova immagine che viene sovrapposta a quella originale. Solitamente si utilizza una immagine con formato PNG oppure GIF che gestiscono la trasparenza dello sfondo

Proprio come con le altre funzioni (ridimensionamento, ritaglio e rotazione), il processo generale di filigrana consiste nell'impostare le preferenze corrispondenti all'azione che si intende eseguire. Ecco un esempio:

```
$config['source_image'] = '/path/to/image/mypic.jpg';  
$config['wm_text'] = 'Copyright 2014 - Giuseppe Bellisano';  
$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();
```

L'esempio precedente utilizza un font True Type di 16 pixel per creare un testo con la dicitura "Copyright 2014 - Giuseppe Bellisano". La filigrana sarà posizionata in basso, al centro dell'immagine principale con 20 pixel di spazio dal bordo inferiore.

Nota: per permettere l'elaborazione dell'immagine è necessario impostare i permessi dell'immagine in modo da consentire la scrittura (per esempio 777).



## PREFERENZE FILIGRANA

Preferenze	Default	Opzioni	Descrizione
<code>wm_type</code>	text	text, overlay	Imposta il tipo di filigrana.
<code>source_image</code>	Nessuno	Nessuno	Imposta l'immagine di origine nome/path che deve essere un percorso relativo o assoluto di server, non un URL.
<code>dynamic_output</code>	FALSE	TRUE/FALSE	Determina se il nuovo file immagine deve essere scritto sull'hard disk o generato dinamicamente. Nota: Se si sceglie l'impostazione dinamica, può essere visualizzata solo un'immagine alla volta, e non può essere posizionata sulla pagina. Invia semplicemente l'immagine raw (grezza) al browser, insieme con le intestazioni di immagine.
<code>quality</code>	90%	1 - 100%	Imposta la qualità dell'immagine: più alta è la qualità, più grande risulterà la dimensione del file.
<code>padding</code>	Nessuno	Un numero	Imposta il padding, in pixel, che sarà applicato alla filigrana per impostare lo spazio dal bordo delle immagini.
<code>wm_vrt_alignment</code>	bottom	top, middle, bottom	Imposta l'allineamento verticale per la filigrana.
<code>wm_hor_alignment</code>	center	left, center, right	Imposta l'allineamento orizzontale per la filigrana.
<code>wm_hor_offset</code>	Nessuno	Nessuno	È possibile specificare un offset orizzontale (in pixel) da applicare alla posizione della filigrana. L'offset normalmente sposta la filigrana a destra, tranne se l'allineamento è impostato sulla destra; in questo caso il valore di offset sposterà la filigrana verso la sinistra dell'immagine.
<code>wm_vrt_offset</code>	Nessuno	Nessuno	È possibile specificare un offset orizzontale (in pixel) da applicare alla posizione della filigrana. L'offset normalmente sposta la filigrana in basso, tranne se l'allineamento è impostato su in basso; in questo caso il valore di offset sposterà la filigrana verso la parte alta dell'immagine.

## PREFERENZE FILIGRANA CON TESTO

Preferenze	Default	Opzioni	Descrizione
<code>wm_text</code>	Nessuno	Nessuno	Il testo mostrato come la filigrana. Solitamente è un avviso di copyright.
<code>wm_font_path</code>	Nessuno	Nessuno	Il percorso del server al font True Type che si desidera utilizzare. Se non si utilizza questa opzione, verrà utilizzato il tipo di carattere GD nativo.
<code>wm_font_size</code>	16	Nessuno	La dimensione del testo. Nota: Se non si utilizza l'opzione True Type di qui sopra, il numero è impostato con un intervallo 1-5. Altrimenti, è possibile utilizzare qualsiasi dimensione, in pixel, valida per il tipo di carattere che si sta utilizzando.
<code>wm_font_color</code>	ffffff	Nessuno	Il colore del carattere, specificato in esadecimale. Nota, è necessario utilizzare il valore composto di 6 caratteri esadecimale (ad esempio, 993300), piuttosto che la versione abbreviata di tre caratteri (cioè fff).
<code>wm_shadow_color</code>	Nessuno	Nessuno	Il colore dell'ombra esterna, specificato in esadecimale. Se si lascia questo campo vuoto non verrà utilizzato alcuna ombra. Nota, è necessario utilizzare il valore composto di 6 caratteri esadecimale (ad esempio, 993300), piuttosto che la versione abbreviata di tre caratteri (cioè fff).
<code>wm_shadow_distance</code>	3	Nessuno	La distanza (in pixel) dal carattere per cui deve apparire l'ombra.

## PREFERENZE FILIGRANA CON IMMAGINE

<code>wm_overlay_path</code>	Nessuno	Nessuno	Il percorso del server per l'immagine che si desidera utilizzare come filigrana. Necessaria solo se si utilizza il metodo di sovrapposizione.
<code>wm_opacity</code>	50	1 - 100	Opacità dell'immagine. È possibile specificare l'opacità (cioè trasparenza) dell'immagine filigrana. Questo permette di non nascondere i dettagli della immagine originale. Solitamente il suo valore è del 50%.
<code>wm_x_transp</code>	4	Un numero	Se l'immagine filigrana è una immagine PNG o GIF, è possibile specificare un colore per lo sfondo che la renda trasparente. Questa impostazione (insieme con la prossima) permette di specificare quel colore, specificando le coordinate in pixel X e Y (misurate a partire dall'angolo in alto a sinistra) che corrispondono al pixel con il colore da rendere trasparente.
<code>wm_y_transp</code>	4	Un numero	Insieme con l'impostazione precedente, consente di specificare le coordinate di un pixel il cui colore che si desidera rendere trasparente.

## MODIFICARE LE IMMAGINI IN SEQUENZA

Se si volessero apportare più modifiche in sequenza ad o più immagini, si dovranno resettare le impostazioni della libreria attraverso l'istruzione:

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

Questa andrà eseguita prima di inizializzare nuovamente la libreria con la configurazione desiderata. Qui di seguito si mostra un esempio con due operazioni in successione sulla stessa immagine: una rotazione seguita da un ridimensionamento. Si noti che dopo la prima operazione, la libreria viene resettata e una volta definiti i parametri della seconda operazione, nuovamente inizializzata.

```
//Impostazioni per rotazione verticale
$config['image_library'] = 'gd2';
$config['source_image'] = '/percorso/miaimmagine.jpg';
$config['rotation_angle'] = 'vrt';

// Viene caricata la libreria con la configurazione
$this->load->library('image_lib', $config);

// Viene eseguita la rotazione
$this->image_lib->rotate();

// Le impostazioni della libreria vengono cancellate
$this->image_lib->clear();

// Vengono definiti i parametri per il cropping
$config['image_library'] = 'gd2';
$config['source_image'] = '/percorso/miaimmagine.jpg';
$config['x_axis'] = '50';
$config['y_axis'] = '50';

// La libreria con la configurazione viene inizializzata
$this->image_lib->initialize($config);

// Infine si esegue il ridimensionamento
$this->image_lib->crop();
```

## 8.15 CLASSE INPUT

La classe Input svolge due importanti compiti:

- effettua una elaborazione preventiva (pre-processing) dei dati per la sicurezza
- fornisce alcune funzioni helper per recuperare i dati di input ed elaborarli anticipatamente

Nota: Questa classe viene inizializzata automaticamente dal sistema in modo che non sia necessario farlo manualmente.

### SECURITY FILTERING

La funzione di filtro di sicurezza viene chiamata automaticamente quando un nuovo controller viene invocato.

- se `$config['allow_get_array']` assume il valore FALSE (quello di default è TRUE), distrugge l'array globale GET
- elimina tutte le variabili globali nel `register_globals` se abilitato
- filtra le chiavi dell'array GET/POST/COOKIE permettendo unicamente caratteri alfanumerici (e pochi altri)
- fornisce il filtro XSS che può essere abilitato globalmente o sotto richiesta
- standardizza i caratteri di ritorno a capo in `\n` (in Windows `\r\n`)

### FILTRO XSS

La classe Input permette di filtrare i dati di input automaticamente prevenendo così gli attacchi basati su cross-site scripting. Se si desidera abilitare il filtro per essere eseguito automaticamente ogni volta che si incontra un dato POST oppure COOKIE, è necessario modificare il file `/application/config/config.php`:

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

Si consiglia di leggere la documentazione della classe Security per maggiori informazioni [8.22 a pagina 320](#).

### DATI POST, COOKIE O SERVER

CodeIgniter è dotato di tre funzioni helper che consentono di ottenere elementi POST, COOKIE e SERVER. Il vantaggio principale nell'utilizzare le funzioni previste,

piuttosto che utilizzare le funzioni native PHP come (`$_POST['qualcosa']`) è che le funzioni di CodeIgniter verificano se l'elemento esiste: in caso contrario restituiscono FALSE. Ciò consente di utilizzare comodamente i dati senza dover verificare se un elemento esiste in primo luogo. In altre parole, normalmente si dovrebbe fare qualcosa di simile a questo:

```
if ( ! isset($_POST['qualcosa']))
{
    $something = FALSE;
}
else
{
    $something = $_POST['qualcosa'];
}
```

Con le funzioni di CodeIgniter invece il processo è molto più semplice:

```
$something = $this->input->post('qualcosa');
```

Le tre funzioni che verranno esaminate nel dettaglio sono:

- `$this->input->post()`
- `$this->input->cookie()`
- `$this->input->server()`

`$this->input->post()` il primo parametro della funzione contiene il nome dell'elemento POST che si sta cercando:

```
$this->input->post('qualche_dato');
```

La funzione restituisce FALSE se l'elemento che si vuole recuperare non esiste. Il secondo parametro (opzionale) permette di eseguire i dati attraverso il filtro XSS. Per abilitare quest'ultimo si imposti il secondo parametro su TRUE.

```
$this->input->post('qualche_dato', TRUE);
```

Se la funzione viene chiamata senza parametri, verrà restituito un array con tutti gli elementi POST. Per ottenere tutti gli elementi POST e passarli al filtro XSS si imposti il primo parametro della funzione su NULL. Questa restituirà FALSE se non ci sono elementi nel POST.

```
$this->input->post(NULL, TRUE); // restituisce tutti gli elementi POST con  
    filtro XSS  
$this->input->post(); // restituisce tutti gli elementi POST senza filtro XSS
```

`$this->input->get()` questa funzione è identica alla precedente, solo che viene utilizzata per recuperare i dati GET.

```
$this->input->get('qualche_dato', TRUE);
```

Per ottenere un array con tutti gli elementi GET chiamare la funzione senza alcun parametro. Per ottenere tutti gli elementi GET e passarli attraverso il filtro XSS, si imposti il primo parametro su NULL. La funzione restituirà FALSE se non ci sono elementi da recuperare.

```
$this->input->get(NULL, TRUE); // restituisce tutti gli elementi GET con  
    filtro XSS  
$this->input->get(); // restituisce tutti gli elementi GET senza filtro XSS
```

`$this->input->get_post()` questa funzione cercherà di recuperare i dati sia da GET che da POST, incominciando prima da quest'ultimo.

```
$this->input->get_post('qualche_dato', TRUE);
```

`$this->input->cookie()` questa funzione è identica alla funzione post, solo che viene utilizzata per recuperare i dati cookie:

```
$this->input->cookie('qualche_dato', TRUE);
```

`$this->input->server()` viene utilizzate per recuperare i dati server:

```
$this->input->server('qualche_dato');
```

`$this->input->set_cookie()` imposta un cookie contenente i valori specificati. Ci sono due modi per passare le informazioni a questa funzione in modo che un cookie possa essere impostato: Metodo Array e Parametri discreti:

Metodo Array

Il Metodo Array utilizza un array associativo passato come primo parametro:

```
$cookie = array(  
    'name' => 'Il nome del Cookie',  
    'value' => 'Il valore',  
    'expire' => '86500',  
    'domain' => '.un-dominio.com',  
    'path' => '/',  
    'prefix' => 'mioprefisso_',  
    'secure' => TRUE  
);  
  
$this->input->set_cookie($cookie);
```

Nota: solo il nome e il valore sono obbligatori. Per eliminare un cookie impostato si lasci vuoto il parametro “value”.

- La scadenza è impostata con un valore espresso in secondi, che deve essere sommato al tempo corrente. Se non si include il tempo, ma solo il numero di secondi, questi determineranno i secondi per i quali il cookie sarà valido. Se la scadenza è impostata a zero il cookie durerà solo fino a quando il browser rimarrà aperto
- Per i cookie del sito, a prescindere da come il sito viene “richiesto”, viene aggiunto al dominio il suo [URL](#) partendo da un periodo, come questo .tuo-dominio.com
- Il percorso non è di solito necessario in quanto la funzione imposta un percorso root
- Il prefisso è necessario solo se si vogliono evitare conflitti con altri cookie che hanno lo stesso nome sul server
- Il valore booleano **secure** è necessario solo se si vuole rendere un cookie sicuro impostandolo su TRUE

#### Parametri discreti

Se si preferisce, è possibile impostare i cookie passando i dati come singoli parametri:

```
$this->input->set_cookie($name, $value, $expire, $domain, $path, $prefix,  
    $secure);
```

`$this->input->cookie()` permette di recuperare un cookie. Il primo parametro conterrà il nome del cookie che si sta cercando (incluso qualsiasi prefisso).

```
cookie('qualche_cookie');
```



Questa funzione restituisce FALSE se l'elemento che si sta cercando di recuperare non esiste. Il secondo parametro permette di eseguire i dati attraverso il filtro XSS che verrà abilitato impostando il secondo parametro su TRUE.

```
cookie('qulche_cookie', TRUE);
```

`$this->input->ip_address()` restituisce l'indirizzo IP dell'utente corrente. Se l'indirizzo IP non è valido, la funzione restituisce l'IP 0.0.0.0.

```
echo $this->input->ip_address();
```

`$this->input->valid_ip($ip)` Prende un indirizzo IP in ingresso e restituisce un valore booleano TRUE/FALSE se è valido o meno. La funzione `$this->input->ip_address()` effettua la validazione dell'IP automaticamente.

```
if ( ! $this->input->valid_ip($ip) )
{
    echo 'Non valido';
}
else
{
    echo 'Valido';
}
```

È accettata opzionalmente una stringa come secondo parametro di IPv4 oppure IPv6 per specificare il formato IP. Per impostazione predefinita vengono verificati tutti e due i formati.

`$this->input->user_agent()` restituisce lo user agent (browser web) dell'utente corrente. Restituisce FALSE se non è disponibile.

```
echo $this->input->user_agent();
```

Per maggiori informazioni si guardi la classe User Agent a pagina [349](#).

`$this->input->request_headers()` È utile se si lavora in un ambiente diverso da Apache dove la funzione `apache_request_headers()` non è supportata. Restituisce un array di header.

```
$headers = $this->input->request_headers();
```

`$this->input->get_request_header()` restituisce un singolo elemento dell'array di header richiesto.

```
$this->input->get_request_header('qualche-header', TRUE);
```

`$this->input->is_ajax_request()` permette di verificare se l'header del server `HTTP_X_REQUESTED_WITH` è stato impostato e restituisce un valore booleano.

`$this->input->is_cli_request()` verifica se la costante `STDIN` è impostata, che è un modo intrinsecamente sicuro per vedere se PHP viene eseguito sulla riga di comando.

```
$this->input->is_cli_request()
```

## 8.16 CLASSE JAVASCRIPT

Nota: questo driver si trova in uno stato sperimentale e molte funzionalità potrebbero cambiare nelle future versioni.

CodeIgniter fornisce una libreria che aiuta lo sviluppatore ad implementare Javascript nel proprio lavoro. Non è comunque necessario installare la libreria jQuery per eseguire qualsiasi funzione presente nella libreria del framework. La classe viene inizializzata con la funzione `$this->load->library` nel proprio costruttore del controller. Attualmente l'unica libreria disponibile è jQuery la quale sarà automaticamente caricata con una istruzione simile alla seguente:

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

La classe Javascript accetta solo i parametri `js_library_driver` (string) default 'jquery' e `autoload` (bool) default TRUE. È anche possibile sovrascrivere le impostazioni predefinite, fornendo alla funzione un array associativo:

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

Anche in questo caso, solo jQuery è disponibile: se comunque non si desidera che la libreria includa automaticamente un tag script, si può impostare il parametro **autoload** sul valore FALSE. Questo è utile se si sta caricando lo script al di fuori di CodeIgniter, o se è già presente un tag script. Una volta caricato, sarà disponibile l'oggetto jQuery usando l'istruzione `$this->javascript`.

### INSTALLAZIONE E CONFIGURAZIONE

Poiché Javascript è un linguaggio client-side, la libreria deve essere in grado di scrivere contenuti nell'output finale del proprio progetto, in genere una Vista. Sarà necessario includere le seguenti variabili nelle sezioni <head> dell'output:

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

`$library_src` indica dove la libreria attuale sarà caricata in seguito ad una chiamata

`$script_head` indica dove gli eventi, le funzioni o altri comandi saranno finalizzati

## PERCORSO DELLE LIBRERIE

I parametri che definiscono la configurazione della libreria Javascript possono essere impostati nel file **/application/config.php** e **/config/javascript.php**, oppure in qualsiasi controller utilizzando la funzione `set_item()`.

Una immagine può essere usata come “ajax loader”, o come indicatore di progresso in modo da informare l’utente sull’avanzamento del caricamento. Senza uno di questi elementi, quando sarà fatta una chiamata Ajax verrà visualizzato semplicemente il messaggio “loading”.

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

Se si mantengono i propri file nella stessa directory da cui sono stati scaricati, allora non sarà necessario impostare alcun elemento di configurazione.

## LA CLASSE JQUERY

Per inizializzare la classe jQuery manualmente nel costruttore del controller si usa la funzione **`$this->load->library`**:

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

Si può specificare un ulteriore parametro opzionale per determinare se un tag script per il file principale jQuery dovrà essere incluso automaticamente durante il caricamento della libreria. Questo sarà creato per impostazione predefinita, ma può essere evitato caricando la libreria nel modo seguente:

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

Una volta caricata, l’oggetto library jQuery sarà disponibile utilizzando: **`$this->jquery`**.

## GLI EVENTI JQUERY

Gli eventi sono impostati con la seguente sintassi:

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

- `event` assume un valore tra `blur`, `change`, `click`, `dblclick`, `error`, `focus`, `hover`, `keydown`, `keyup`, `load`, `mousedown`, `mouseup`, `mouseover`, `mouseup`, `resize`, `scroll`, `unload`
- `element_path` indica qualsiasi selettore valido jQuery (si veda <http://api.jquery.com/category/selectors/>: questo di solito è un elemento id oppure un selettore CSS. Un esempio chiarificatore è dato da `# notice_area` che ha l'effetto di produrre `<div id="notice_area">`, mentre `"# content a.notice"` ha effetto su tutte le ancore (i collegamenti) con una classe di "notice" nel div con id "content".
- `code_to_run()` è lo script che si è sviluppati da soli, o un'azione Effect forniti dalla libreria jQuery

## EFFETTI

La libreria jQuery supporta un vasto elenco di potenti strumenti per aggiungere animazioni alle pagine web. Questi elementi, definiti Effect <http://api.jquery.com/category/effects/>, per essere utilizzati hanno bisogno di essere caricati:

```
$this->jquery->effect([optional path] plugin name); // for example $this->
jquery->effect('bounce');
```

- `hide()` / `show()` ciascuna di queste funzioni influenzerà la visibilità di un articolo sulla pagina: `hide()` imposterà un elemento invisibile, invece `show()` lo renderà visibile.

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

- `target`: qualsiasi selettore jQuery o selettore comune.
  - `speed`: è opzionale ed indica la velocità con delle parole chiave (`slow`, `normal`, `fast`) oppure indicando il numero di millisecondi desiderati.
  - `extra information`: è opzionale e può contenere un callback o altre informazioni aggiuntive.
- `toggle()` farà sì che un elemento assuma lo stato opposto a quello corrente.

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

Il parametro `target` potrà essere un selettore comune o uno jQuery

- `animate()` fornisce funzionalità legate all'animazione dell'elemento.

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

- `target`: qualsiasi selettore jQuery o selettore comune
- `parameters`: una serie di proprietà CSS che si desidera modificare
- `speed`: è opzionale ed indica la velocità con delle parole chiave (slow, normal, fast) oppure specificata con il numero di millisecondi desiderati
- `extra information`: è opzionale e può contenere una callback o altre informazioni aggiuntive

Qui di seguito si mostra un esempio della funzione **animate()** richiamata all'interno di un elemento div con un id "note" che verrà azionato attraverso un evento `click()` della libreria jQuery.

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

Per un elenco approfondito di tutti i parametri si consulti la documentazione ufficiale: <http://api.jquery.com/animate/>.

- `fadeIn()` / `fadeOut()`

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

- `target`: qualsiasi selettore jQuery o selettore comune
- `speed`: è opzionale ed indica la velocità con delle parole chiave (slow, normal, fast) oppure specificata con il numero di millisecondi desiderati
- `extra information`: è opzionale e può contenere una callback o altre informazioni aggiuntive

- `toggleClass()` questa funzione aggiunge o rimuove una classe CCS al proprio target.

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

- target: qualsiasi selettore jQuery o selettore comune
  - class: è il nome di una classe CSS che deve essere definita e disponibile prima di essere caricata
- fadeIn() / fadeOut() questi effetti permettono ad uno o più elementi di scomparire o riapparire nel tempo.

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

- target: qualsiasi selettore jQuery o selettore comune
  - speed: è opzionale ed indica la velocità con delle parole chiave (slow, normal, fast) oppure specificata con il numero di millisecondi desiderati
  - extra information: è opzionale e può contenere una callback o altre informazioni aggiuntive
- slideUp() / slideDown() / slideToggle() questi effetti permettono lo slide (scorrimento) di uno o più elementi:

```
$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: qualsiasi selettore jQuery o selettore comune
- speed: è opzionale ed indica la velocità con delle parole chiave (slow, normal, fast) oppure specificata con il numero di millisecondi desiderati
- extra information: è opzionale e può contenere una callback o altre informazioni aggiuntive

## PLUGIN

Diversi plugin jQuery possono essere utilizzati con questa libreria.

`corner()` è utilizzato per aggiungere angoli distinti agli elementi della pagina. Per ulteriori dettagli si veda la documentazione ufficiale: <http://www.malsup.com/jquery/corner/>

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

- `target`: qualsiasi selettore jQuery o selettore comune
- `corner_style`: è un parametro opzionale e può essere impostato per ogni stile valido come `round`, `sharp`, `bevel`, `bite`, `dog`, etc. Possono essere impostati angoli individuali con uno stile che prevede uno spazio e con `t1` (in alto a sinistra), `tr` (in alto a destra), `b1` (in basso a sinistra), o `br` (in basso a destra).

```
$this->jquery->corner("#note", "cool t1 br");
```

`tablesorter()` [...]

`modal()` [...]

`calendar()` [...]



## 8.17 CLASSE LOADER

Questa classe, come il nome suggerisce, è utilizzata per caricare elementi come librerie (classi), Viste, Helper e Modelli nei propri file. Essa viene inizializzata automaticamente dal sistema, quindi non è necessario farlo manualmente.

La seguente funzione ampiamente configurabile è la:

- `$this->load->library('class_name', $config, 'object name')`  
questa funzione è utilizzata per caricare le classi core. Dove `class_name` è il nome della classe che si vuole caricare. Si utilizzeranno i termini classe e libreria senza distinzioni. Per esempio se si vuole inviare una email con CodeIgniter, il primo passo sarà quello di caricare la classe Email con il proprio controller:

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

Una volta fatto, la libreria sarà pronta per essere utilizzata, con la funzione `$this->email->some_function()`.

I file della libreria possono essere memorizzati nelle sottodirectory all'interno della cartella principale "libraries", o dentro la propria cartella **/application/libraries**. Per caricare un file che si trova in una sottodirectory, è sufficiente includere il percorso, relativo alla cartella "libraries". Ad esempio, se avete il file si trova in:

```
libraries/flavors/chocolate.php
```

Questo sarà caricato utilizzando l'istruzione:

```
$this->load->library('flavors/chocolate');
```

É possibile inserire i file nella sottodirectory desiderata. Inoltre, più librerie possono essere caricate contemporaneamente passando un array di librerie alla funzione di caricamento.

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

## OPZIONI

Il secondo parametro (opzionale) consente di passare i parametri di configurazione; questo avviene solitamente tramite un array:

```
$config = array (
    'mailtype' => 'html',
    'charset'  => 'utf-8',
    'priority' => '1'
);

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

Le opzioni di configurazione di solito possono essere anche impostate tramite un apposito file. Ogni libreria è spiegata in dettaglio nella propria pagina, quindi per maggiori informazioni si rimanda alla lettura della documentazione specifica.

Si prega di prendere nota: quando più librerie vengono fornite in un array come primo parametro, ognuna di esse riceverà le stesse informazioni di parametro.

## ASSEGNARE LA LIBRARY

Se il terzo parametro (opzionale) è vuoto, la libreria sarà assegnata generalmente ad un oggetto con il medesimo nome. Per esempio, se la libreria si chiama Session, essa sarà assegnata ad una variabile di nome **\$this->session**. Se si preferisce configurare il nome della propria classe, si può indicare il suo valore come terzo parametro:

```
$this->load->library('session', '', 'my_session');

// la classe Session è ora disponibile utilizzando:

$this->my_session
```

Se più librerie sono specificate in un array e fornite come primo parametro della funzione, questo parametro sarà scartato.

- **\$this->load->view('file\_name', \$data, true/false)** questa funzione viene utilizzata per caricare la propria Vista. Il primo parametro è obbligatorio poiché rappresenta il nome della Vista da caricare. Nota: l'estensione del file php non deve essere specificata se non si utilizza un file differente da **.php**.

Il secondo parametro opzionale può essere un array associativo o un oggetto, che può essere eseguito attraverso la funzione **extract** di PHP (<http://www.php.net/extract>) per convertire le variabili che possono essere utilizzate nelle Viste.

Il terzo parametro opzionale consente di modificare il comportamento della funzione in modo che i dati siano restituiti come una stringa piuttosto che inviati al browser. Questo può essere utile se si desidera elaborare i dati in qualche modo: se si imposta il parametro su TRUE saranno restituiti i dati. Il comportamento predefinito è comunque FALSE, e questo fa sì che i dati

vengano inviati al browser. È necessario assegnare la funzione ad una variabile per recuperare i dati restituiti:

```
$string = $this->load->view('myfile', '', true);
```

- `$this->load->model('nome_Modello')`

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

Se il proprio Modello si trova in una sottodirectory, si include anche il percorso relativo rispetto alla directory dei propri “modelli”. Per esempio se il proprio Modello si trova nel percorso `/application/models/blog/queries.php` si utilizzerà il seguente codice:

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

Se si vuole che il proprio Modello sia assegnato ad un oggetto dal nome differente, si può specificare quest’ultimo tramite il secondo parametro della funzione adibita al caricamento:

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

- `$this->load->database('options', true/false)` questa funzione permette di caricare la classe Database. I due parametri sono opzionali.
- `$this->load->vars($array)` questa funzione prende in ingresso un array associativo per generare variabili utilizzando la funzione `extract` del PHP: il risultato è il medesimo che si otterrebbe utilizzando il secondo parametro della funzione `$this->load->view()`. La ragione per cui si potrebbe desiderare di utilizzare questa funzione è che con quest’ultima si possono impostare alcune variabili globali nel costruttore del proprio controller, i quali saranno resi disponibili in qualsiasi Vista caricata. È possibile avere più chiamate a questa funzione. I dati vengono memorizzati nella cache e fusi in un unico array per essere convertiti in variabili.
- `$this->load->get_var($key)` questa funzione controlla l’array associativo delle variabili disponibili nelle proprie Viste. Risulta utile, se per qualsiasi ragione una variabile è impostata in una libreria oppure se un altro metodo del controller utilizza `$this->load->vars()`.

- `$this->load->helper('nome_file')` essa carica i file helper dove `nome_file` è per l'appunto il nome del file, senza l'estensione `_helper.php`.
- `$this->load->file('filepath/filename', true/false)` si tratta di una funzione generica per il caricamento di un file. Si fornisce alla funzione il percorso del file e il nome come primo parametro, e questa aprirà e leggerà il contenuto del file. Per impostazione predefinita i dati sono inviati direttamente al browser come per esempio una Vista, ma è possibile impostare il secondo parametro su `TRUE` se si vuole che i dati siano restituiti come una stringa.
- `$this->load->language('file_name')` questa funzione restituisce un alias alla funzione `language` (caricamento della lingua):

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

- `$this->load->config('file_name')` questa funzione fornisce un alias alla funzione `config` (caricamento dei file):

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

## PACKAGE

Un pacchetto applicativo (definito spesso come `application package`) consente la facile distribuzione di gruppi di risorse in una singola directory, insieme con le proprie librerie, modelli, helper, config e file di linguaggio. Si raccomanda di inserire il proprio package nella cartella `application/third_party`. Qui di seguito si fornisce come esempio una mappa di una directory di package:

```
/application/third_party/foo_bar  
  
config/  
helpers/  
language/  
libraries/  
models/
```

Qualunque sia lo scopo dell'`application package` "Foo bar", questo ha i suoi file di configurazione, helper, file di lingua, librerie e modelli. Per utilizzare queste risorse nel proprio controller, è necessario prima informare la classe `Loader` che queste verranno caricate da un package, aggiungendo il percorso di quest'ultimo.

- `$this->load->add_package_path()` Aggiungendo il percorso del package si indica alla classe `Loader` di anteporre questo percorso alle successive richieste

delle risorse. Come esempio prendiamo in considerazione il package Foo Bar di cui sopra che ha una libreria denominata `Foo_bar.php`. Il nostro controller sarà pertanto:

```
$this->load->add_package_path(APPPATH.'third_party/foo_bar/');  
$this->load->library('foo_bar');
```

- `$this->load->remove_package_path()` quando il proprio controller ha terminato di utilizzare le risorse di un application package e, in particolare si ha bisogno di lavorare con un altro application package, si potrebbe desiderare di rimuovere il suo percorso, in modo che il Loader non appaia più nella cartella delle risorse. Per rimuovere l'ultimo percorso aggiunto, basta semplicemente chiamare il metodo senza alcun parametro.

Altrimenti, se si vuole rimuovere il percorso di uno specifico package, si deve specificare lo stesso percorso precedentemente fornito alla funzione `add_package_path()`.

```
$this->load->remove_package_path(APPPATH.'third_party/foo_bar/');
```

## PACKAGE VIEW

Per impostazione predefinita i percorsi delle Viste vengono impostati tramite `add_package_path()`. Si scorrono i percorsi delle Viste e non appena si trova una corrispondenza, la Vista viene caricata. È possibile che si verifichino delle collisioni con i nomi delle Viste, aspetto che preclude il loro caricamento. Per evitare questo inconveniente si imposta un secondo parametro (opzionale) su `FALSE`, quando si richiama la funzione `add_package_path()`.

```
$this->load->add_package_path(APPPATH.'my_app', TRUE);  
$this->load->view('my_app_index'); // Load  
$this->load->view('welcome_message'); // Non carica il messaggio predefinito  
    welcome_message b/c il secondo parametro di add_package_path è TRUE  
  
// Reset  
$this->load->remove_package_path(APPPATH.'my_app');  
  
// Nuovamente senza il secondo parametro  
$this->load->add_package_path(APPPATH.'my_app', TRUE);  
$this->load->view('my_app_index'); // Load  
$this->load->view('welcome_message'); // Load
```

## 8.18 CLASSE LANGUAGE

La classe `Language` fornisce delle funzioni per definire file e linee di testo in determinate lingue e rendere così internazionale il proprio progetto. Nella cartella di sistema si trova una directory di nome `language` che contiene insiemi di file per la lingua. È possibile creare le proprie definizioni al fine di visualizzare avvisi di errore ed altri messaggi nell'idioma desiderato.

I file linguaggio sono generalmente memorizzati nella directory `/system/language`. In alternativa è possibile creare una cartella denominata **language** all'interno di **application** e memorizzare al suo interno i propri file. CodeIgniter cercherà prima nella directory `/system/language`: se la directory non esiste oppure se la lingua specificata non si trova in quel percorso, allora CI effettuerà una nuova ricerca nella cartella `/system/language`.

Nota: ogni lingua deve essere memorizzata nella propria cartella. Per esempio i file per la lingua inglese sono memorizzati in `/system/language/english`.

Tutti i file linguaggio contengono l'estensione `_lang.php`. Per esempio, un file che contenga i messaggi di errore si chiamerà `error_lang.php`. All'interno del file è possibile assegnare ogni linea di testo ad un array chiamato **\$lang** come nel prototipo seguente:

```
$lang['language_key'] = "Il messaggio che si sta visualizzando";
```

Nota: è buona pratica utilizzare un prefisso comune per tutti i messaggi di una certa categoria, in modo da prevenire collisioni tra nomi simili. Per esempio, se si creano file riguardando i messaggi di errore, una buona abitudine sarà quella di utilizzare il prefisso `_error` come nell'esempio seguente:

```
$lang['error_email_missing'] = "Si deve indicare un indirizzo email";  
$lang['error_url_missing'] = "Si deve specificare un URL";  
$lang['error_username_missing'] = "Si deve specificare uno username";
```

## CARICARE UN LINGUAGGIO

Il caricamento di un file linguaggio avviene attraverso l'istruzione:

```
$this->lang->load('filename', 'language');
```

Dove "filename" è il nome del file che si vuole caricare senza l'estensione del file; la parola **language** invece definisce la lista di linguaggi che contiene all'interno. Se manca il secondo parametro, sarà utilizzata la lingua predefinita memorizzata nel file `/application/config/config.php`

## RECUPERARE UN TESTO

Una volta caricato il file linguaggio, per accedere a qualsiasi linea di testo, si utilizza la funzione:

```
$this->lang->line('language_key');
```

Dove `language_key` è la chiave dell'array che corrisponde alla linea che si desidera visualizzare. La funzione in esame, semplicemente restituisce la linea e non visualizza il suo contenuto.

Nota: l'utilizzo della funzione per definire i label (etichette) di un form è stato deprecato nell'ultima versione di CodeIgniter. Se si ha questa esigenza, è consigliato utilizzare la funzione **lang()** definita nell'helper Language.

## CARICAMENTO AUTOMATICO

Per ultimo, si vuole introdurre un sistema per caricare globalmente e automaticamente un particolare linguaggio nella propria applicazione durante la sua inizializzazione. Per ottenere questo risultato si aggiunga la lingua desiderata nell'array autoload che si trova nel file `/application/config/autoload.php`.

## 8.19 CLASSE MIGRATION

Le migrazioni rappresentano un modo conveniente per modificare il database in modo strutturato e organizzato. Si può sempre modificare il codice SQL a mano ovviamente, ma questo comporta vari compiti poco gratificanti, come il dover spiegare le modifiche introdotte a tutti gli sviluppatori che utilizzeranno il nostro progetto. Si dovrà inoltre tenere traccia di quali modifiche sono necessarie per far funzionare le macchine di produzione, la prossima volta che si distribuirà il proprio software. La tabella del database di migrazione tiene traccia di ciò che è stato già eseguito, quindi tutto quello che si dovrà fare è aggiornare i file dell'applicazione e richiamare la funzione `$this->migrate->current()` per capire quale migrazione dovrebbe essere eseguita. La versione attuale si trova nel file `/config/migration.php`.

### CREARE UNA MIGRATION

Per comprendere meglio il funzionamento della classe, effettueremo una prima migrazione per un nuovo sito costituito da un blog. Tutte le migrazioni si trovano nella cartella `/application/migrations/` e sono caratterizzate da un nome come `001_add_blog.php`.



```
defined('BASEPATH') OR exit('Non è consentito alcun accesso diretto');

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');
    }
}
```

Nel file `/application/config/migration.php` sarà necessario impostare il parametro `$config['migration_version']` sul valore 1.

```
$config['migration_version'] = 1;.
```

## IN PRATICA

Quello sin qui descritto è un semplice codice posto nel file **migrate.php** che aggiorna lo schema:

```

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

if ( ! $this->migration->current() )
{
    show_error($this->migration->error_string());
}

```

## REFERENCE

- `$this->migration->current()` la migrazione attuale è quella impostata in `$config['migration_version']` nel file **application/config/migration.php**
- `$this->migration->latest()` il funzionamento è più o meno simile a quello della funzione **current()**, ma invece di cercare `$config['migration_version']` la classe Migration utilizzerà quella più recente presente nel filesystem
- `$this->migration->version()` la versione utilizzata per il rollback delle modifiche. Funziona proprio come la funzione **current()**, ma ignora il parametro `$config['migration_version']`

## PREFERENZE

Preferenze	Default	Opzioni	Descrizione
<code>migration_enabled</code>	FALSE	TRUE/FALSE	Abilita o disabilita le migrazioni.
<code>migration_version</code>	0	Nessuno	La versione attuale del database da utilizzare.
<code>migration_path</code>	APPPATH.'migrations/'	Nessuno	Il percorso della cartella migrations.

## 8.20 CLASSE OUTPUT

La classe Output ha un unico scopo: inviare le pagine web finali al browser web che le richiede. É anche responsabile per la memorizzazione nella cache delle pagine web, se si utilizza questa funzionalità. La classe viene inizializzata automaticamente dal sistema, quindi non deve essere caricata manualmente. In circostanze normali, non ci si accorge neppure delle azioni svolte dalla classe Output dato che funziona in modo trasparente senza alcun intervento da parte dello sviluppatore. Ad esempio, quando si utilizza la classe Loader per caricare una Vista, questa viene passata automaticamente alla classe Output, che sarà richiamata da CodeIgniter alla fine dell'esecuzione del sistema. É possibile, tuttavia, intervenire manualmente utilizzando una delle seguenti funzioni:

- `$this->output->set_output()` permette di definire manualmente le stringhe di output. Per esempio:

```
$this->output->set_output($data);
```

Se si configura manualmente l'output, questo deve essere l'ultimo elemento a venire richiamato. Per esempio se si costruisce una pagina utilizzando uno dei metodi del controller, non si può richiamare il suo output se non come ultima istruzione.

- `$this->output->set_content_type()` permette di definire il tipo di mime della pagina in maniera tale da poter servire facilmente i dati JSON, JPEG, di XML, ecc.

```
$this->output
->set_content_type('application/json')
->set_output(json_encode(array('foo' => 'bar')));

$this->output
->set_content_type('jpeg') // Si puo' anche utilizzare ".jpeg" which
will have the full stop removed before looking in config/mimes.php
->set_output(file_get_contents('files/qualcosa.jpg'));
```

É importante che ogni stringa passata al metodo che non è del tipo mime, esista all'interno del file **/config/mimes.php** oppure la funzione non sarà eseguita correttamente.

- `$this->output->get_output()` permette di recuperare manualmente ogni dato in uscita inviato per la memorizzazione nella classe Output. Per esempio:

```
$string = $this->output->get_output();
```

I dati non saranno recuperabili tramite questa funzione se questi sono stati inviati precedentemente alla classe Output tramite una funzione di CodeIgniter come `$this->load->view()`.

- `$this->output->append_output()` è utilizzata per aggiungere dati nella stringa di output. Per esempio:

```
$this->output->append_output($data);
```

- `$this->output->set_header()` consente di impostare manualmente gli header server che la classe Output invierà per noi quando sarà completata la pagina da visualizzare. Per esempio:

```
$this->output->set_header("HTTP/1.0 200 OK");
$this->output->set_header("HTTP/1.1 200 OK");
$this->output->set_header('Last-Modified: '.gmdate('D, d M Y H:i:s',
    $last_update).' GMT');
$this->output->set_header("Cache-Control: no-store, no-cache, must-
    revalidate");
$this->output->set_header("Cache-Control: post-check=0, pre-check=0");
$this->output->set_header("Pragma: no-cache");
```

- `$this->output->set_status_header(code, 'text')` permette di definire manualmente gli header server status (le intestazioni con lo stato del server). Per esempio:

```
$this->output->set_status_header('401');
// Imposta l'header come: non autorizzato
```

Tutti i codici Status sono disponibili sul sito: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>.

- `$this->output->enable_profiler()` consente di abilitare/disabilitare il Profiler per visualizzare il benchmark (analisi delle prestazioni) e altre informazioni in fondo alle proprie pagine per scopi di debug e di ottimizzazione. Per abilitare il Profiler si inserisca la seguente istruzione in ogni funzione del Controller desiderato:

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

Quando il report sarà abilitato, questo verrà generato e inserito in fondo alle proprie pagine. Per disabilitare il Profiler si utilizzerà l'istruzione:

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

- `$this->output->set_profiler_sections()` questa funzione abilita/disabilita specifiche sezioni del Profiler quando questo è attivo. Per maggiori informazioni si rimanda alla documentazione del Profiler a pagina [96](#).
- `$this->output->cache()` la libreria Output di CodeIgniter controlla anche l'operazione di memorizzazione (caching). Per maggiori informazioni si veda la sezione Web Page Caching a pagina [94](#).

## PARSING DELLE VARIABILI

CodeIgniter analizza, per impostazione predefinita, tutte le pseudo variabili di tipo `elapsed_time` e `memory_usage` nel proprio output. Per disabilitare questa analisi, si imposti nel proprio controller la proprietà della classe `$parse_exec_vars` su `FALSE`:

```
$this->output->parse_exec_vars = FALSE;
```

## 8.21 CLASSE PAGINATION

La classe `Pagination` è molto intuitiva, oltre ad essere completamente personalizzabile sia dinamicamente che tramite opportuni parametri. Se non si ha dimestichezza con il termine “pagination” si può pensare ad esso come alla lista dei link che permettono di navigare di pagina in pagina, come nell’esempio seguente:

« First < 1 2 3 4 > Last »

Ecco un semplice codice che introduce la classe pagination all’interno del proprio controller:

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

$config['base_url'] = 'http://example.com/index.php/test/page/';
$config['total_rows'] = 200;
$config['per_page'] = 20;

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

echo $this->pagination->create_links();
```

L’array `$config` contiene le variabili di configurazione: questi viene passato alla funzione `$this->pagination->initialize()`. Esistono almeno una ventina di parametri che si possono impostare, ma solo tre saranno fondamentali:

- `base_url` si tratta dell’URL completo del controller classe/metodo che contiene il pagination. Nell’esempio precedente, esso punta al controller chiamato **Test** e quindi al metodo **page**. Si tenga a mente che se si modifica la struttura è possibile apportare le opportune variazioni di instradamento agendo sull’URI (si veda la sezione [4.1 a pagina 37](#))
- `total_rows` questo numero rappresenta il numero totale di righe che compariranno nella lista dei risultati del pagination. Solitamente questo combacia con il numero di record restituiti dall’interrogazione del database
- `per_page` è il numero di elementi che si intende visualizzare in ogni pagina. Nell’esempio precedente, questo parametro è stato impostato su 20 elementi per pagina

La funzione `create_link` restituisce una stringa vuota quando non vi è alcun pagination da mostrare.

Se si preferisce impostare le preferenze non dinamicamente, ma tramite un file di configurazione, si deve definire l’array `$config` all’interno del nuovo file **pagination.php**; questo verrà salvato in `sys/config/pagination.php`. In questo modo i parametri definiti al suo interno verranno automaticamente utilizzati, senza la necessità di istanziarli tramite la funzione `$this->pagination->initialize()`.

## PARAMETRI DI CONFIGURAZIONE

Qui di seguito vengono mostrati tutti i parametri per configurare il pagination secondo le proprie esigenze:

- `$config['uri_segment'] = 3`; la funzione pagination determina automaticamente quale segmento dell'URI contiene il numero di pagina. Se si ha bisogno di qualcosa di diverso è possibile specificarlo
- `$config['num_links'] = 2`; il numero di link che si vorrebbe visualizzare prima e dopo il numero di pagina selezionata. Ad esempio, il numero 2 conterrà due cifre (link) su entrambi i lati, come nell'esempio pubblicato in cima al capitolo
- `$config['use_page_numbers'] = TRUE`; per impostazione predefinita, il segmento URI utilizzerà l'indice iniziale per gli elementi che si stanno impaginando. Se si preferisce visualizzare il numero di pagina attuale, si imposti il valore su TRUE
- `$config['page_query_string'] = TRUE`; per impostazione predefinita, la libreria Pagination assume che si stia utilizzando la segmentazione URI, e costruisce i link come nell'esempio:

```
http://example.com/index.php/test/page/20
```

Se si utilizzano gli slash per definire l'URI (query string) e quindi si è impostato su TRUE il parametro `$config['enable_query_strings']`, allora i link verranno automaticamente impostati utilizzando le stringhe di query (questa opzione può anche essere impostata esplicitamente). Utilizzando il parametro `config['page_query_string']` con valore TRUE, il link pagination diventerà:

```
http://example.com/index.php?c=test&m=page&per_page=20
```

La stringa `per_page` nell'istruzione precedente, è la stringa query che viene passata per default. Il suo valore si può impostare personalizzando il parametro `$config['query_string_segment'] = 'tua_stringa'`

## MARKUP

Se si vuole inserire la pagination tra elementi di markup, si possono utilizzare queste due funzioni:

- `$config['full_tag_open'] = '<p>'`; definisce il tag di apertura sul lato sinistro dei risultati

- `$config['full_tag_close'] = '</p>'`; definisce il tag di chiusura sul lato destro dei risultati

## IL LINK FIRST

- `$config['first_link'] = 'First'`; specifica il testo che comparirà nel link “first” (primo) sulla sinistra. Si può disabilitare impostando il valore su FALSE
- `$config['first_tag_open'] = '<div>'`; Il tag di apertura del link first (primo)
- `$config['first_tag_close'] = '</div>'`; Il tag di chiusura del link first (primo)

## IL LINK LAST

- `$config['last_link'] = 'Last'`; specifica il testo che comparirà sul link “last” (ultimo) sulla destra. È possibile disabilitarlo impostando il valore su FALSE
- `$config['last_tag_open'] = '<div>'`; il tag di apertura del link last (ultimo)
- `$config['last_tag_close'] = '</div>'`; il tag di chiusura del link last (ultimo)

## IL LINK NEXT

- `$config['next_link'] = '&gt;'`; specifica il testo che comparirà sul link “next” (prossimo). È possibile disabilitarlo impostando il valore su FALSE
- `$config['next_tag_open'] = '<div>'`; il tag di apertura del link next (prossimo)
- `$config['next_tag_close'] = '</div>'`; il tag di chiusura del link next (prossimo)

## IL LINK PREVIOUS

- `$config['prev_link'] = '&lt;'`; specifica il testo che si comparirà sul link “previous” (precedente). È possibile disabilitarlo, impostando il valore su FALSE
- `$config['prev_tag_open'] = '<div>'`; il tag di apertura del link previous (precedente)



- `$config['prev_tag_close'] = '</div>'`; il tag di chiusura del link previous (precedente)

#### IL LINK CURRENT PAGE

- `$config['cur_tag_open'] = '<b>'`; il tag di apertura del link current (attuale)
- `$config['cur_tag_close'] = '</b>'`; il tag di chiusura del link current (attuale)

#### IL LINK DIGIT

- `$config['num_tag_open'] = '<div>'`; il tag di apertura del link digit (cifra)
- `$config['num_tag_close'] = '</div>'`; il tag di chiusura del link digit (cifra)

#### NASCONDERE LE PAGINE

- `$config['display_pages'] = FALSE`; questo parametro impostato su FALSE non visualizza la pagina indicata nella lista dei link. In questo modo, per esempio, si possono visualizzare solo i link “next” e “previous”.

#### ANCHOR

Nota: è possibile aggiungere una propria classe di attributi ad ogni link di navigazione attraverso la classe Pagination. Per fare questo si deve impostare la `anchor_class` con lo stesso nome della classe desiderata.

## 8.22 CLASSE SECURITY

La classe Security contiene i metodi utili a definire un'applicazione sicura, grazie all'elaborazione dei dati in ingresso.

### FILTRO XSS

CodeIgniter è dotato di un filtro di prevenzione Cross Site Scripting Hack, che può essere eseguito automaticamente per filtrare tutti i dati POST e COOKIE oppure manualmente, scegliendo uno o più elementi specifici. Per impostazione predefinita questo filtro non viene eseguito a livello globale poiché potrebbe non essere sempre necessario e poi perché aumenta, anche se di poco, il carico di elaborazione sul server. Il filtro XSS scova tecniche comunemente utilizzate per attivare Javascript o altri tipi di codice che dirottano i cookie o compiono altre operazioni potenzialmente dannose. Se si incontra "qualcosa di non conosciuto", questo viene reso sicuro convertendo i dati in entità carattere.

Nota: questa funzione deve essere utilizzata solo per gestire i dati che vengono inviati al server. Non deve quindi essere utilizzata per esaminare i dati coinvolti nei processi interni di esecuzione in quanto verrebbero inutilmente impegnate parte delle risorse di elaborazione del server.

Per filtrare i dati attraverso il filtro XSS viene utilizzata questa funzione:

- `$this->security->xss_clean()` il cui argomento è una variabile che contiene i dati da esaminare. Per esempio:

```
$data = $this->security->xss_clean($data);
```

Se si vuole che il filtro XSS sia eseguito automaticamente ogni volta che si incontra un dato POST oppure COOKIE, è necessario intervenire sul file di configurazione generale `/application/config/config.php`:

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

Nota: se si utilizzano i metodi della classe Form Validation (validazione dei form), il filtro XSS verrà eseguito automaticamente.

È inoltre possibile esaminare le immagini per prevenire sempre potenziali attacchi XSS. Per abilitare questa funzionalità si utilizza un secondo parametro opzionale, `is_image`: utile per l'upload di file sul server. Quando questo secondo parametro è impostato su TRUE, la funzione invece di restituire una stringa alterata, restituisce un valore booleano: TRUE se l'immagine è sicura, e FALSE se contiene informazioni potenzialmente dannose per il browser.

```
if ($this->security->xss_clean($file, TRUE) === FALSE)
{
    // Il file ha fallito il test XSS
}
```

- `$this->security->sanitize_filename()` quando si accettano i nomi di file inviati dall'utente, si è a rischio di attacchi di tipo "Directory traversal" (attraversamento di directory). Per prevenire anche questi rischi, CodeIgniter mette a disposizione il metodo `sanitize_filename()`. Qui di seguito un esempio:

```
$filename = $this->security->sanitize_filename($this->input->post('filename'));
```

Se si vuole consentire all'utente l'inserimento del percorso relativo di un file, come per esempio `file/in/some/approved/folder.txt`, si può impostare un secondo parametro opzionale `$relative_path` sul valore `TRUE`:

```
$filename = $this->security->sanitize_filename($this->input->post('filename'), TRUE);
```

## CROSS-SITE REQUEST FORGERY (CSRF)

La protezione CSRF ([CSRF](#)) si abilita agendo sul file di configurazione generale, e più precisamente impostando il parametro sul valore booleano `TRUE`:

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

Nota: se si utilizza la funzione `form_open()` dell'Helper Form, verrà inserito un campo nascosto [CSRF](#) nei propri form.

## 8.23 CLASSE SESSION

La classe Session (sessione) permette di tenere traccia dell'utente collegato e di seguire la sua attività all'interno del sito sino a quando sarà presente. La classe Session memorizza le informazioni sulle sessioni riguardanti ogni utente all'interno di appositi file denominati cookie; queste informazioni possono essere opzionalmente crittografate. È anche possibile memorizzare i dati della sessione all'interno di una tabella del database per incrementare la sicurezza: questo permette di confrontare l'ID della sessione presente nel cookie dell'utente con quelli precedentemente memorizzati nel database. In questo modo si effettuano degli ulteriori controlli incrociati sull'identità dell'utente. Per impostazione predefinita solo i cookie vengono salvati, ma se si sceglie di memorizzare anche le informazioni sul database, si dovrà creare una tabella apposita adibita alla memorizzazione delle sessioni, come verrà descritto più avanti.

Nota: la classe Session non utilizza le sessioni native del [PHP](#) poiché genera dei propri dati di sessione, che offrono maggiore flessibilità agli sviluppatori.

Nota: anche se non si utilizzano sessioni crittografate, è necessario impostare una chiave di cifratura nel file di configurazione che verrà utilizzata per aiutare a prevenire la manipolazione dei dati di sessione.

### INIZIALIZZARE UNA SESSIONE

Una sessione è generalmente eseguita globalmente con il caricamento di qualsiasi pagina, quindi la classe Session deve essere inizializzata nei metodi costruttore del proprio controller oppure sarà caricata automaticamente dal sistema. Per inizializzare la classe, manualmente nel proprio costruttore del controller, si usa la seguente funzione: `$this->load->library`

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

Una volta caricata, l'oggetto della libreria Session sarà disponibile con l'istruzione `$this->session`.

### COME FUNZIONA UNA SESSIONE

Il meccanismo alla base delle sessioni è abbastanza semplice: quando una pagina viene caricata, la classe controlla prima di tutto se, per l'utente, esiste già una sessione valida all'interno dei suoi cookie. Se i dati della sessione non esistono (oppure se sono scaduti), verrà creata una nuova sessione memorizzata all'interno del cookie. Invece, se una sessione utente è presente, le sue informazioni verranno aggiornate così come il cookie che le contiene. Con ogni aggiornamento il `session_id` sarà rigenerato. È importante tenere a mente che la classe Session, una volta inizializzata, funzionerà automaticamente senza che sarà necessario apportare alcuna modifica.

Inoltre è possibile lavorare con i dati della sessione o anche aggiungere i propri dati alla sessione di un utente, ma in generale il processo di lettura, scrittura e aggiornamento di una sessione è automatico.

## I DATI DI SESSIONE

Una sessione, per quanto concerne CodeIgniter, è semplicemente un array contenente le seguenti informazioni:

- ID Session: è l'identificativo univoco dell'utente che si collega al sito. A partire dalla versione 2.2.0 di CodeIgniter viene utilizzata una autenticazione basata su keyed-hash message authentication code ([HMAC](#))
- IP Address: l'indirizzo IP dell'utente
- User Agent: i primi 120 caratteri contenuti nella stringa delle informazioni del browser

I dati precedentemente menzionati sono memorizzati nei cookie in array serializzati come nel prototipo seguente:

```
[array]
(
    'session_id'    => random hash,
    'ip_address'    => 'string - user IP address',
    'user_agent'    => 'string - user agent data',
    'last_activity' => timestamp
)
```

Se si ha abilitata l'opzione di cifratura, l'array serializzato verrà crittografato prima di essere memorizzato nel cookie, rendendo i dati altamente sicuri. Maggiori informazioni riguardo la crittografia sono disponibili nella sezione dedicata alla classe Encryption a pagina [233](#).

Nota: i cookie di sessione sono aggiornati solo ogni cinque minuti per impostazione predefinita, al fine di ridurre il carico del server. Se si ricarica più volte una pagina si noterà che il tempo relativo all'ultima attività (last activity) sarà aggiornato solo se sono trascorsi 5 o più minuti dall'ultima volta in cui si è scritto nel cookie. Questo tempo è comunque personalizzabile modificando il parametro `$config['sess_time_to_update']` nel file `file/system/config/config.php`.

## RECUPERARE I DATI DI SESSIONE

Ogni dato dall'array di sessione si può recuperare utilizzando la seguente funzione:

```
$this->session->userdata('item');
```

Dove **item** è l'indice dell'array che corrisponde all'elemento che si vuole recuperare. Per esempio per ottenere l'ID di sessione si fa affidamento sull'istruzione:

```
$session_id = $this->session->userdata('session_id');
```

La funzione restituisce FALSE se l'elemento richiesto non esiste.

## AGGIUNGERE I PROPRI DATI SESSIONE

Un'aspetto molto utile degli array di sessione è la possibilità di aggiungere i propri dati e memorizzarli nel cookie dell'utente. Ma perché si dovrebbe fare una cosa simile? Si immagini una situazione di questo tipo:

Un utente accede al sito tramite login/password. Una volta autenticato si potrebbe voler memorizzare lo username e l'email nel cookie di sessione, in modo da rendere disponibili queste informazioni globalmente, senza dover eseguire successivamente delle query sul database per recuperarle.

Per aggiungere le informazioni all'array di sessione si passa un array contenente i nuovi dati, tramite la funzione:

```
$this->session->set_userdata($array);
```

Dove **\$array** è per l'appunto un array associativo contenente le nuove informazioni. Ecco un estratto di codice:

### Definizione 19: HMAC

---

#### HMAC

È un codice per l'autenticazione di messaggi basata su una funzione di hash: fornisce un ragionevole grado di sicurezza atto a garantire sia l'integrità che l'autenticità di un messaggio. Una caratteristica peculiare di HMAC è il non essere legato a nessuna funzione di hash predefinita, questo per rendere possibile una sostituzione della funzione nel caso non fosse abbastanza sicura. Informazioni più dettagliate sono disponibili sul documento RFC 2104: <http://tools.ietf.org/html/rfc2104>

---

```
$newdata = array(  
    'username' => 'johndoe',  
    'email'    => 'johndoe@some-site.com',  
    'logged_in' => TRUE  
);  
  
$this->session->set_userdata($newdata);
```

Per aggiungere un valore dell'utente alla volta, la funzione `set_userdata()` supporta anche questa sintassi:

```
$this->session->set_userdata('qualche_nome', 'qualche_valore');
```

Nota: i cookie possono contenere solo 4KB di dati, quindi si deve prestare attenzione a non superare tale capacità. Il processo di crittografia, in particolare, produce una stringa di dati più lunga di quella originale in modo da tenere una traccia accurata dei dati memorizzati.

## RECUPERARE TUTTI I DATI SESSIONE

Un array con tutti i dati dell'utente può essere ottenuto con la seguente istruzione:

```
$this->session->all_userdata()
```

Mentre per ottenere un array associativo si utilizzerà:

```
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  
)
```

## ELIMINARE I DATI SESSIONE

Così come è possibile aggiungere alcune informazioni in una sessione con `set_userdata()`, allo stesso modo è possibile rimuoverle con `unset_userdata()`

a cui si passerà semplicemente la chiave di sessione. Per esempio, se si vuole rimuovere `qualche_nome` dal flusso di informazioni della sessione:

```
$this->session->unset_userdata('qualche_nome');
```

Se si vogliono eliminare più dati in una sola volta, si utilizzerà sempre questa funzione a cui si passerà, questa volta, un array associativo con gli specifici elementi non più necessari:

```
$array_items = array('username' => '', 'email' => '');  
  
$this->session->unset_userdata($array_items);
```

## FLASHDATA

CodeIgniter supporta i “flashdata”, o i dati di sessione, che sono disponibili solo per la successiva richiesta del server per poi venire automaticamente cancellati. Questi possono essere molto utili poiché sono tipicamente utilizzati per i messaggi informativi o di stato (per esempio: “il record 2 è stato cancellato”).

Nota: le variabili Flash sono precedute da `flash_` quindi si eviti di utilizzare questo prefisso nei propri nomi di sessione.

Per aggiungere i flashdata:

```
$this->session->set_flashdata('item', 'value');
```

Inoltre si può passare un’array a `set_flashdata()`, così come in `set_userdata()`.  
Per leggere una variabile flashdata:

```
$this->session->flashdata('item');
```

Se si vuole preservare una variabile flashdata per un’ulteriore richiesta, è possibile farlo usando la funzione `keep_flashdata()`.

## SALVARE UNA SESSIONE IN UN DATABASE

Sin quando l’array di sessione contenente l’ID Session è memorizzato nel cookie dell’utente, non lo si può validare: l’unica possibilità è memorizzarlo in un database. Per alcune applicazioni che richiedono poca o nessuna sicurezza, la convalida dell’ID Session può non essere necessaria, ma se l’applicazione richiede una maggiore



sicurezza, la convalida si rende obbligatoria. In caso contrario, una vecchia sessione potrebbe essere ripristinata da un utente modificando il proprio cookie.

Quando i dati di sessione sono disponibili in un database, ogni volta che una sessione valida si trova nel cookie dell'utente, viene eseguita una query per effettuare un confronto e vedere se coincidono. Se l'ID di sessione non corrisponde, la sessione viene distrutta. Si presti attenzione al fatto che l'ID di sessione non può mai essere aggiornato, ma solamente generato quando si crea una nuova sessione.

Per memorizzare le sessioni è necessario innanzitutto creare una tabella di database per questo scopo. Ecco il prototipo di base (per MySQL) richiesto dalla classe sessione:

```
CREATE TABLE IF NOT EXISTS `ci_sessions` (  
  session_id varchar(40) DEFAULT '0' NOT NULL,  
  ip_address varchar(45) DEFAULT '0' NOT NULL,  
  user_agent varchar(120) NOT NULL,  
  last_activity int(10) unsigned DEFAULT 0 NOT NULL,  
  user_data text NOT NULL,  
  PRIMARY KEY (session_id),  
  KEY `last_activity_idx` (`last_activity`)  
);
```

Nota: la tabella è chiamata per default `ci_sessions`, ma nulla vieta di impostare un altro nome: è sufficiente aggiornare il file `/application/config/config.php` con il nome desiderato.

```
$config['sess_table_name'] = 'ci_sessions';
```

Una volta creata la tabella nel database, la si abilita nel file `config.php`:

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

Ora la classe Sessione memorizzerà le informazioni nel database. Una particolarità di questa classe è quella di avere incluso un sistema di “garbage-collection” che elimina le sessioni scadute automaticamente, senza un intervento da parte dello sviluppatore.

## DISTRUGGERE UNA SESSIONE

Per eliminare la sessione attualmente attiva l'istruzione è:

```
$this->session->sess_destroy();
```

Questa funzione deve essere invocata per ultima, altrimenti anche le variabili flash non saranno più disponibili. Se invece si vuole eliminare, non tutta la sessione, ma solo alcuni suoi elementi, allora la funzione da utilizzare è `unset_userdata()`.

## PREFERENZE

Qui di seguito si fornisce un elenco delle preferenze relative alla classe `Session`, modificabili nel file `/application/config/config.php`:

Preferenze	Default	Opzioni	Descrizione
<code>sess_cookie_name</code>	<code>ci_session</code>	Nessuno	Il nome con cui il cookie di sessione verrà salvato.
<code>sess_expiration</code>	7200	Nessuno	Il numero di secondi dopo il quale la sessione scadrà. Il valore di default è 2 ore (7200 secondi) ma se non si vuole che la sessione scada, si imposti il valore su 0.
<code>sess_expire_on_close</code>	FALSE	TRUE/FALSE	Determina se la sessione deve scadere automaticamente quando la finestra del browser viene chiusa.
<code>sess_encrypt_cookie</code>	FALSE	TRUE/FALSE	Determina se crittografare i dati della sessione.
<code>sess_use_database</code>	FALSE	TRUE/FALSE	Determina se i dati di sessione devono essere salvati in un database. Se impostato su TRUE, ci si ricordi di creare una apposita tabella.
<code>sess_table_name</code>	<code>ci_sessions</code>	Ogni nome di tabella SQL valida	Il nome della tabella adibita a contenere i dati di sessione.
<code>sess_time_to_update</code>	300	Tempo in secondi	Questa opzione stabilisce quanto spesso la classe di sessione deve rigenerarsi e creare un nuovo id di sessione.
<code>sess_match_ip</code>	FALSE	TRUE/FALSE	Determina se confrontare l'indirizzo IP dell'utente durante la lettura dei dati di sessione. Si noti che alcuni ISP cambiano dinamicamente l'indirizzo IP, quindi se si vuole una sessione che non scada, probabilmente questo valore deve essere FALSE.
<code>sess_match_useragent</code>	TRUE	TRUE/FALSE	Stabilisce se si deve confrontare lo User Agent quando si leggono i dati di sessione.

## 8.24 CLASSE TRACKBACK

La classe Trackback fornisce numerose funzionalità per l'invio e la ricezione dei dati Trackback.

Il Trackback è un meccanismo per la comunicazione e la notifica tra due risorse: la risorsa A manda un Trackback ping (spesso chiamato Pingback) alla risorsa B, la quale risponde con un messaggio di avvenuta notifica o con un eventuale errore. La risorsa A invia poi un ping alla risorsa B, nel caso in cui sia presente, nella risorsa A, un approfondimento o una citazione della risorsa B.

Il Trackback si è molto diffuso nei blog. In questo caso, un blog riceve una serie di ping da altri blog e, solitamente mostra sotto ad ogni post, l'elenco dei ping ricevuti e riferiti a quello specifico post.

Il Trackback ping può contenere informazioni relative al titolo della risorsa notificata, un estratto della stessa e il titolo del sito web che ha inviato il ping. L'unica informazione obbligatoria che un ping deve contenere è il permalink della risorsa notificata.

Oggi, il Trackback è disponibile nei principali programmi di blogging, ma il termine Trackback è nato nel 2002, così battezzato da Six Apart, che introdusse questo sistema nella sua piattaforma di blogging Movable Type, sebbene il Trackback utilizzi funzionalità presenti nelle specifiche HTTP 1.0, e quindi già dal 1992.

Un spiacevole fenomeno diffuso è l'utilizzo dei Trackback a scopo di spam, cioè l'invio indiscriminato di Trackback ping allo scopo di far apparire i propri link nell'elenco dei ping di qualche blog. Esistono, però, filtri per individuare i Trackback di spam, implementati in molte delle piattaforme di blogging che supportano il Trackback, riducendo, almeno parzialmente, il fastidio causato dallo spam.

La classe viene inizializzata nel controller con la funzione **\$this->load->library**:

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

Una volta caricata, l'oggetto della libreria Trackback sarà disponibile utilizzando **\$this->trackback**.

### INVIARE TRACKBACK

Un Trackback può essere inviato da ogni metodo del controller utilizzando un codice simile al seguente esempo:

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

$tb_data = array(
    'ping_url' => 'http://example.com/trackback/456',
    'url'      => 'http://www.my-example.com/blog/entry/123',
    'title'    => 'The Title of My Entry',
    'excerpt'  => 'The entry content.',
    'blog_name' => 'My Blog Name',
    'charset'  => 'utf-8'
);

if ( ! $this->trackback->send($tb_data) )
{
    echo $this->trackback->display_errors();
}
else
{
    echo 'Trackback was sent!';
}
```

Un elenco dei parametri dell'array viene qui elencato prendendo come esempio un blog:

- ping\_url: specifica l'URL del sito a cui si sta inviando il Trackback. É possibile inviare Trackback a più URL separando ognuno di questi ultimi con una virgola
- url: indica l'URL del proprio sito dove la voce del blog può essere inviata
- title: il titolo del proprio blog
- excerpt: è un estratto del weblog. La classe invierà automaticamente solo i primi 500 caratteri della sua voce che saranno anche privati di qualsiasi tag HTML
- blog\_name il nome del blog
- charset: l'encoding utilizzato per la rappresentazione dei caratteri utilizzati nel blog. Se omissso, sarà utilizzata la codifica UTF-8

La funzione di “Invio Trackback” restituisce il valore booleano TRUE/FALSE a seconda del successo o meno dell'operazione. Se essa fallisce si può recuperare il messaggio di errore utilizzando:

```
$this->trackback->display_errors();
```

## RICEVERE TRACKBACK

Prima che si possa ricevere i Trackback è necessario creare un blog, ovviamente. Se non se ne ha uno, e non si ha intenzione di utilizzare un servizio di blog in futuro, probabilmente non ha senso continuare nella lettura delle prossime funzionalità della classe.

Ricevere i Trackback è un po' più complesso di quanto non sia inviarli per il solo motivo che sarà necessaria una tabella di database in cui memorizzarli. Inoltre sarà necessario per convalidare i dati trackback in ingresso. È importante implementare un processo di validazione approfondito per evitare spam e duplicazione dei dati. Si consiglia inoltre di limitare il numero di Trackback in ingresso da un particolare IP in un intervallo di tempo specifico, sempre con il fine di ridurre ulteriormente lo spam. Il processo di ricezione di un Trackback è abbastanza semplice mentre la loro validazione richiede un maggiore sforzo.

## PING URL

Per accettare i Trackback è necessario visualizzare un [URL](#) di Trackback accanto ad ognuno delle voci del blog. Questo [URL](#) sarà l'indirizzo che la gente utilizzerà per inviare Trackback (si farà riferimento a questi ultimi come Ping URL). Il proprio Ping URL deve puntare al metodo del Controller dove si trova il codice del Trackback di ricezione, e l'URL deve contenere il numero ID per ogni particolare voce, in modo che quando viene ricevuto il Trackback si sarà in grado di associarlo ad una particolare voce. Ad esempio, se la classe controller si chiama **Trackback**, e il metodo di ricezione viene chiamato **receive**, il proprio URL Ping sarà simile a questo:

```
http://example.com/index.php/trackback/receive/entry_id
```

Dove `entry_id` rappresenta il numero ID specifico per ogni voce.

## TABELLA TRACKBACK

Prima che si possano ricevere i Trackback è necessario creare una tabella in cui poterli memorizzare. Qui di seguito si mostra il prototipo di una tabella adatta allo scopo:

```
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`)  
);
```

Nelle specifiche del prototipo sono richieste solo quattro porzioni di informazioni da inviare al Trackback (`url`, `title`, `excerpt`, `blog_name`), ma per rendere i dati più utili sono stati aggiunti un paio di ulteriori campi (`date`, `IP address`, etc.).

## ELABORARE UN TRACKBACK

Ora si mostrerà un esempio per ricevere e processare i Trackback; il seguente codice si abbina ad un metodo del controller adibito alla ricezione dei dati Trackback.

```

$this->load->library('trackback');
$this->load->database();

if ($this->uri->segment(3) == FALSE)
{
    $this->trackback->send_error("Unable to determine the entry ID");
}

if ( ! $this->trackback->receive() )
{
    $this->trackback->send_error("The Trackback did not contain valid data");
}

$data = array(
    'tb_id'      => '',
    'entry_id'   => $this->uri->segment(3),
    'url'        => $this->trackback->data('url'),
    'title'      => $this->trackback->data('title'),
    'excerpt'    => $this->trackback->data('excerpt'),
    'blog_name'  => $this->trackback->data('blog_name'),
    'tb_date'    => time(),
    'ip_address' => $this->input->ip_address()
);

$sql = $this->db->insert_string('trackbacks', $data);
$this->db->query($sql);

$this->trackback->send_success();

```

Nota: il numero entry ID è nella terza parte del proprio URL che in questo specifico caso è basato sull'URI seguente:

```
http://example.com/index.php/trackback/receive/entry_id
```

Si osservi come entry\_id sia per l'appunto nella terza parte dell'URI, che si può recuperare usando:

```
$this->uri->segment(3);
```

In questo modo, Trackback riceverà il codice di qui sopra, mentre se la terza parte è assente, sarà visualizzato un errore: senza un ID entry valido non si ha alcun motivo per continuare.

La funzione `$this->trackback->receive()` ha lo scopo di validare i dati in ingresso, assicurandosi che siano presenti le quattro parti di informazione richieste: url, title, excerpt, blog\_name. La funzione restituisce il valore booleano TRUE/FALSE in caso di successo o meno; in caso di insuccesso verrà prodotto un messaggio di errore.



I Trackback in ingresso sono recuperati con la funzione seguente:

```
$this->trackback->data('item')
```

Dove **item** rappresenta uno delle quattro parti delle informazioni: url, title, excerpt, blog\_name. Se i dati Trackback saranno correttamente ricevuti, il messaggio di successo sarà disponibile con la funzione:

```
$this->trackback->send_success();
```

Il codice sopra descritto non contiene alcuna validazione dei dati: si consiglia di definire sempre questa funzionalità nel proprio progetto.

## 8.25 CLASSE TEMPLATE PARSER

La classe in esame permette di abilitare il Template Parser per l'analisi delle pseudo variabili contenute nelle Viste. È possibile effettuare l'analisi delle semplici variabili, così come delle coppie dei tag. Se non si è mai utilizzato un Template Parser, una idea delle pseudo-variabili è fornita dal seguente listato:

```
<html>
<head>
<title>{blog_title}</title>
</head>
<body>

<h3>{blog_heading}</h3>

{blog_entries}
<h5>{title}</h5>
<p>{body}</p>
{/blog_entries}
</body>
</html>
```

Le variabili utilizzate non sono del tipo [PHP](#), ma rappresentazioni in semplice testo, che hanno proprio lo scopo di eliminare ogni riferimento al codice di programmazione nei propri template. Questa classe di CodeIgniter è opzionale. Nonostante sia possibile continuare a utilizzare il codice espresso in [PHP](#), che si dimostra, tra l'altro, leggermente più veloce, l'uso di un motore di template è caldamente consigliato in tutti quei progetti in cui si deve condividere il codice di sviluppo con i designer grafici. La classe inoltre non è una vera e propria soluzione al parsing dei template grafici poiché gli sviluppatori di CodeIgniter hanno realizzato la classe concentrandosi più sulla velocità di esecuzione che sulla sua completezza.

### INIZIALIZZARE LA CLASSE

La classe è inizializzata nel proprio controller con la funzione:

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

Una volta caricata, l'oggetto di libreria sarà disponibile con la funzione **\$this->parser**. La libreria è completata con la definizione delle seguenti funzioni:

- `$this->parser->parse()` questo metodo accetta in ingresso un nome di template e un array di dati e genera una loro versione "parsed". Per esempio:

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

$data = array(
    'blog_title' => 'My Blog Title',
    'blog_heading' => 'My Blog Heading'
);

$this->parser->parse('blog_template', $data);
```

Il primo parametro, chiamato in questo esempio `blog_template.php`), contiene il nome della Vista mentre il secondo è un array associativo di dati che verrà sostituito nel template. Nel codice descritto il template contiene due variabili `blog_title` e `blog_heading`.

Non è necessario utilizzare **echo** (per la visualizzazione) o fare altro con i dati restituiti dal metodo `$this->parser->parse()` perché questi saranno automaticamente inviati alla classe output per essere visualizzati nel browser. È anche possibile non inviare i dati alla classe output semplicemente utilizzando il valore booleano `TRUE` come terzo parametro:

```
$string = $this->parser->parse('blog_template', $data, TRUE);
```

- `$this->parser->parse_string()` questo metodo che funziona come quello precedente, accetta unicamente una stringa come primo parametro al posto di un file Vista.

## BLOCCHI DI VARIABILI

Gli esempi precedenti hanno permesso la sostituzione di semplici variabili. Ma per sostituire interi blocchi di variabili ripetutamente, è necessario utilizzare delle specifiche “coppie di variabili”. Per comprendere meglio il loro funzionamento si consideri il template di esempio visualizzato precedentemente:

```
<html>
<head>
<title>{blog_title}</title>
</head>
<body>

<h3>{blog_heading}</h3>

{blog_entries}
<h5>{title}</h5>
```

```
<p>{body}</p>
{/blog_entries}
</body>
</html>
```

In questo codice sono presenti le coppie di variabili {blog\_entries} e {\blog\_entries}. L'intero blocco di dati tra queste coppie sarà presumibilmente ripetuto più volte per esempio per visualizzare i post di un blog. Il parsing delle coppie di variabili avviene nello stesso identico modo visto per le singole variabili, ad eccezione del fatto che si dovrà aggiungere un array multi dimensionale corrispondente ai dati delle variabili a coppie. Per esempio:

```
$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);
```

Se i propri dati sono prelevati dal risultato di un database, che è già un array multidimensionale, si può semplicemente utilizzare la funzione `result_array()`:

```
$query = $this->db->query("SELECT * FROM blog");

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

$data = array(
    'blog_title' => 'My Blog Title',
    'blog_heading' => 'My Blog Heading',
    'blog_entries' => $query->result_array()
);

$this->parser->parse('blog_template', $data);
```

In questo modo sarà possibile riportare agevolmente tutti i risultati di una base dei dati. Ovviamente si dovrà avere l'accortezza di chiamare le variabili del template con lo stesso nome dei campi della tabella da cui provengono i risultati.

## 8.26 CLASSE TYPOGRAPHY

Questa classe fornisce importanti funzioni per la formattazione del testo. Come di consueto, la sua inizializzazione avviene nel controller con il metodo **`$this->load->library`**:

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

Una volta caricata, l'oggetto libreria sarà disponibile utilizzando: **`$this->typography`**■

- `auto_typography()` formatta il testo per essere semanticamente e tipograficamente corretto nel formato [HTML](#): viene presa una stringa in ingresso che viene poi restituita con la seguente formattazione:
  - delimita i paragrafi con i tag `<p></p>`. I paragrafi sono individuati dal doppio ritorno a capo
  - il singolo ritorno a capo è convertito nel tag `<br />`, eccetto per il testo che si trova all'interno dei tag `<pre>` (preformattato)
  - i tag che delimitano i blocchi di livello come `<div>` non vengono inseriti all'interno dei paragrafi, ma solo il testo da essi delimitato se contengono paragrafi.
  - le doppie virgolette sono convertite con i simboli corretti, tranne se queste appaiono all'interno dei tag
  - gli apostrofi sono convertiti nel corretto carattere apostrofo
  - i doppi trattini tra le parole come `primo -- ultimo` oppure `primo--ultimo` diventano lineette.
  - tre punti consecutivi, precedenti o successivi una parola vengono convertiti in puntini di sospensione . . .
  - i doppi spazi che seguono una frase vengono convertiti in spazi caratteri che imitano la spaziatura doppia

Per esempio:

```
$string = $this->typography->auto_typography($string);
```

### Parametri

Per quanto riguarda i parametri, è possibile utilizzarne uno che determina se il parser deve suddividere su due righe due interruzioni di riga consecutive. Per impostazione predefinita il parser non riduce le interruzioni di linea, ovvero, se nessun parametro (TRUE/FALSE) viene fornito, è come se si eseguisse:

```
$string = $this->typography->auto_typography($string, FALSE);
```

La formattazione attraverso questa classe può essere molto impegnativa per il processore del proprio server, soprattutto se si hanno molti contenuti da formattare. L'utilizzo di questa funzione è consigliata se abbinata alla funzione di caching (si veda la sezione [6.5 a pagina 94](#)).

- `format_characters()` simile alla funzione `auto_typography`, si differenzia per il fatto che è utilizzata solo per la conversione dei caratteri:
  - le doppie virgolette sono convertite con i simboli corretti, tranne se queste appaiono all'interno dei tag
  - gli apostrofi sono convertiti nel corretto carattere apostrofo
  - i doppi trattini tra le parole come `primo -- ultimo` oppure `primo--ultimo` diventano lineette
  - tre punti consecutivi, precedenti o successivi una parola vengono convertiti in puntini di sospensione ...
  - i doppi spazi che seguono una frase vengono convertiti in spazi caratteri che imitano la spaziatura doppia

Per esempio:

```
$string = $this->typography->format_characters($string);
```

- `nl2br_except_pre()` converte il ritorno a capo nel tag `<br />` a meno che esso non appaia con il tag `<pre>`. Questa funzione è simile a quella nativa PHP `nl2br()`, tranne per il fatto che questa ignora i tag `<pre>`:

```
$string = $this->typography->nl2br_except_pre($string);
```

- `protect_braced_quotes` quando viene usata la libreria `Typography` insieme con quella `Template Parser`, spesso si desidera proteggere le virgolette singole e doppie all'interno delle parentesi graffe. Per abilitare questa funzione, si deve impostare la proprietà della classe `protect_braced_quotes` su `TRUE`.

Esempio di utilizzo:

```
$this->load->library('typography');  
$this->typography->protect_braced_quotes = TRUE;
```

## 8.27 CLASSE UNIT TESTING

Unit testing è un approccio allo sviluppo del software in cui vengono scritti test per ogni funzione nell'applicazione.

La classe che risulta abbastanza intuitiva, è costituita da una funzione di valutazione e da due funzioni di risultato. Non deve essere considerata come uno strumento per fare veri e propri test di debug, ma piuttosto come un semplice meccanismo per valutare il codice e determinare se si sta producendo il tipo di dati e i risultati corretti.

La classe viene inizializzata nel proprio controller utilizzando la funzione **\$this->load->library**:

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

Una volta caricata la classe, l'oggetto Unit Test sarà disponibile utilizzando **\$this->unit**.

### TEST

Per eseguire un test è necessario che si fornisca una condizione e il risultato atteso tramite la seguente funzione:

- `$this->unit->run( test, expected result, 'test name', 'notes')`

Dove **test** è il risultato del codice che si vuole testare, **expected result** è il dato che ci si aspetta, **test name** è un nome opzionale che può essere dato al test, e **notes** è un commento, anch'esso opzionale. Per esempio:

```
$test = 1 + 1;

$expected_result = 2;

$test_name = 'Somma di uno piu' uno';

$this->unit->run($test, $expected_result, $test_name);
```

Il risultato atteso che si fornisce alla funzione deve avere una corrispondenza "letterale" o per "tipo". Ecco un esempio di corrispondenza letterale:

```
$this->unit->run('Foo', 'Foo');
```

Ecco un esempio di corrispondenza per tipo:

```
$this->unit->run('Foo', 'is_string');
```

In quest'ultimo esempio, come si sarà intuito, il secondo parametro indica alla funzione di valutare se il tipo di dato è una stringa. Ecco una lista dei tipi di dati utilizzabili nel test:

```
is_object  
is_string  
is_bool  
is_true  
is_false  
is_int  
is_numeric  
is_float  
is_double  
is_array  
is_null
```

## GENERARE UN REPORT

È possibile visualizzare i risultati dopo ogni test, oppure eseguire diversi test e generare un rapporto finale in una pagina formattata come una tabella [HTML](#). Per visualizzare un report si utilizzi la funzione **eco** oppure si utilizzi la funzione **run()**:

```
echo $this->unit->run($test, $expected_result);
```

Per ottenere un rapporto di tutti i test:

```
echo $this->unit->report();
```

Se invece di una formattazione [HTML](#) si preferisce ottenere il report come raw data (dati grezzi), si può utilizzare un array:

```
echo $this->unit->result();
```



## MODALITÀ STRICT

La classe Unit Test come impostazione predefinita effettua un confronto letterale di tipo loose. Si consideri questo esempio:

```
$this->unit->run(1, TRUE);
```

Il test valuta un valore intero (1) ma il risultato atteso è un booleano (TRUE). Il linguaggio PHP, però, grazie alla sua tipizzazione dei dati loose valuterà il codice di cui sopra come TRUE utilizzando un test di uguaglianza normale:

```
if (1 == TRUE) echo 'Valutato come TRUE';
```

Se si preferisce, si può impostare la classe sulla modalità Strict per effettuare un confronto sia con il tipo di dati, che il suo valore:

```
$this->unit->use_strict(TRUE);
```

Ora, con questa modalità, il test di prima darà esito negativo:

```
if (1 === TRUE) echo 'Valutato come FALSE';
```

## UNIT TESTING

È possibile disabilitare alcuni test nei propri script utilizzando:

```
$this->unit->active(FALSE)
```

## VISUALIZZARE UNIT TESTING

Quando vengono visualizzati i risultati dei test, i seguenti elementi saranno messi in evidenza per impostazione predefinita:

- test\_name: il nome del test
- test\_datatype: il tipo di dati del test
- res\_datatype: il tipo di dati atteso

- result: il risultato atteso
- file: il nome del file
- line: il numero di linea
- notes: eventuale descrizione

Si può modificare uno o più elementi sopracitati con la funzione `$this->unit->set_items()`. Per esempio se si desidera visualizzare solo il nome del test e il risultato:

```
$this->unit->set_test_items(array('test_name', 'result'));
```

## CREARE UN TEMPLATE

Se si vuole modificare la formattazione con cui vengono visualizzati i risultati del test si può definire un proprio template. Qui di seguito si mostra un semplice template in cui sono utilizzate delle pseudo variabili:

```
$str = '  
<table border="0" cellpadding="4" cellspacing="1">  
  {rows}  
    <tr>  
      <td>{item}</td>  
      <td>{result}</td>  
    </tr>  
  {/rows}  
</table>';  
  
$this->unit->set_template($str);
```

Nota: il proprio template deve essere dichiarato prima di eseguire il processo Unit Test.

## 8.27.1 Classe URI

Questa classe fornisce un'insieme di funzioni per recuperare preziosi dati dalle proprie stringhe [URI](#). Se si utilizza l'instradamento [URI](#), è possibile anche ottenere informazioni sui segmenti re-instradati. Questa classe, come altre presenti in CodeIgniter è inizializzata automaticamente dal sistema: nessun intervento manuale è necessario.

- `$this->uri->segment(n)` permette di recuperare uno specifico segmento individuato dal valore `n`. La numerazione dei segmenti avviene da sinistra a destra. Per esempio, se l'[URL](#) completo è il seguente:

```
http://example.com/index.php/news/local/metro/crime_is_up
```

I numeri di segmento saranno così indicati:

1. news
2. local
3. metro
4. crime\_is\_up

Per impostazione predefinita la funzione restituisce `FALSE` se il segmento indicato non esiste. Esiste anche un parametro secondario opzionale che permette di impostare autonomamente il valore predefinito se il segmento non esiste. Per esempio la funzione successiva restituisce zero se il segmento non esiste.

```
$product_id = $this->uri->segment(3, 0);
```

Questa funzione consente di evitare di dover scrivere un codice come questo:

```
if ($this->uri->segment(3) === FALSE)
{
    $product_id = 0;
}
else
{
    $product_id = $this->uri->segment(3);
}
```

- `$this->uri->rsegment(n)` questa funzione è identica a quella precedente, tranne che consente di recuperare un segmento specifico dal proprio URI re-indirizzato. Questo, nel caso in cui si stia usando la funzione URI Routing di CodeIgniter (si veda a pagina [37](#)).

- `$this->uri->slash_segment(n)` si differenzia da `$this->uri->segment()` solo per il fatto che aggiunge uno slash al segmento, la cui posizione è determinata dal secondo parametro. Se il parametro non viene utilizzato, viene aggiunto uno slash alla fine. Ecco alcuni esempi:

```
$this->uri->slash_segment(3); // slash finale
$this->uri->slash_segment(3, 'leading'); // slash iniziale
$this->uri->slash_segment(3, 'both'); // slash all'inizio e alla fine
```

Restituiscono rispettivamente:

- `segment/`
- `/segment`
- `/segment/`
- `$this->uri->slash_rsegment(n)` questa funzione identica alla precedente aggiunge uno slash all'URI reinstradato (si deve utilizzare la funzionalità URI Rounting).
- `$this->uri->uri_to_assoc(n)` trasforma un segmento URI in un array associativo di chiavi/coppie di valori. Si consideri il seguente URI:

```
index.php/user/search/name/joe/location/UK/gender/male
```

L'URI verrà così trasformato in un array associativo come nel prototipo seguente:

```
[array]
(
    'name' => 'joe'
    'location' => 'UK'
    'gender' => 'male'
)
```

Il primo parametro della funzione definisce un offset che per valore predefinito è uguale a 3. Il secondo parametro permette di impostare i nomi delle key predefinite, in modo che l'array restituito dalla funzione conterrà sempre gli indici attesi, anche se manca l'URI. Per esempio:

```
$default = array('name', 'gender', 'location', 'type', 'sort');

$array = $this->uri->uri_to_assoc(3, $default);
```

Se l'URI non contiene un valore predefinito, verrà impostato su quel nome un indice di array con un valore FALSE. Infine, se un valore corrispondente non è disponibile per una determinata chiave (se vi è un numero dispari di segmenti URI) il valore verrà impostato su FALSE.

- `$this->uri->ruri_to_assoc(n)` si differenzia dalla precedente funzione per il fatto che viene creato un array associativo utilizzando l'URI reindirizzato.
- `$this->uri->assoc_to_uri()` prende un array associativo e genera da esso una stringa URI. Le chiavi dell'array saranno incluse nella stringa. Esempio:

```
$array = array('product' => 'shoes', 'size' => 'large', 'color' => 'red')
;

$str = $this->uri->assoc_to_uri($array);

// Produce: product/shoes/size/large/color/red
```

- `$this->uri->uri_string()` restituisce una stringa con un URI completo. Per esempio se il proprio URL completo è il seguente:

```
http://example.com/index.php/news/local/345
```

La funzione restituisce:

```
news/local/345
```

- `$this->uri->ruri_string()` si differenzia dalla funzione precedente per il fatto che restituisce l'URI reindirizzato.
- `$this->uri->total_segments()` restituisce il numero totale di segmenti.
- `$this->uri->total_rsegments()` restituisce il numero totale di segmenti nell'URI reindirizzato.
- `$this->uri->segment_array()` restituisce un array che contiene i segmenti dell'URI. Per esempio:

```
$segs = $this->uri->segment_array();  
  
foreach ($segs as $segment)  
{  
    echo $segment;  
    echo '<br />';  
}
```

- `$this->uri->rsegment_array()` questa funzione è identica a quella precedente, tranne che restituisce l'array di segmenti dell'URI re-indirizzato nel caso in cui si stia usando la funzione URI Routing di CodeIgniter.

## 8.28 CLASSE USER AGENT

La classe User Agent fornisce degli utili strumenti per identificare le informazioni riguardanti i browser, i device mobile e i robot che si collegano al proprio sito. In aggiunta a questo, la classe rivela altri dati come il linguaggio e la codifica dei caratteri supportata.

La classe viene inizializzata nel proprio controller con la funzione `$this->load->library('user_agent')`. Una volta caricata, l'oggetto sarà disponibile con la funzione `$this->agent`.

### DEFINIZIONE

Le definizioni riguardanti lo “user agent” sono memorizzate nel file `/application/config/user_agents.php` in cui è possibile aggiungere ulteriori elementi se necessario. Per esempio, quando la classe è inizializzata è possibile determinare se lo user agent che naviga nel sito è un browser web, un dispositivo mobile o un robot. È anche possibile ottenere altre informazioni sulla piattaforma come il sistema operativo utilizzato.

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

if ($this->agent->is_browser())
{
    $agent = $this->agent->browser().' '.$this->agent->version();
}
elseif ($this->agent->is_robot())
{
    $agent = $this->agent->robot();
}
elseif ($this->agent->is_mobile())
{
    $agent = $this->agent->mobile();
}
else
{
    $agent = 'Unidentified User Agent';
}

echo $agent;

echo $this->agent->platform(); // Platform info (Windows, Linux, Mac, etc.)
```

## REFERENCE DELLE FUNZIONI

- `$this->agent->is_browser()` restituisce un valore booleano TRUE/FALSE se il browser web è un user agent conosciuto o meno:

```
if ($this->agent->is_browser('Safari'))
{
    echo 'You are using Safari.';
}
else if ($this->agent->is_browser())
{
    echo 'You are using a browser.';
}
```

La stringa “Safari”, presente in questo esempio è una chiave dell’array che contiene la lista dei browser web. Il file `application/config/user_agents.php` contiene la lista di tutti i browser a cui è possibile aggiungerne di nuovi o modificare quelli esistenti.

- `$this->agent->is_mobile()` restituisce un valore booleano TRUE/FALSE se il dispositivo mobile è uno user agent conosciuto o meno:

```
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_mobile()` restituisce un valore booleano TRUE/FALSE se il dispositivo mobile è uno user agent conosciuto o meno:



```
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');
}
```

- `$this->agent->is_robot()` restituisce un valore booleano TRUE/FALSE se il robot è uno user agent conosciuto o meno:  
Nota: La libreria degli user agent contiene solo le definizioni dei robot più comuni. Non è certo un elenco esaustivo dei bot: ne esistono centinaia e verificare ognuno di questi non sarebbe molto efficiente. In tutti i casi, se si scopre che alcuni bot che normalmente visitano il sito mancano nella lista è possibile aggiungerli manualmente al file `application/config/user_agents.php`.
- `$this->agent->is_referral()` restituisce il valore booleano TRUE/FALSE se lo user agent è riferito a quello di un altro sito.
- `$this->agent->browser()` restituisce una stringa che contiene il nome del browser web che visita il proprio sito.
- `$this->agent->version()` restituisce una stringa che contiene il numero di versione del browser web che visita il proprio sito.
- `$this->agent->mobile()` restituisce una stringa che contiene il nome del dispositivo mobile che visita il proprio sito.
- `$this->agent->robot()` restituisce una stringa che contiene il nome del robot che visita il proprio sito.
- `$this->agent->platform()` restituisce una stringa che contiene il nome sistema operativo (Linux, Windows, OS X, etc.) utilizzato per visitare il proprio sito.
- `$this->agent->referrer()` è una stringa che viene prodotta quando lo user agent è dovuto alla visita di un altro sito:

```
if ($this->agent->is_referral())
{
    echo $this->agent->referrer();
}
```

- `$this->agent->agent_string()` restituisce una stringa che contiene la stringa completa user agent. Un esempio:

```
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()` determina se lo user agent accetta una particolare lingua. Questa funzione non è in genere molto affidabile in quanto alcuni browser non forniscono informazioni sulla lingua, e tra quelli che lo fanno, il dato non è sempre preciso.

```
if ($this->agent->accept_lang('en'))  
{  
    echo 'You accept English!';  
}
```

- `$this->agent->accept_charset()` determina se lo user agent accetta una codifica dei caratteri. Questa funzione non è in genere molto affidabile in quanto alcuni browser non forniscono informazioni sulla codifica dei caratteri, e tra quelli che lo fanno, il dato non è sempre preciso. Per esempio:

```
if ($this->agent->accept_charset('utf-8'))  
{  
    echo 'You browser supports UTF-8!';  
}
```

## 8.29 CLASSE XML-RPC E XML-RPC SERVER

Le classi XML-RPC di CodeIgniter vengono utilizzate per inviare richieste ad un altro server, o impostare il proprio server XML-RPC per ricevere le richieste.

### INTRODUZIONE A XML-RPC

Molto semplicemente è un modo per due computer di comunicare su Internet utilizzando XML. Un computer, che chiameremo client, invia una richiesta XML-RPC ad un altro computer, che chiameremo server. Una volta che il server riceve ed elabora la richiesta invierà una risposta al client.

Ad esempio, utilizzando l'API MetaWeblog, un client XML-RPC (di solito uno strumento di desktop publishing), invierà una richiesta a un server XML-RPC in esecuzione sul proprio sito. Questa richiesta potrebbe essere una nuova voce blog inviata per relativa pubblicazione, o potrebbe essere la richiesta di una voce esistente per la sua modifica. Quando il server XML-RPC riceve questa richiesta, la esaminerà per determinare quale classe/metodo si deve richiamare per elaborare la richiesta. Una volta elaborata, il server invierà indietro un messaggio di risposta. Per le specifiche dettagliate, si visiti il sito XML-RPC: <http://xmlrpc.scripting.com/>.

Le classi XML-RPC e XML-RPC vengono inizializzate nel proprio controller utilizzando la funzione `$this->load->library`. La classe XML-RPC è caricata con la seguente istruzione:

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

Una volta caricata, la libreria oggetto sarà disponibile usando: `$this->xmlrpc`. Per caricare la classe XML-RPC Server si utilizzerà:

```
$this->load->library('xmlrpc');  
$this->load->library('xmlrpcs');
```

Anche questa classe, una volta caricata, fornirà una libreria oggetto utilizzabile con `$this->xmlrpcs`. Quando si usa la classe XML-RPC Server è necessario caricare ambedue le classi XML-RPC e XML-RPC Server.

### REQUEST XML-RPC

Per inviare una richiesta ad un server XML-RPC si devono specificare le seguenti informazioni:

- l'URL del server
- il metodo sul server che si desidera caricare

- i dati request (spiegati più avanti)

Qui di seguito viene descritto un semplice esempio che spedisce un semplice ping Weblogs.com verso Ping-o-matic: <http://pingomatic.com/>:

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

$this->xmlrpc->server('http://rpc.pingomatic.com/', 80);
$this->xmlrpc->method('weblogUpdates.ping');

$request = array('My Photoblog', 'http://www.my-site.com/photoblog/');
$this->xmlrpc->request($request);

if ( ! $this->xmlrpc->send_request() )
{
    echo $this->xmlrpc->display_error();
}
```

#### Spiegazione

Il codice di qui sopra inizializza la classe XML-RPC, imposta l'URL del server e il metodo da chiamare (weblogUpdates.ping). La richiesta (in questo caso il titolo e l'URL del sito) viene inserita in un array per il trasporto, e compilata utilizzando la funzione **request()**. Infine, la richiesta completa viene inviata. Se il metodo `send_request()` restituisce FALSE si potrà visualizzare il messaggio di errore restituito dal server XML-RPC.

#### ANATOMIA DI UN REQUEST

Una richiesta XML-RPC è semplicemente il dato che si sta inviando al server XML-RPC. Ogni pezzo di informazione in una richiesta è riferito ad un parametro request. L'esempio di qui sopra ha due parametri: l'indirizzo e il titolo del sito. Quando il server XML-RPC riceve la richiesta, cercherà i parametri di cui ha bisogno. I parametri request devono essere collocati in un array per il trasporto, e ogni parametro può essere uno tra i sette tipi di dati consentiti (stringhe, numeri, date, ecc.). Se i parametri sono qualcosa di diverso dalle stringhe, si dovrà includere il tipo di dati nell'array request.

Questo è un esempio di un semplice array con tre parametri:

```
$request = array('John', 'Doe', 'www.some-site.com');
$this->xmlrpc->request($request);
```

Se si utilizzano tipi di dati diversi dalle stringhe, o se si dispone di diversi tipi di dati, si può inserire ogni parametro nel proprio array, con il tipo di dati in seconda posizione:

```
$request = array (
    array('John', 'string'),
    array('Doe', 'string'),
    array(FALSE, 'boolean'),
    array(12345, 'int')
);
$this->xmlrpc->request($request);
```

La sezione dei Dati Type seguente presenta una lista di tutti i tipi di dati.

## CREARE UN SERVER XML-RPC

Un server XML-RPC agisce come un vigile urbano: attende le richieste in arrivo e le redirige verso le funzioni appropriate per l'elaborazione. Per creare il proprio server XML-RPC è necessario inizializzare la classe XML-RPC Server nel proprio controller in cui si prevede arriveranno le request. Quindi, si deve creare un array con le istruzioni che permettono alle request in ingresso di essere inviate alla classe e al metodo appropriato per l'elaborazione.

Ecco un esempio:

```
$this->load->library('xmlrpc');
$this->load->library('xmlrpcs');

$config['functions']['new_post'] = array('function' => 'My_blog.new_entry'),
$config['functions']['update_post'] = array('function' => 'My_blog.
    update_entry');
$config['object'] = $this;

$this->xmlrpcs->initialize($config);
$this->xmlrpcs->serve();
```

L'esempio precedente contiene un array che specifica due metodi request consentiti dal Server. I metodi accettati sono sul lato sinistro dell'array (new\_post e update\_post): quando uno di questi viene ricevuto, saranno mappati sulla classe e il metodo specificato sulla destra.

La chiave "object" è una chiave speciale a cui si passa un oggetto di classe istanziata. Essa è necessaria quando il metodo che si sta mappando non fa parte del super oggetto di CodeIgniter.

In altre parole, se un client XML-RPC invia una richiesta per il metodo new\_post, il proprio server caricherà la classe My\_blog e chiamerà la funzione new\_entry. Se la

richiesta è per il metodo `update_post`, allora il server caricherà la classe `My_blog` e invocherà la funzione `update_entry`.

I nomi delle funzioni nell'esempio di cui sopra sono arbitrari: si potrà decidere cosa devono richiamare sul server, oppure i nomi delle funzioni se si utilizzano delle API standard, come Blogger o API MetaWeblog.

Ci sono due chiavi di configurazione aggiuntive che si possono utilizzare quando si inizializza la classe di server: **debug** può essere impostata su `TRUE` per attivare il debug, e **xss\_clean** può essere impostata su `FALSE` per evitare l'invio di dati attraverso la funzione `xss_clean` della library `Security`.

## ELABORARE LE REQUEST SERVER

Quando il server XML-RPC riceve un request e carica la classe/metodo per l'elaborazione, esso passerà un oggetto a tale metodo contenente i dati inviati dal client. Utilizzando l'esempio precedente, se viene richiesto il metodo `new_post`, il server si aspetterà una classe come in questo prototipo:

```
class My_blog extends CI_Controller {  
  
    function new_post($request)  
    {  
  
    }  
  
}
```

La variabile **\$request** è un oggetto compilato dal Server, che contiene i dati inviati dal Client XML-RPC. Utilizzando questo oggetto si avrà accesso ai parametri request che permettono di elaborare la richiesta. Alla fine verrà inviato una Response al client. Qui di seguito è presentato un esempio reale, utilizzando l'API di Blogger. Uno dei metodi della API di Blogger è **getUserInfo()**: usando questo metodo, un Client XML-RPC può inviare al server un nome utente e una password per poi ricevere in cambio informazioni su quel particolare utente (nick, ID utente, indirizzo email, ecc.).

```

class My_blog extends CI_Controller {

    function getUserInfo($request)
    {
        $username = 'smitty';
        $password = 'secretsmittypass';

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

        $parameters = $request->output_parameters();

        if ($parameters['1'] != $username AND $parameters['2'] != $password)
        {
            return $this->xmlrpc->send_error_message('100', 'Invalid Access');
        }

        $response = array(array('nickname' => array('Smitty','string'),
                                'userid'   => array('99','string'),
                                'url'      => array('http://yoursite.com','
string'),
                                'email'    => array('jsmith@yoursite.com','
string'),
                                'lastname' => array('Smith','string'),
                                'firstname' => array('John','string')
                                ),
                            'struct');

        return $this->xmlrpc->send_response($response);
    }
}

```

Nota: la funzione `output_parameters()` recupera un array indicizzato corrispondente ai parametri request inviati dal client. Nell'esempio precedente, i parametri di output saranno il nome utente e la password.

Se il nome utente e la password inviati dal client non sono validi, un messaggio di errore verrà restituito utilizzando `send_error_message()`.

Se l'operazione ha avuto successo, il client farà tornare indietro un array di risposta contenente le informazioni dell'utente.

## FORMATTARE UN RESPONSE

Come avviene per le Request, anche le Response devono essere formattate come un array. Tuttavia, a differenza delle request, un response è un array che contiene un singolo elemento che può essere un array con ulteriori array al suo interno, anche se può esserci solo un indice di array primario. In altre parole, il prototipo di base è questo:

```
$response = array('Response data', 'array');
```

Le Response però, di solito contengono più pezzi di informazione. Per realizzare ciò, si deve inserire la risposta nel proprio array in modo che l'array primario continui a contenere un singolo pezzo di dati. Ecco un esempio che mostra come questo potrebbe essere realizzato:

```
$response = array (
    array(
        'first_name' => array('John', 'string'),
        'last_name' => array('Doe', 'string'),
        'member_id' => array(123435, 'int'),
        'todo_list' => array(array('clean house', 'call mom',
            'water plants'), 'array'),
    ),
    'struct'
);
```

Si noti che l'array di cui sopra viene formattato come una struct (struttura). Questo è il tipo più comune di dati per le response. Come con le Request, una response può essere uno dei sette tipi di dati elencati nella sezione Data Type.

## INVIARE UN ERROR RESPONSE

Se è necessario inviare al client un error response si userà la seguente istruzione:

```
return $this->xmlrpc->send_error_message('123', 'Requested data not available'
);
```

Il primo parametro è il numero di errore, mentre il secondo è la stringa contenente il messaggio di errore.

## CREARE IL PROPRIO CLIENT E SERVER

Per aiutare a comprendere tutto quello che abbiamo visto finora, si creino un paio di controller che agiscano come client XML-RPC e server. In questo modo si potrà utilizzare il client per inviare una richiesta al server e ricevere una risposta.



## IL CLIENT

Utilizzando un editor di testo, si crei un controller chiamato `xmlrpc_client.php`. In esso si inserisca questo codice e lo si salvi nella cartella `/applications/controllers/`:

```
<?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('How is it going?');
        $this->xmlrpc->request($request);

        if ( ! $this->xmlrpc->send_request() )
        {
            echo $this->xmlrpc->display_error();
        }
        else
        {
            echo '<pre>';
            print_r($this->xmlrpc->display_response());
            echo '</pre>';
        }
    }
}
?>
```

Nota: nel codice precedente è stato utilizzato un Helper URL (si veda la sezione [5.5 a pagina 70](#)).

## IL SERVER

Utilizzando un editor di testo, si crei un controller chiamato `xmlrpc_server.php`. In esso si inserisca questo codice e lo si salvi nella cartella `/applications/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' => 'Not bad at all.'),
            'struct');

        return $this->xmlrpc->send_response($response);
    }
}
?>
```

## SIAMO PRONTI!

Ora, si inserisca l'URL del proprio sito:

```
example.com/index.php/xmlrpc_client/
```

Si visualizzerà il messaggio inviato al server e la risposta restituita. Il client creato, invia un messaggio (How's is going?) al server, insieme con un request per i Greetings. Il server riceve la richiesta e la mappa alla funzione processo, con una risposta.

## UTILIZZARE GLI ARRAY ASSOCIATIVI

Se si vuole utilizzare un array associativo nei parametri del proprio metodo, è necessario utilizzare un tipo di dati “datatype struct”:

```
$request = array(  
    array(  
        // Param 0  
        array(  
            'name'=>'John'  
        ),  
        'struct'  
    ),  
    array(  
        // Param 1  
        array(  
            'size'=>'large',  
            'shape'=>'round'  
        ),  
        'struct'  
    )  
);  
$this->xmlrpc->request($request);
```

L'array associativo si può poi recuperare quando il request sarà elaborata nel Server.

```
$parameters = $request->output_parameters();  
$name = $parameters['0']['name'];  
$size = $parameters['1']['size'];  
$shape = $parameters['1']['shape'];
```

## REFERENCE DELLE FUNZIONI XML-RPC

- `$this->xmlrpc->server()` imposta l'URL e il numero della porta del server a cui il request deve essere inviato:

```
$this->xmlrpc->server('http://www.sometimes.com/pings.php', 80);
```

`$this->xmlrpc->timeout()` imposta il tempo (in secondi) dopo il quale il request verrà cancellato:

```
$this->xmlrpc->timeout(6);
```

- `$this->xmlrpc->method()` definisce il metodo che sarà richiesto dal server XML-RPC:

```
$this->xmlrpc->method('method');
```

Dove il **method** è il nome del metodo.

- `$this->xmlrpc->request()` viene preso un array di dati e costruisce un request per essere inviato al server XML-RPC.

```
$request = array(array('My Photoblog', 'string'), 'http://www.yoursite.com/photoblog/');  
$this->xmlrpc->request($request);
```

- `$this->xmlrpc->send_request()` la funzione di invio request, restituisce un valore booleano TRUE/FALSE sulla base del suo successo o fallimento. La si abilita per essere usata in una struttura condizionale.
- `$this->xmlrpc->set_debug(TRUE)` viene abilitato il debug, con cui si visualizzano molte informazioni, errori che aiutano nello sviluppo
- `$this->xmlrpc->display_error()` restituisce un messaggio di errore sotto forma di stringa se il request fallisce

```
echo $this->xmlrpc->display_error();
```

- `$this->xmlrpc->display_response()` restituisce la risposta di un server remoto quando il request viene ricevuto. La risposta sarà generalmente un array associativo

```
$this->xmlrpc->display_response();
```

- `$this->xmlrpc->send_error_message()` questa funzione consente di inviare un messaggio di errore dal server al client. Il primo parametro è il numero di errore mentre il secondo è il messaggio di errore.

```
return $this->xmlrpc->send_error_message('123', 'Requested data not
    available');
```

- `$this->xmlrpc->send_response()` consente di inviare la risposta dal server al client. Un array di valori di dati validi deve essere inviato con questo metodo.

```
$response = array(
    array(
        'ferror' => array(FALSE, 'boolean'),
        'message' => "Thanks for the ping!"
    )
    'struct');
return $this->xmlrpc->send_response($response);
```

## DATA TYPE

Secondo le specifiche XML-RPC ci sono sette tipi di valori che è possibile inviare tramite XML-RPC:

- int oppure i4
- boolean
- string
- double
- dateTime.iso8601
- base64
- struct (contiene un array di valori)
- array (contiene un array di valori)

## 8.30 CLASSE ZIP ENCODING

La classe in oggetto permette di creare degli archivi Zip che successivamente possono essere scaricati nel proprio desktop o salvati in una directory.

### INIZIALIZZARE LA CLASSE

La classe è inizializzata nel controller con la funzione **`$this->load->library`**:

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

Una volta caricata la classe, l'oggetto della libreria Zip sarà disponibile usando **`$this->zip`**:

### ESEMPIO DI UTILIZZO

Questo esempio mostra come comprimere un file, salvarlo in una cartella sul server e scaricarla sul desktop.

```
$name = 'mydata1.txt';  
$data = 'A Data String!';  
  
$this->zip->add_data($name, $data);  
  
// Scrive il file zip (my_backup.zip) in una cartella del server  
$this->zip->archive('/path/to/directory/my_backup.zip');  
  
// Scarica il file sul Desktop  
$this->zip->download('my_backup.zip');
```

### REFERENCE

- **`$this->zip->add_data()`** consente di aggiungere i dati all'archivio Zip. Il primo parametro deve contenere il nome che si vuole dare al file, mentre il secondo parametro contiene il file vero e proprio come una stringa:

```
$name = 'my_bio.txt';  
$data = 'Sono nato in un ascensore...';  
  
$this->zip->add_data($name, $data);
```

Sono consentite più chiamate a questa funzione per aggiungere più file al proprio archivio. Per esempio:

```
$name = 'mydata1.txt';  
$data = 'A Data String!';  
$this->zip->add_data($name, $data);  
  
$name = 'mydata2.txt';  
$data = 'Another Data String!';  
$this->zip->add_data($name, $data);
```

Oppure si possono passare più file utilizzando un array:

```
$data = array(  
    'mydata1.txt' => 'A Data String!',  
    'mydata2.txt' => 'Another Data String!'  
);  
  
$this->zip->add_data($data);  
  
$this->zip->download('my_backup.zip');
```

Se si vogliono comprimere i dati organizzati in sottocartelle, è necessario includere il loro percorso come parte del filename:

```
$name = 'personal/my_bio.txt';  
$data = 'Sono nato in un ascensore...';  
  
$this->zip->add_data($name, $data);
```

L'esempio precedente inserirà my\_bio.txt dentro una cartella di nome **personal**.

- `$this->zip->add_dir()` consente di aggiungere una directory. Solitamente questa funzione non è necessaria poiché è possibile inserire i dati nelle directory quando si utilizza la funzione `$this->zip->add_data()`, a meno che non si voglia creare una cartella vuota (senza alcun file all'interno). Per esempio:

```
$this->zip->add_dir('myfolder'); // Crea una directory di nome "myfolder"
```

- `$this->zip->read_file()` permette di comprimere un file già esistente nel proprio server. Si fornisce il percorso del file e la classe Zip per leggere e aggiungere il file all'archivio.

```
$path = '/path/to/photo.jpg';  
  
$this->zip->read_file($path);  
  
// Scarica il file sul desktop  
$this->zip->download('my_backup.zip');
```

Se si vuole mantenere la struttura delle directory, è necessario passare un secondo parametro con il valore TRUE. Nel prossimo esempio il file **photo.jpeg** sarà inserito all'interno di due sottocartelle **path/to/**:

```
$path = '/path/to/photo.jpg';  
  
$this->zip->read_file($path, TRUE);  
  
// Scarica il file sul desktop  
$this->zip->download('my_backup.zip');
```

- `$this->zip->read_dir()` permette di comprimere una cartella e il suo contenuto già esistente sul server. Si fornisce il percorso della directory e la classe Zip che leggerà ricorsivamente la directory per generare un archivio zip. Tutti i file presenti nel percorso fornito alla funzione saranno codificati, così come tutte le sottodirectory. Per esempio:

```
$path = '/path/to/your/directory/';  
  
$this->zip->read_dir($path);  
  
// Scarica il file sul desktop  
$this->zip->download('my_backup.zip');
```

Per impostazione predefinita, l'archivio Zip inserirà tutte le directory elencate nel primo parametro all'interno dell'archivio zip. Se si desidera che, il percorso che precede la cartella di destinazione, sia ignorato si può passare il valore booleano FALSE nel secondo parametro. Per esempio:



```
$path = '/path/to/your/directory/';  
$this->zip->read_dir($path, FALSE);
```

Questo codice crea un archivio zip con la cartella “directory” e tutte le relative sottocartelle. Non sarà però incluso il percorso radice, ovvero **/path/to/your**

- `$this->zip->archive()` scrive i file compressi in una directory specifica sul server. È necessario fornire un percorso valido alla fine del filename ed essere sicuri che la directory di destinazione abbia i permessi di scrittura (solitamente 666 o 777). Per esempio:

```
$this->zip->archive('/path/to/folder/myarchive.zip'); // Creates a file  
named myarchive.zip
```

- `$this->zip->download()` fa sì che il file zip possa essere scaricato dal server. La funzione accetta come argomento il nome con cui il file zip sarà nominato nel momento del download. Per esempio:

```
$this->zip->download('latest_stuff.zip'); // Il file si chiamerà "  
latest_stuff.zip"
```

Nota: nel controller non si visualizzerà alcun dato relativo alla chiamata di questa funzione dato che essa invia vari server header (intestazioni di server) che determinano il download del file zip come file binario.

- `$this->zip->get_zip()` restituisce i file zip compressi. In genere non si avrà bisogno di questa funzione se non si vuole fare qualcosa di specifico con i dati. Esempio:

```
$name = 'my_bio.txt';  
$data = 'Sono nato in un ascensore...';  
  
$this->zip->add_data($name, $data);  
  
$zip_file = $this->zip->get_zip();
```

- `$this->zip->clear_data()` la classe memorizza nella cache i dati zip in modo che non si abbia bisogno di comprimerli per ogni funzione specificata precedentemente. Tuttavia se è necessario creare più Zip, ognuno con dati diversi, è possibile cancellare la cache tra le varie chiamate. Esempio:

```
$name = 'my_bio.txt';  
$data = 'Sono nato in un ascensore...';  
  
$this->zip->add_data($name, $data);  
$zip_file = $this->zip->get_zip();  
  
$this->zip->clear_data();  
  
$name = 'photo.jpg';  
$this->zip->read_file("/path/to/photo.jpg"); // Leggei contenuti del file  
  
$this->zip->download('myphotos.zip');
```

## 9 | ELENCO DEGLI HELPER

In questo capitolo verranno esaminate gli Helper presenti in CodeIgnitor che offrono funzionalità potenti e flessibili per lo sviluppo e la gestione del proprio progetto.

## 9.1 HELPER ARRAY

Le funzionalità di questo Helper aiutano nella gestione degli array. Il suo caricamento avviene con l'istruzione:

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

Qui di seguito si mettono in evidenza le funzioni disponibili:

- `element()` consente di recuperare un elemento da un array. La funzione verifica se l'indice dell'array esiste e se questo ha un valore. Se esiste un valore, questo viene restituito, se invece il valore non esiste viene restituito FALSE, o qualsiasi altra cosa si sia specificato come valore di default tramite il terzo parametro. Per esempio:

```
$array = array('colore' => 'rosso', 'forma' => 'circolare', 'dimensione'
=> '');

// restituisce "rosso"
echo element('colore', $array);

// restituisce NULL
echo element('dimensione', $array, NULL);
```

- `random_element()` viene preso un array in ingresso e restituito un elemento casuale. Un esempio è il seguente:

```
$quotes = array(
    "I find that the harder I work, the more luck I seem to have.
    - Thomas Jefferson",
    "Don't stay in bed, unless you can make money in bed. -
    George Burns",
    "We didn't lose the game; we just ran out of time. - Vince
    Lombardi",
    "If everything seems under control, you're not going fast
    enough. - Mario Andretti",
    "Reality is merely an illusion, albeit a very persistent one.
    - Albert Einstein",
    "Chance favors the prepared mind - Louis Pasteur"
);

echo random_element($quotes);
```

- `elements()` consente di ottenere il numero di elementi da un array. La funzionalità verifica se ciascuno degli indici dell'array è impostato: se un indice non

esiste, questo assume il valore su FALSE, o qualsiasi altra cosa si sia specificato come valore di default tramite il terzo parametro. Per esempio:

```
$array = array(  
    'colore' => 'rosso',  
    'forma' => 'circolare',  
    'raggio' => '10',  
    'diametro' => '20'  
);  
  
$my_shape = elements(array('colore', 'forma', 'altezza'), $array);
```

Il codice restituisce il seguente array:

```
array(  
    'color' => 'rosso',  
    'shape' => 'circolare',  
    'height' => FALSE // questo indice non esiste  
);
```

È possibile impostare il terzo parametro con qualsiasi valore di default desiderato:

```
$my_shape = elements(array('colore', 'forma', 'altezza'), $array, NULL);
```

L'esempio precedente restituisce il seguente array:

```
array(  
    'color' => 'rosso',  
    'shape' => 'circolare',  
    'altezza' => NULL // nuovo valore impostato  
);
```

Questo è utile quando si invia l'array \$\_POST ad uno dei propri Model poiché impedisce agli utenti di inviare ulteriori dati POST per essere inseriti nelle proprie tabelle:

```
$this->load->model('post_model');  
  
$this->post_model->update(elements(array('id', 'title', 'content'),  
    $_POST));
```

In questo modo, solo i campi id, title e content saranno aggiornati.

## 9.2 HELPER CAPTCHA

L'Helper Captcha mette a disposizione molte funzionalità per la gestione delle immagini captcha. Il suo caricamento avviene con la funzione:

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



Fig. 21: Captcha

Le funzioni disponibili sono le seguenti:

`create_captcha($data)` genera il captcha come un'immagine sulla base delle specifiche fornite, quindi restituisce un array di dati associativi sull'immagine.

```
[array]
(
  'image' => IMAGE TAG
  'time'  => TIMESTAMP (in microsecondi)
  'word'  => CAPTCHA WORD
)
```

L'“immagine” è il tag dell'immagine corrente:

```

```

La parola “time” è il nome dell'immagine per cui si utilizza un timestamp in microsecondi (senza l'estensione del file). È un numero come questo: 1139612155.3422. Invece “word” è la parola che appare nell'immagine captcha che, se non è fornita dalla funzione, sarà una stringa casuale.

## UTILIZZARE L'HELPER CAPTCHA

Una volta caricato l'helper è possibile generare un captcha come nel seguente codice:

```
$vals = array(
    'word'    => 'Parola casuale',
    'img_path' => './captcha/',
    'img_url'  => 'http://example.com/captcha/',
    'font_path' => './path/to/fonts/teb.ttf',
    'img_width' => '150',
    'img_height' => 30,
    'expiration' => 7200
);

$cap = create_captcha($vals);
echo $cap['image'];
```

- la funzione captcha richiede la libreria GD
- sono richieste unicamente img\_path e img\_url
- se non viene fornita una “word”, la funzione genererà una stringa ASCII casuale. Altrimenti si potrebbe usare una libreria per la loro generazione casuale.
- se non viene specificato un percorso per il font TRUE TYPE, sarà utilizzato quello nativo (e decisamente brutto) della libreria GD
- la directory “captcha” deve avere i permessi di scrittura (solitamente 666 o 777)
- l’indice “expiration” espresso in secondi, indica per quanto tempo l’immagine rimarrà nella cartella captcha prima di essere eliminata. Il valore predefinito è di due ore.

## AGGIUNGERE UN DATABASE

Affinché la funzione captcha funzioni correttamente, sarà necessario aggiungere le informazioni restituite dalla funzione create\_captcha() al proprio database. Poi, quando l’utente compilerà il form captcha e invierà i dati, si verificherà se questi sono presenti nel database e che non siano scaduti.

Ecco un prototipo della tabella:

```
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`)
);
```



Qui vi è un esempio di utilizzo dell'Helper con un database. Nella pagina dove verrà visualizzato il captcha, si avrà un codice simile a questo:

```
$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="" />';
```

Quindi nella pagina che accetta l'invio dei dati, il codice sarà come il seguente:

```
// Prima di tutto si cancellano i vecchi capthca
$expiration = time()-7200; // Limite di due ore
$this->db->query("DELETE FROM captcha WHERE captcha_time < ".$expiration);

// Quindi se esiste 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 "Scrivi la parola che compare nell'immagine";
}
```

## 9.3 HELPER COOKIE

L'Helper viene caricato con il seguente codice:

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

Sono tre le funzioni messe a disposizione dall'Helper che assistono lo sviluppatore nel lavoro con i cookie:

- `set_cookie()` questa funzione fornisce alla propria Vista una sintassi intuitiva per l'impostazione dei cookie del browser. Per maggiori informazioni, si faccia riferimento alla classe `Input` a pagina [291](#).
- `get_cookie()` fornisce alla propria Vista una sintassi intuitiva per recuperare i cookie del browser. Per maggiori informazioni, si faccia riferimento alla classe `Input` a pagina [291](#).
- `delete_cookie()` cancella un cookie specificando il solo nome:

```
delete_cookie("name");
```

Questa funzione è identica a `set_cookie()`, tranne che non è caratterizzata da un valore e da parametri di scadenza. È possibile inviare un array di valori nel primo parametro oppure si possono impostare i singoli parametri.

```
delete_cookie($name, $domain, $path, $prefix)
```

## 9.4 HELPER DATE

Il caricamento di questo Helper fornisce alcune funzioni di aiuto nella gestione delle date:

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

Di seguito una serie di funzioni che permettono di gestire al meglio i formati e i valori “temporali”.

- `now()` restituisce l’ora attuale come timestamp Unix, facendo riferimento all’ora locale del server oppure GMT, sulla base del riferimento temporale definito nel file di configurazione. Se non si ha intenzione di impostare l’orario principale sul formato GMT (generalmente adottato se il sito permette ad ogni utente di impostare il proprio fuso orario) non si ricava alcun beneficio dall’utilizzo di questa funzione rispetto a quella `time()` del PHP.
- `mdate()` questa funzione è identica a quella del PHP `date()`, tranne per il fatto che utilizza i codici delle date in stile MySQL: ogni lettera che definisce una parte della data è preceduta dal segno percentuale (%Y %m %d etc.). Il vantaggio di questo approccio è che non ci si deve preoccupare di fare l’escaping dei caratteri che non sono codici riferiti ad una data, come si farebbe normalmente con la funzione PHP `date()`. Per esempio:

```
$datestring = "Year: %Y Month: %m Day: %d - %h:%i %a";  
$time = time();  
  
echo mdate($datestring, $time);
```

Se non viene incluso un timestamp nel secondo parametro, verrà utilizzata la data attuale.

- `standard_date()` permette di generare una stringa con la data in uno dei formati standard. Per esempio:

```
$format = 'DATE_RFC822';  
$time = time();  
  
echo standard_date($format, $time);
```

Il primo parametro deve contenere il formato, mentre il secondo, la data nel formato timestamp Unix. Qui di seguito un elenco dei formati supportati:

Costante	Descrizione	Esempio
DATE_ATOM	Atom	2005-08-15T16:13:03 +0000
DATE_COOKIE	HTTP Cookies	Sun, 14 Aug 2005 16:13:03 UTC
DATE_ISO8601	ISO-8601	2005-08-14T16:13:03 +00:00
DATE_RFC822	RFC 822	Sun, 14 Aug 05 16:13:03 UTC
DATE_RFC850	RFC 850	Sunday, 14-Aug-05 16:13:03 UTC
DATE_RFC1036	RFC 1036	Sunday, 14-Aug-05 16:13:03 UTC
DATE_RFC1123	RFC 1123	Sun, 14 Aug 2005 16:13:03 UTC
DATE_RFC2822	RFC 2822	Sun, 14 Aug 2005 16:13:03 +0000
DATE_RSS	RSS	Sun, 14 Aug 2005 16:13:03 UTC
DATE_W3C	WWW Consortium	2005-08-14T16:13:03 +0000

- `local_to_gmt()` prende un timestamp in formato Unix e lo restituisce nel formato GMT:

```
$now = time();

$gmt = local_to_gmt($now);
```

- `gmt_to_local()` prende un timestamp Unix (riferito al formato GMT) e lo converte in un timestamp locale, basato sul fuso orario e l'ora legale forniti:

```
$timestamp = '1140153693';
$timezone = 'UM8';
$daylight_saving = TRUE;

echo gmt_to_local($timestamp, $timezone, $daylight_saving);
```

- `mysql_to_unix()` prende in ingresso un timestamp in formato MySQL e lo restituisce nel formato Unix. Per esempio:

```
$mysql = '20061124092345';

$unix = mysql_to_unix($mysql);
```

- `unix_to_human()` prende un timestamp Unix e restituisce una data in formato comprensibile, detto "human":

```
YYYY-MM-DD HH:MM:SS AM/PM
```

Si tratta di una funzione utile, se si ha la necessità di visualizzare una data in un formato leggibile all'interno del proprio form. La data può essere personalizzata con o senza i secondi, utilizzando il formato europeo o americano. Se viene fornito solo il timestamp, sarà restituito il tempo senza secondi nel formato americano:

```
$now = time();

echo unix_to_human($now); // data americana senza secondi

echo unix_to_human($now, TRUE, 'us'); // data americana con i secondi

echo unix_to_human($now, TRUE, 'eu'); // data europea con i secondi
```

- `human_to_unix()` questa funzione svolge il compito opposto alla funzione precedente. Prende una data “human” e la restituisce nel formato Unix. La funzione è utile per acquisire una data impostata dall'utente in un form, e ottenere un timestamp Unix utilizzabile nelle operazioni di elaborazione. Viene restituito un valore booleano FALSE se la data non è formattata come indicato nella funzione precedente. Per esempio:

```
$now = time();

$human = unix_to_human($now);

$unix = human_to_unix($human);
```

- `timespan()` formatta un timestamp Unix in modo che appaia in modo simile a:

```
1 Year, 10 Months, 2 Weeks, 5 Days, 10 Hours, 16 Minutes
```

Il primo parametro è un timestamp Unix, mentre il secondo deve contenere un timestamp più grande di quello inserito nel primo parametro. Se il secondo parametro è vuoto, verrà utilizzato il timestamp corrente. Generalmente lo scopo di questa funzione è quello di visualizzare il tempo trascorso da un certo periodo passato sino ad oggi:

```
$post_date = '1079621429';
$now = time();

echo timespan($post_date, $now);
```

il testo generato da questa funzione si trova nel seguente file della lingua /language/<laproprialingua>/date\_lang.php.

- `days_in_month()` restituisce il numero di giorni in un certo mese/anno (tiene in considerazione anche gli anni bisestili):

```
echo days_in_month(06, 2005);
```

Se il secondo parametro è vuoto, sarà usato l'anno corrente.

- `timezones()` prende un fuso orario di riferimento e restituisce il ore di differenza rispetto all'UTC:

```
echo timezones('UM5');
```

Questa funzione si dimostra utile quando è abbinata a `timezone_menu()`.

- `timezone_menu()` genera un menu a scorrimento di fus orari come:

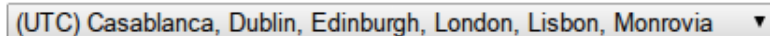


Fig. 22: Menu Timezone

Questo menu è utile se utilizzato in un sito in cui gli utenti possono impostare il fuso orario locale.

Il primo parametro permette di impostare lo stato select del menu. Ad esempio, per impostare il fuso orario del Pacifico come predefinito si usa:

```
echo timezone_menu('UM8');
```

Il secondo parametro imposta un nome di classe CSS per il menu.

Nota: il testo nel menu si trova nel file della lingua /language/<laproprialingua>/date\_lang.php

Fuso orario	UTC	Nazione
UM12	(UTC - 12:00)	Enitwetok, Kwajalien
UM11	(UTC - 11:00)	Nome, Midway Island, Samoa
UM10	(UTC - 10:00)	Hawaii
UM9	(UTC - 9:00)	Alaska
UM8	(UTC - 8:00)	Pacific Time
UM7	(UTC - 7:00)	Mountain Time
UM6	(UTC - 6:00)	Central Time, Mexico City
UM5	(UTC - 5:00)	Eastern Time, Bogota, Lima, Quito
UM4	(UTC - 4:00)	Atlantic Time, Caracas, La Paz
UM25	(UTC - 3:30)	Newfoundland
UM3	(UTC - 3:00)	Brazil, Buenos Aires, Georgetown, Falkland Is.
UM2	(UTC - 2:00)	Mid-Atlantic, Ascension Is., St Helena
UM1	(UTC - 1:00)	Azores, Cape Verde Islands
UTC	(UTC)	Casablanca, Dublin, Edinburgh, London, Lisbon, Monrovia
UP1	(UTC + 1:00)	Berlin, Brussels, Copenhagen, Madrid, Paris, Rome
UP2	(UTC + 2:00)	Kaliningrad, South Africa, Warsaw
UP3	(UTC + 3:00)	Baghdad, Riyadh, Moscow, Nairobi
UP25	(UTC + 3:30)	Tehran
UP4	(UTC + 4:00)	Adu Dhabi, Baku, Muscat, Tbilisi
UP35	(UTC + 4:30)	Kabul

## 9.5 HELPER DIRECTORY

Questo Helper, che offre dei metodi per lavorare più agilmente con le directory, è caricato mediante il codice:

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

Qui di seguito un elenco di funzioni disponibili:

`directory_map('source directory')` questa funzione legge il percorso della directory specificata nel primo parametro e produce una array di tutti i file contenuti in essa e nelle sottodirectory. Per esempio:

```
$map = directory_map('./mydirectory/');
```

Nota: il percorso deve essere sempre definito relativamente al file **index.php** nella directory root.

É possibile gestire il livello di profondità con cui saranno mappati i file nella directory attraverso un secondo parametro. Il suo valore specificato nel formato intero rappresenterà la profondità di mappatura. Per esempio, impostando il secondo parametro su "1" verrà mappata solo la directory definita nel primo parametro.

```
$map = directory_map('./mydirectory/', 1);
```

Per impostazione predefinita, i file nascosti non saranno inclusi nell'array, a meno che non si imposti un terzo parametro con il valore booleano TRUE.

```
$map = directory_map('./mydirectory/', FALSE, TRUE);
```

Ogni nome di directory verrà definito come indice dell'array e i suoi file saranno elencati in ordine numerico. Qui di seguito un esempio di un tipico array:



```
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
        )
)
```

## 9.6 HELPER DOWNLOAD

L'Helper che fornisce di scaricare i dati sul proprio desktop, viene caricato con l'istruzione:

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

`force_download('filename', 'data')` genera i server header che forza il download dei dati sul proprio desktop. Questa funzione è utilizzata per scaricare i file sul pc locale. Il primo parametro è il nome che si vuole dare al file scaricato, mentre il secondo parametro è il nome del file da scaricare. Per esempio:

```
$data = 'qui inserisci il testo';  
$name = 'mytext.txt';  
  
force_download($name, $data);
```

Se si vuole scaricare dal server un file esistente, sarà necessario leggere il file in una stringa:

```
$data = file_get_contents("/path/to/photo.jpg"); // Read the file's contents  
$name = 'myphoto.jpg';  
  
force_download($name, $data);
```

## 9.7 HELPER EMAIL

L'Helper fornisce assistenza quando si lavora con le Email. Per funzionalità più potenti comunque si rimanda alla classe Email a pagina [227](#).

L'Helper si carica con l'istruzione:

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

`valid_email('email')` controlla se una email è correttamente formattata, ma non fornisce alcuna garanzia o prova che l'email sarà ricevuta dal destinatario. Il valore restituito può essere TRUE/FALSE.

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

if (valid_email('email@somesite.com'))
{
    echo 'email valida';
}
else
{
    echo 'email non valida';
}
```

`send_email('recipient', 'subject', 'message')` invia una email utilizzando la funzione nativa [PHP mail\(\)](#). Per funzionalità più potenti comunque si rimanda alla classe Email a pagina [227](#).

## 9.8 HELPER FILE

L'assistente permette di lavorare agevolmente con i file. Il suo caricamento avviene con il codice:

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

Qui di seguito una serie di funzioni disponibili:

- `read_file('path')` restituisce i dati contenuti nel file specificato nel percorso. Per esempio:

```
$string = read_file('./path/to/file.php');
```

Il percorso del file specificato può essere relativo o assoluto. La funzione restituisce FALSE in caso di insuccesso.

Nota: il percorso è relativo rispetto al file **index.php** nella directory root e non rispetto ai Controller o alle Viste. CodeIgniter usa un front controller, quindi i percorsi sono sempre relativi rispetto all'index principale.

Se il server esegue una restrizione `open_basedir` questa funzione potrebbe non funzionare con i file che si trovano in una directory superiore al /omissis

- `write_file('path', $data)` i dati vengono scritti nel file specificato nel percorso. Se il file non esiste la funzione lo creerà. Esempio:

```
$data = 'Some file data';

if ( ! write_file('./path/to/file.php', $data))
{
    echo 'Unable to write the file';
}
else
{
    echo 'File written!';
}
```

Facoltativamente, è possibile impostare la modalità di scrittura tramite il terzo parametro:

```
write_file('./path/to/file.php', $data, 'r+');
```

La modalità predefinita è **wb**. É consigliato consultare la guida PHP per le altre opzioni: <http://php.net/fopen>

Nota: Per utilizzare questa funzione i permessi dei file devono essere impostati per la scrittura (666, 777, ecc.) Se il file non esiste già, anche la directory che lo contiene deve avere i permessi di scrittura.

Nota: il percorso è relativo rispetto al file **index.php** nella directory root e non rispetto ai Controller o alle Viste. CodeIgniter usa un front controller, quindi i percorsi sono sempre relativi rispetto all'index principale.

- `delete_files('path')` cancella tutti i file contenuti nel percorso fornito. Per esempio:

```
delete_files('./path/to/directory/');
```

Se il secondo parametro è impostato su `TRUE`, ogni directory nel percorso specificato sarà eliminata. Per esempio:

```
delete_files('./path/to/directory/', TRUE);
```

Nota: anche in questo caso i file devono avere i permessi di scrittura oppure devono essere proprietà del sistema per poter essere eliminati.

- `get_filenames('path/to/directory/')` prende un percorso del server e restituisce un array contenente i nomi di tutti i file definiti al suo interno. Il percorso può essere opzionalmente aggiunto ai nomi dei file impostando il secondo parametro su `TRUE`.
- `get_dir_file_info('path/to/directory/', $top_level_only = TRUE)` dato un file e un percorso, restituisce il relativo nome, percorso, dimensione, data di modifica. Il secondo parametro permette di dichiarare esplicitamente quali informazioni si desidera vengano restituite. Le opzioni sono: `name`, `server_path`, `size`, `date`, `readable`, `writable`, `executable`, `fileperms`. Restituisce `FALSE` se il file non viene trovato.
- `get_mime_by_extension('file')` traduce un'estensione di file in un tipo mime basato sul file **config/mimes.php**. Restituisce `FALSE` se non è possibile determinare il tipo, o aprire il file di configurazione mime.

```
$file = "somefile.png";  
echo $file . ' is has a mime type of ' . get_mime_by_extension($file);
```

Nota: questo non è un modo preciso per determinare i tipi di file mime; deve essere utilizzato per scopi che non riguardano la sicurezza.

- `symbolic_permissions($perms)` prende i permessi numerici (come quelli restituiti da **fileperms()**) e restituisce la notazione simbolica standard dei permessi dei file.

```
echo symbolic_permissions(fileperms('./index.php'));  
  
// -rw-r--r--
```

- `octal_permissions($perms)` prende i permessi numerici (come quelli restituiti da `fileperms()`) e restituisce i permessi del file nella notazione ottale.

```
echo octal_permissions(fileperms('./index.php'));  
  
// 644
```

## 9.9 HELPER FORM

Per caricare gli Helper che aiutano nella gestione dei form, si utilizza l'istruzione:

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

Le funzioni disponibili sono:

`form_open()` crea un tag di apertura form con un URL di base definito sulla base dalle preferenze di configurazione. Questo eventualmente permetterà di aggiungere gli attributi del form e i campi di input hidden (nascosti), mentre aggiungerà sempre l'attributo `accept-charset` basato sul valore `charset` definito nel file di configurazione.

Il principale vantaggio nell'utilizzo di questo tag, piuttosto che codificare il proprio codice in HTML è che fornisce al proprio sito una maggiore portabilità nel caso in cui gli URL dovessero cambiare.

Ecco un semplice esempio:

```
echo form_open('email/send');
```

L'esempio precedente potrebbe creare un form che punta al vostro URL di base più il segmento URI "email/send", come questo:

```
<form method="post" accept-charset="utf-8" action="http://example.com/index.php  
/email/send" />
```

Aggiungere attributi

Gli attributi possono essere aggiunti passando un array associativo al secondo parametro:

```
$attributes = array('class' => 'email', 'id' => 'myform');  
  
echo form_open('email/send', $attributes);
```

L'esempio precedente creerà un form simile a:

```
<form method="post" accept-charset="utf-8" action="http://example.com/index.php  
/email/send" class="email" id="myform" />
```

## Aggiungere Campi di input nascosti

I campi hidden possono essere aggiunti passando un array associativo al terzo parametro:

```
$hidden = array('username' => 'Joe', 'member_id' => '234');  
  
echo form_open('email/send', '', $hidden);
```

Il form risultante sarà:

```
<form method="post" accept-charset="utf-8" action="http://example.com/index.php  
/email/send">  
<input type="hidden" name="username" value="Joe" />  
<input type="hidden" name="member_id" value="234" />
```

`form_open_multipart()` consente di generare campi di input nascosti. È possibile inviare una stringa name/value (nome/valore) per creare un campo:

```
form_hidden('username', 'johndoe');  
  
// Produce:  
  
<input type="hidden" name="username" value="johndoe" />
```

Oppure per creare campi multipli si può fornire un array associativo:

```
$data = array(  
    'name' => 'John Doe',  
    'email' => 'john@example.com',  
    'url' => 'http://example.com'  
);  
  
echo form_hidden($data);  
  
// Produce:  
  
<input type="hidden" name="name" value="John Doe" />  
<input type="hidden" name="email" value="john@example.com" />  
<input type="hidden" name="url" value="http://example.com" />
```

`form_input()` consente di generare un campo di inserimento testo. È necessario passare il nome e il valore del campo nel primo e nel secondo parametro della funzione:



```

$data = array(
    'name'      => 'username',
    'id'        => 'username',
    'value'     => 'johndoe',
    'maxlength' => '100',
    'size'      => '50',
    'style'     => 'width:50%',
);

echo form_input($data);

// Produce:

<input type="text" name="username" id="username" value="johndoe" maxlength="
    100" size="50" style="width:50%" />

```

Se si vuole che il form possa contenere altri dati aggiuntivi, come Javascript, è possibile passare questi ultimi come una stringa nel terzo parametro:

```

$js = 'onClick="some_function()"';

echo form_input('username', 'johndoe', $js);

```

`form_password()` questa funzione è identica in tutto e per tutto alla `form_input` di qui sopra, salvo che questa è utilizzata per i dati di tipo “password”.

`form_upload()` questa funzione è identica in tutto e per tutto la funzione `form_input` di qui sopra, salvo che questa è utilizzata per l’upload dei file.

`form_textarea()` questa funzione è identica in tutto e per tutto la funzione `form_input` di qui sopra, salvo che questa genera un tipo “textarea”. Nota: al posto degli attributi `maxlength` e `size` visti nell’esempio precedente, si utilizzano `file` e `cols`.

`form_dropdown()` consente di creare un campo dropdown (menu a selezione multipla). Il primo parametro conterrà il nome del campo, il secondo parametro conterrà un array associativo di opzioni, mentre il terzo parametro conterrà il valore che si desidera selezionare. È anche possibile passare un array di oggetti multipli attraverso il terzo parametro: in questo caso CodeIgniter creerà una selezione multipla. Per esempio:

```

$options = array(
    'small' => 'Small Shirt',
    'med'   => 'Medium Shirt',
    'large' => 'Large Shirt',
    'xlarge' => 'Extra Large Shirt',
);

$shirts_on_sale = array('small', 'large');

echo form_dropdown('shirts', $options, 'large');

// Produce:

<select name="shirts">
<option value="small">Small Shirt</option>
<option value="med">Medium Shirt</option>
<option value="large" selected="selected">Large Shirt</option>
<option value="xlarge">Extra Large Shirt</option>
</select>

echo form_dropdown('shirts', $options, $shirts_on_sale);

// Produce:

<select name="shirts" multiple="multiple">
<option value="small" selected="selected">Small Shirt</option>
<option value="med">Medium Shirt</option>
<option value="large" selected="selected">Large Shirt</option>
<option value="xlarge">Extra Large Shirt</option>
</select>

```

Se si vuole aprire `<select>` per inserire dati aggiuntivi, come un attributo `id` o JavaScript, è possibile passare questi dati come una stringa nel quarto parametro:

```

$js = 'id="shirts" onChange="some_function();"';

echo form_dropdown('shirts', $options, 'large', $js);

```

Se l'array passato come **\$options** è un array multidimensionale, allora `form_dropdown()` produrrà un `<optgroup>` con la chiave dell'array come label (etichetta).

`form_multiselect()` consente di creare un campo di selezione multipla. Il primo parametro conterrà il nome del campo, il secondo parametro conterrà un array associativo di opzioni, e il terzo parametro conterrà il valore oppure i valori che si desidera selezionare. L'utilizzo del parametro è identico a quello visto in `form_dropdown()` di cui sopra, tranne ovviamente il fatto che per il nome del campo sarà necessario utilizzare sintassi dell'array POST, ad esempio, **foo[]**.

`form_fieldset()` permette di generare i campi `fieldset/legend`:

```
echo form_fieldset('Address Information');
echo "<p>fieldset content here</p>\n";
echo form_fieldset_close();

// Produce:
<fieldset>
<legend>Address Information</legend>
<p>form content here</p>
</fieldset>
```

Simile ad altre funzioni, è possibile inviare un array associativo nel secondo parametro, se si vogliono impostare attributi aggiuntivi.

```
$attributes = array('id' => 'address_info', 'class' => 'address_info');
echo form_fieldset('Address Information', $attributes);
echo "<p>fieldset content here</p>\n";
echo form_fieldset_close();

// Produce:
<fieldset id="address_info" class="address_info">
<legend>Address Information</legend>
<p>form content here</p>
</fieldset>
```

`form_fieldset_close()` produce un tag di chiusura `</fieldset>`. L'unico vantaggio di questa funzione è che permette di passargli dei dati che saranno aggiunti dopo il tag. Per esempio:

```
$string = "</div></div>";

echo form_fieldset_close($string);

// Produce:
</fieldset>
</div></div>
```

`form_checkbox()` consente di generare un campo checkbox. Ecco un semplice esempio:

```
echo form_checkbox('newsletter', 'accept', TRUE);

// Produce:

<input type="checkbox" name="newsletter" value="accept" checked="checked" />
```

Il terzo parametro contiene un valore booleano TRUE/FALSE per determinare se la caselladeve avere il parametro check. È possibile anche passare a questa funzione un array.

```
$data = array(  
    'name'      => 'newsletter',  
    'id'        => 'newsletter',  
    'value'     => 'accept',  
    'checked'   => TRUE,  
    'style'     => 'margin:10px',  
);  
  
echo form_checkbox($data);  
  
// Produce:  
  
<input type="checkbox" name="newsletter" id="newsletter" value="accept"  
checked="checked" style="margin:10px" />
```

Come con altre funzioni, se si vuole che il tag contenga dati aggiuntivi, come JavaScript, è possibile passarli come una stringa nel quarto parametro:

```
$js = 'onClick="some_function()";  
echo form_checkbox('newsletter', 'accept', TRUE, $js)
```

`form_radio()` Questa funzione è identica in tutto e per tutto alla funzione `form_checkbox()` di qui sopra, salvo che è utilizzata per impostare un tipo radio.

`form_submit()` genera un pulsante Submit (Invio) per il form. Per esempio:

```
echo form_submit('mysubmit', 'Submit Post!');  
  
// Produce:  
  
<input type="submit" name="mysubmit" value="Submit Post!" />
```

Simile ad altre funzioni, è possibile usare un array associativo nel primo parametro se si preferisce impostare i propri attributi. Il terzo parametro consente di aggiungere dati aggiuntivi al form, come per esempio JavaScript.

`form_label()` permette di generare un `<label>`. Ecco un esempio:

```
echo form_label('What is your Name', 'username');

// Produce:
<label for="username">What is your Name</label>
```

Simile ad altre funzioni, è possibile usare un array associativo nel terzo parametro se si preferisce impostare gli attributi.

```
$attributes = array(
    'class' => 'mycustomclass',
    'style' => 'color: #000;',
);
echo form_label('What is your Name', 'username', $attributes);

// Produce:
<label for="username" class="mycustomclass" style="color: #000;">What is your
    Name</label>
```

`form_reset()` genera un pulsante standard reset. Il suo utilizzo è simile a quello della funzione `form_submit()`.

`form_button()` genera un pulsante standard. È possibile passare il nome del pulsante e il contenuto al primo e secondo parametro.

```
$data = array(
    'name' => 'button',
    'id' => 'button',
    'value' => 'true',
    'type' => 'reset',
    'content' => 'Reset'
);
echo form_button($data);

// Produce:
<button name="button" id="button" value="true" type="reset">Reset</button>
```

Se si vuole che il form contenere alcuni dati aggiuntivi, come JavaScript, è possibile passare questi come una stringa nel terzo parametro:

```
$js = 'onClick="some_function()";'
echo form_button('mybutton', 'Click Me', $js);
```

`form_close()` produce un tag di chiusura `</form>`. L'unico vantaggio nell'usare questa funzione è che permette di passarle dei dati che saranno aggiunti dopo il tag. Per esempio:

```
$string = "</div></div>";  
  
echo form_close($string);  
  
// Produce:  
  
</form>  
</div></div>
```

`form_prep()` consente di utilizzare in modo sicuro l'HTML e qualsiasi carattere come le doppie virgolette all'interno degli elementi del form evitando che siano generati degli errori. Si consideri questo esempio:

```
$string = 'Here is a string containing "quoted" text.';  
  
<input type="text" name="myform" value="$string" />
```

Dal momento che la stringa di cui sopra contiene una serie di doppie virgolette, questo causerà l'interruzione del form. La funzione `form_prep` converte l'HTML in modo che questo possa essere utilizzato in modo sicuro:

```
<input type="text" name="myform" value="<?php echo form_prep($string); ?>" />
```

Nota: se si utilizza uno degli Helper dedicati ai form elencati in questa sezione, tutti i valori del form saranno processati automaticamente, quindi non c'è bisogno di utilizzare questa funzione che verrà invocata solo nel caso si stiano creando i propri elementi del form.

`set_value()` permette di impostare il valore di un attributo `input` o `textarea` del form. È necessario specificare il nome del campo tramite il primo parametro della funzione. Il secondo parametro (opzionale) consente di impostare un valore predefinito per il form. Per esempio:

```
<input type="text" name="quantity" value="<?php echo set_value('quantity',  
    '0'); ?>" size="50" />
```

Il form di cui sopra mostrerà o quando verrà caricato per la prima volta.

`set_select()` se si utilizza un menu `<select>`, questa funzione consente di visualizzare gli elementi selezionati del menu. Il primo parametro deve contenere il nome del menu di selezione, il secondo parametro deve contenere il valore di ogni

elemento, e il terzo parametro (opzionale) consente di impostare come predefinito uno degli elementi (si usa il valore booleano TRUE/FALSE). Per esempio:

```
<select name="myselect">
<option value="one" <?php echo set_select('myselect', 'one', TRUE); ?> >One</
  option>
<option value="two" <?php echo set_select('myselect', 'two'); ?> >Two</option>
<option value="three" <?php echo set_select('myselect', 'three'); ?> >Three</
  option>
</select>
```

set\_checkbox() permette di visualizzare un checkbox (casella di controllo) nello stato in cui è stato inviato. Il primo parametro deve contenere il nome della casella di controllo, il secondo parametro invece il suo valore, e il terzo parametro (opzionale) consente di impostare una voce come predefinita (si usa il valore booleano TRUE/FALSE). Per esempio:

```
<input type="checkbox" name="mycheck" value="1" <?php echo set_checkbox('
  mycheck', '1'); ?> />
<input type="checkbox" name="mycheck" value="2" <?php echo set_checkbox('
  mycheck', '2'); ?> />
```

set\_radio() permette di visualizzare i pulsanti radio nello stato in cui sono state inviate. Questa funzione è identica alla funzione set\_checkbox() di qui sopra.

```
<input type="radio" name="myradio" value="1" <?php echo set_radio('myradio', '
  1', TRUE); ?> />
<input type="radio" name="myradio" value="2" <?php echo set_radio('myradio', '
  2'); ?> />
```

## 9.10 HELPER HTML

L'Helper, che ha diverse funzione che aiutano nel lavoro con il codice HTML, viene caricato con:

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

`br()` genera un tag `<br />` (interruzione di linea) basato sul numero inviato. Per esempio

```
echo br(3);
```

Il codice precedente produce tre ritorni a capo: `<br /><br /><br />`

`heading()` produce il tag HTML `<h1>` dove il primo parametro contiene i dati, e il secondo la dimensione dell'head (testata). Per esempio:

```
echo heading('Benvenuto!', 3);
```

Viene generato: `<h3>Benvenuto!</h3>`

Inoltre, tramite un terzo parametro si possono aggiungere ulteriori attributi al tag come per esempio:

```
echo heading('Welcome!', 3, 'class="pink"');
```

Il codice produce: `<h3 class=pink>Benvenuto!</h3>`

`img()` permette di generare i tag `<img />` in cui il primo parametro contiene il percorso dell'immagine. Per esempio:

```
echo img('images/picture.jpg');  
// genera 
```

Vi è un secondo parametro opzionale che può essere TRUE/FALSE. Se viene specificato TRUE nell'attributo src viene inserita anche il segmento specificato da `$config['index_page']` oltre all'indirizzo dell'immagine. Generalmente, questo parametro è consigliato quando si sta utilizzando un media controller.

```
echo img('images/picture.jpg', TRUE);  
// genera 
```



Inoltre, un array associativo può essere passato alla funzione `img()` per il controllo completo su tutti gli attributi e valori. Se non viene fornito un attributo `alt`, CodeIgniter genererà una stringa vuota.

```
$image_properties = array(
    'src' => 'images/picture.jpg',
    'alt' => 'Me, demonstrating how to eat 4 slices of pizza at one time',
    'class' => 'post_images',
    'width' => '200',
    'height' => '200',
    'title' => 'That was quite a night',
    'rel' => 'lightbox',
);

echo img($image_properties);
// 
```

`link_tag()` consente di creare i tag HTML `<link />`. Ciò è utile per i collegamenti ai fogli di stile, così come ad altre risorse. I parametri sono `href` (con `rel` opzionale), `type`, `title`, `media` e `index_page`. Quest'ultimo può assumere un valore `TRUE/FALSE` a seconda che l'attributo `href` debba contenere la pagina specificata da `$config['index_page']` aggiunta all'indirizzo creato.

```
echo link_tag('css/mystyles.css');
//genera <link href="http://site.com/css/mystyles.css" rel="stylesheet" type="
    text/css" />
```

Un altro esempio:

```
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', 'Il mio Feed RSS');
// <link href="http://site.com/feed" rel="alternate" type="application/rss+xml
    " title="Il mio Feed RSS" />
```

Inoltre, un array associativo può essere passato alla funzione `link()` per il controllo completo su tutti gli attributi e valori.

```
$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 un numero di non-breaking space, ovvero spazi indivisibili **&nbsp;**; in base al numero fornito:

```
echo nbs(3);
```

Produce: **&nbsp;&nbsp;&nbsp;**;

`ol()` e `ul()` permettono di generare liste HTML rispettivamente ordinate e non ordinate grazie ad array mono o multidimensionali. Per esempio:

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

$list = array(
    'rosso',
    'blu',
    'verde',
    'giallo'
);

$attributes = array(
    'class' => 'boldlist',
    'id'    => 'mylist'
);

echo ul($list, $attributes);
```

Il codice precedente produce:

```
<ul class="boldlist" id="mylist">
  <li>rosso</li>
  <li>blu</li>
  <li>verde</li>
  <li>giallo</li>
</ul>
```

Qui di seguito un esempio più complesso, utilizzando un array multidimensionale:

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

$attributes = array(
    'class' => 'boldlist',
    'id'    => 'mylist'
);

$list = array(
    'colori' => array(
        'rosso',
        'blue',
        'verde'
    ),
    'forme' => array(
        'rotondo',
        'quadrato',
        'cerchio' => array(
            'ellisse',
            'ovale',
            'sfera'
        )
    ),
    'stati' => array(
        'felice',
        'emozioni' => array(
            'sconfitto' => array(
                'sconsolato',
                'sfiduciato',
                'depresso'
            ),
            'annoiato',
            'irritato',
            'affamato'
        )
    )
);

echo ul($list, $attributes);
```

il codice precedente genera:

```
<ul class="boldlist" id="mylist">
  <li>colori
    <ul>
      <li>rosso</li>
      <li>blue</li>
      <li>verde</li>
    </ul>
  </li>
  <li>forme
    <ul>
      <li>rotondo</li>
      <li>quadrato</li>
      <li>circolo</li>
      <ul>
        <li>ellisse</li>
        <li>ovale</li>
        <li>sfera</li>
      </ul>
    </ul>
  </li>
  <li>stati
    <ul>
      <li>felice</li>
      <li>emozioni
        <ul>
          <li>sconfitto
            <ul>
              <li>sconsolato</li>
              <li>sfiduciato</li>
              <li>depresso</li>
            </ul>
          </li>
          <li>annoiato</li>
          <li>irritato</li>
          <li>affamato</li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

meta() aiuta a generare i meta tag. É possibile passare alla funzione le stringhe, oppure array semplici o multidimensionali. Esempio:

```

echo meta('descrizione', 'Il mio grande sito');
// Genera: <meta name="descrizione" content="Il mio grande sito" />

echo meta('Content-type', 'text/html; charset=utf-8', 'equiv'); // Nota il
    terzo parametro: òpu assumere il valore di "equiv" oppure "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' => 'Il mio grande sito'),
    array('name' => 'keywords', 'content' => 'amore, passione, intrigo,
    inganno'),
    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="Il mio grande sito" />
// <meta name="keywords" content="amore, passione, intrigo, inganno" />
// <meta name="robots" content="no-cache" />
// <meta http-equiv="Content-type" content="text/html; charset=utf-8" />

```

doctype() aiuta a generare dichiarazioni specificando il tipo di documento, o DTD. Per impostazione predefinita viene utilizzato “XHTML 1.0”, ma sono disponibili molti altri doctype.

```

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">

```

Di seguito è riportato un elenco di scelte DOCTYPE. Queste sono configurabili nel file `/application/config/doctypes.php`.

Doctype	Opzioni	Risultato
XHTML 1.1	doctype('xhtml11')	<!DOCTYPE html PUBLIC - //W3C//DTD XHTML 1.1//EN http://www.w3.org/TR/xhtml11/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 Frameset	doctype('xhtml1-frame')	<!DOCTYPE html PUBLIC -//W3C//DTD XHTML 1.0 Fra- meset//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»

## 9.11 HELPER INFLECTOR

L'Helper Inflector contiene diverse funzioni che permettono di cambiare le parole nel loro plurale, singolare, con notazione camel case, ecc. Il suo caricamento avviene come di consueto con l'istruzione:

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

Sono quindi disponibili le seguenti funzioni:

`singular()` modifica il plurale delle parole, trasformandola nella forma singolare. Per esempio:

```
$word = "dogs";  
echo singular($word); // Restituisce "dog"
```

`plural()` modifica il singolare di una parola, trasformandola nel plurale. Per esempio:

```
$word = "dog";  
echo plural($word); // Restituisce "dogs"
```

Per forzare il plurale di una parola che termina con `end` si utilizzi un secondo argomento impostato sul valore booleano `TRUE`:

```
$word = "pass";  
echo plural($word, TRUE); // Restituisce "passes"
```

`camelize` modifica una stringa di parole separate da spazi o underscore in notazione camel case. Per esempio:

```
$word = "my_dog_spot";  
echo camelize($word); // Restituisce "myDogSpot"
```

`underscore()` trasforma più parole separate da spazi trasformando questi ultimi in underscore:

```
$word = "my_dog_spot";  
echo humanize($word); // Restituisce "My Dog Spot"
```

`humanize()` prende più parole separate dal carattere underscore e li sostituisce con degli spazi singoli. Inoltre ogni parola verrà formattata con l'iniziale maiuscola. Per esempio:

```
$word = "my_dog_spot";  
echo humanize($word); // Restituisce "My Dog Spot"
```



## 9.12 HELPER LANGUAGE

L'Helper viene caricato con l'istruzione:

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

La funzione disponibile con questo Helper è la seguente:

`lang('language line', 'element id')` restituisce una riga di testo caricata da un file di lingua con sintassi semplificata. Questo metodo può essere per le Viste può essere preferibile alla funzione `$this->lang->line()`. Il secondo parametro opzionale sarà anche l'output del label del form. Per esempio:

```
echo lang('language_key', 'form_item_id');  
// diventa <label for="form_item_id">language_key</label>
```

### 9.13 HELPER NUMBER

Questo Helper permette di lavorare più facilmente con dati numerici. Il suo caricamento avviene con l'istruzione:

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

La funzione disponibile con questo Helper è la seguente:

`byte_format()` trasforma un numero nel formato byte, basato sulla dimensione e aggiunge il suffisso appropriato. Per esempio:

```
echo byte_format(456); // Restituisce 456 Bytes
echo byte_format(4567); // Restituisce 4.5 KB
echo byte_format(45678); // Restituisce 44.6 KB
echo byte_format(456789); // Restituisce 447.8 KB
echo byte_format(3456789); // Restituisce 3.3 MB
echo byte_format(12345678912345); // Restituisce 1.8 GB
echo byte_format(123456789123456789); // Restituisce 11,228.3 TB
```

Un secondo parametro permette di impostare la precisione del risultato, ovvero le cifre decimali solitamente omesse.

```
echo byte_format(45678, 2); // Restituisce 44.61 KB
```

## 9.14 HELPER PATH

È l'Helper che permette di lavorare con i percorsi dei file sul server. Il caricamento dell'helper avviene con l'istruzione:

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

La funzione disponibile è `set_realpath()` che controlla se il percorso esiste. Questa funzione restituisce il percorso server senza link simbolici o directory relative. Un secondo argomento opzionale causerà un errore innescato (trigger) se il percorso non può essere risolto,

```
$directory = '/etc/passwd';  
echo set_realpath($directory);  
// restituisce "/etc/passwd"  
  
$non_existent_directory = '/path/to/nowhere';  
echo set_realpath($non_existent_directory, TRUE);  
// restituisce an error, as the path could not be resolved  
  
echo set_realpath($non_existent_directory, FALSE);  
// restituisce "/path/to/nowhere"
```

## 9.15 HELPER SECURITY

L'Helper viene caricato con l'istruzione:

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

Qui di seguito le funzioni disponibili:

`xss_clean()` fornisce un filtro Cross Site Script Hack. Questa funzione è un alias della funzione definita nella classe Input (si veda la sezione [8.15 a pagina 291](#)).

`sanitize_filename()` fornisce una protezione contro il directory traversal. Questa funzione è un alias della funzione definita nella classe Security (si veda la sezione [8.22 a pagina 320](#)).

`do_hash()` permette di creare un hash SHA1 oppure MD5 sfruttabile per il criptaggio delle password. Per impostazione predefinita viene utilizzato l'hash SHA1:

```
$str = do_hash($str); // SHA1  
  
$str = do_hash($str, 'md5'); // MD5
```

Nota: questa funzione era precedentemente chiamata `dohash()` ed è stata deprecata a favore dell'attuale `do_hash()`.

`strip_image_tags()` questa funzione di sicurezza priverà una stringa dai tag immagine. In questo modo il risultato sarà un semplice testo con l'URL dell'immagine.

```
$string = strip_image_tags($string);
```

`encode_php_tags()` è una funzione di sicurezza che converte i tag PHP in entità. Questa funzione viene utilizzata automaticamente se si usa la funzione che permette il filtro XSS:

```
$string = encode_php_tags($string);
```

## 9.16 HELPER SMILEY

Questo Helper permette di trasformare una serie definita di caratteri testuali in una immagine che rappresenta uno smiley. Inoltre, consente di visualizzare una serie di immagini smiley che quando vengono cliccate saranno inserite in un campo form. Ad esempio, se avete un blog che permette all'utente di commentare è possibile mostrare le faccine accanto per commentare il form. Gli utenti possono fare scegliere lo smiley desiderato che, con l'aiuto di alcuni JavaScript, verrà posizionato nel campo form.

Ecco un esempio che dimostra come è possibile creare un set di emoticon cliccabili accanto a un campo di un form. Questo esempio richiede che prima si scarichi e si installi le immagini degli smiley, quindi è necessario creare un Controller e una Vista come descritto.

Importante: prima di iniziare, si scarichino le immagini smiley che andranno inserite in un luogo accessibile (si vedano i permessi) sul server. Questo helper presuppone inoltre che l'array di sostituzione smiley si trovi nel file `/application/config/smileys.php`.

### IL CONTROLLER

Si crei il file chiamato **smileys.php** nella directory `/application/controllers/` e si inserisca il codice descritto in seguito. Si modifichi nella funzione `get_clickable_smileys()` l'[URL](#) in modo che punti alla directory degli smiley. L'esempio utilizza anche la classe Table:

```

<?php

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);
    }
}
?>

```

Nota: l'istruzione `$image_array = get_clickable_smileys('http://example.com/images/smiley` indica il percorso in cui si trovano le immagini degli smile.

Ora si definisca una Vista denominata `smiley_view.php` nella directory `/application/views/` e si inserisca il codice:

```

<html>
<head>
<title>Smiley</title>

<?php echo smiley_js(); ?>

</head>
<body>

<form name="blog">
<textarea name="comments" id="comments" cols="40" rows="4"></textarea>
</form>

<p>Scegli il tuo smile</p>

<?php echo $smiley_table; ?>

```

```
</body>
</html>
```

Ora sarà possibile vedere quanto prodotto, collegandosi all'URL: <http://www.example.com/index.php/smileys/>

Nota: se non compaiono le immagini, è semplicemente perché queste non sono presenti nel server. CodeIgniter permette di utilizzare gli smile che si desidera senza alcuna limitazione.

## CAMPI ALIAS

Quando si effettuano modifiche ad una Vista può essere scomodo avere il campo id nel controller. Per ovviare a questo, è possibile fornire il link degli smiley e un nome generico nome che sarà legato a un ID specifico nella Vista.

```
$image_array = get_smiley_links("http://example.com/images/smileys/", "
    comment_textarea_alias");
```

Per mappare l'alias ad un campo id, entrambi devono essere passati nella funzione smiley\_js:

```
$image_array = smiley_js("comment_textarea_alias", "comments");
```

get\_clickable\_smileys() restituisce un array contenente le immagini smiley insieme al relativo link cliccabile. È necessario fornire l'URL alla cartella smiley e un campo ID oppure un campo alias.

```
$image_array = get_smiley_links("http://example.com/images/smileys/", "comment
    ");
```

L'uso di questa funzione senza il secondo parametro, in combinazione con js\_insert\_smiley è ora deprecato.

smiley\_js() genera il codice JavaScript che permette alle immagini di essere cliccate ed inserite in un campo del form. Se si è fornito un alias invece di un id quando si generano i link degli smiley, è necessario passare l'alias e il corrispondente campo id alla funzione. Questa funzione è progettata per essere collocata nell'area <head> della propria pagina web.

```
<?php echo smiley_js(); ?>
```

Questa funzione sostituisce js\_insert\_smiley che è stata deprecata.

`parse_smileys()` prende una stringa di testo come input e sostituisce tutto il testo che codifica gli smile nell'equivalente immagine. Il primo parametro deve contenere la propria stringa, mentre il secondo deve contenere l'URL che punta alla cartella smiley:

```
$str = 'Here are some simileys: :-) ;-)'; $str = parse_smileys($str, "http://  
example.com/images/smileys/"); echo $str;
```



## 9.17 HELPER STRING

Questo Helper assiste nel lavoro con le stringhe. Il suo caricamento avviene con l'istruzione:

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

- `random_string()` genera una stringa casuale basata sul tipo e sulla lunghezza specificata. Questa funzione è utile per creare password o generare hash random. Il primo parametro specifica il tipo di stringa, il secondo parametro specifica la lunghezza. Le scelte elencate in seguito sono disponibili come parametri: `alpha`, `alnum`, `numeric`, `nozero`, `unique`, `md5`, `encrypt` e `sha1`.
  - `alpha`: una stringa con solo lettere minuscole e maiuscole
  - `alnum`: una stringa alfanumerica con caratteri minuscole e maiuscole
  - `numeric`: stringa numerica
  - `nozero`: stringa numerica senza zeri
  - `unique`: stringa crittografata con MD5 e `uniqid()`. Nota: il parametro `length` che specifica la lunghezza non è disponibile per questo parametro, poiché viene restituita sempre una stringa di lunghezza fissa (32 caratteri)
  - `sha1`: stringa numerica casuale crittografata con la funzione `do_hash()`. Per maggiori informazioni si veda la sezione [9.15 a pagina 410](#)

Esempio di utilizzo:

```
echo random_string('alnum', 16);
```

- `increment_string()` incrementa una stringa aggiungendo nella parte finale un numero o incrementando il numero della stringa. Questa funzione è utile per creare copie di un file o per duplicare il contenuto del database con un titolo o un marcatore univoco.

Per esempio:

```
echo increment_string('file', '_'); // "file_1"  
echo increment_string('file', '-', 2); // "file-2"  
echo increment_string('file-4'); // "file-5"
```

- `alternator()` permette di alternare due o più elementi in un ciclo. Per esempio:

```
for ($i = 0; $i < 10; $i++)  
{  
    echo alternator('string one', 'string two');  
}
```

Si possono aggiungere quanti parametri si desidera e ad ogni iterazione del ciclo verrà restituito un nuovo elemento:

```
for ($i = 0; $i < 10; $i++)  
{  
    echo alternator('one', 'two', 'three', 'four', 'five');  
}
```

Per utilizzare più chiamate a se stanti a questa funzione, basta invocarla senza argomenti per reinizializzarla.

- `repeater()` genera copie ripetute dei dati forniti alla funzione. L'esempio seguente genera 30 ritorni a capo:

```
$string = "\n";  
echo repeater($string, 30);
```

- `reduce_double_slashes()` converte gli slash doppi in uno singolo, eccetto quello definito in **http://**. Per esempio:

```
$string = "http://example.com//index.php";  
echo reduce_double_slashes($string); // results in "http://example.com/  
index.php"
```

- `trim_slashes()` rimuove ogni slash all'inizio o alla fine di una stringa. Per esempio:

```
$string = "/this/that/theother/";  
echo trim_slashes($string); // results in this/that/theother
```

- `reduce_multiples()` consente di ridurre la ripetizione di un determinato carattere in una stringa subito dopo un altro carattere specificato. Per esempio:

```
$string="Fred, Bill,, Joe, Jimmy";  
$string=reduce_multiples($string,","); //results in "Fred, Bill, Joe,  
Jimmy"
```

La funzione accetta i seguenti parametri:

```
reduce_multiples(string: text to search in, string: character to reduce,  
boolean: whether to remove the character from the front and end of  
the string)
```

Il primo parametro contiene la stringa che si vuole esaminare, il secondo parametro contiene il carattere che si vuole ridurre, mentre il terzo parametro assume il valore FALSE di default. Se questo viene impostato su TRUE rimuoverà le occorrenze del carattere all'inizio e alla fine della stringa. Esempio:

```
$string=",Fred, Bill,, Joe, Jimmy,";  
$string=reduce_multiples($string, ",", TRUE); //results in "Fred, Bill,  
Joe, Jimmy"
```

- `quotes_to_entities()` converte le singole e le doppie virgolette in una stringa nelle corrispondenze entità HTML. Per esempio:

```
$string="Joe's \"dinner\"";  
$string=quotes_to_entities($string); //results in "Joe&#39;s &quot;dinner  
&quot;"
```

- `strip_quotes()` rimuove le virgolette singole e doppie da una stringa. Per esempio:

```
$string="Joe's \"dinner\"";  
$string=strip_quotes($string); //results in "Joes dinner"
```

## 9.18 HELPER TEXT

L'Helper che aiuta nella gestione del testo, viene caricato con la funzione:

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

- `word_limiter()` effettua il troncamento di una stringa dopo il numero di parole specificato. Per esempio:

```
$string = "Una piccola stringa composta da sole otto parole";  
  
$string = word_limiter($string, 4);  
  
// Restituisce: Una piccola stringa composta...
```

Il terzo parametro è un suffisso opzionale che viene aggiunto alla stringa. Per impostazione predefinita vengono aggiunti i puntini di sospensione.

- `character_limiter()` effettua il troncamento di una stringa dopo un numero di caratteri specificato. Per mantenere l'integrità delle parole il conteggio dei caratteri potrebbe essere leggermente differente da quello specifico. Per esempio:

```
$string = "Una piccola stringa composta da sole otto parole";  
  
$string = character_limiter($string, 20);  
  
// Restituisce: Una piccola stringa...
```

Il terzo parametro è un suffisso opzionale che viene aggiunto alla stringa. Se non dichiarato vengono aggiunti i puntini di sospensione.

- `ascii_to_entities()` converte i valori ASCII nei corrispondenti caratteri, includendo gli ASCII alti e i caratteri MS Word che possono causare dei problemi quando utilizzati in una pagina web. In questo modo i caratteri possono essere visualizzati in modo coerente indipendentemente dalle impostazioni del browser e immagazzinati in modo affidabile in un database. Questa funzione mostra una certa dipendenza dal set di caratteri supportati dal server, e per questo può non essere sempre affidabile. Nella maggior parte questa funzione identifica correttamente i caratteri escludendo caratteri non convenzionali (come i caratteri accentati). Per esempio:

```
$string = ascii_to_entities($string);
```

- `entities_to_ascii()` svolge il compito opposto a quello di `ascii_to_entities()` ■ restituisce i caratteri codificati in ASCII.
- `convert_accented_characters()` traslittera i caratteri ASCII alti in quelli equivalenti ASCII bassi; questa funzione è utile quando è necessario utilizzare i caratteri non-English dove solo i caratteri ASCII standard sono utilizzati in modo sicuro, per esempio, negli [URL](#).

```
$string = convert_accented_characters($string);
```

Questa funzione utilizza un file di configurazione `/application/config/foreign_chars.php` ■ per definire un array di provenienza e di destinazione per la traslitterazione.

- `word_censor()` consente di censurare le parole all'interno di una stringa di testo. Il primo parametro conterrà la stringa originale. Il secondo conterrà una serie di parole che si vogliono disabilitare. Il terzo parametro (opzionale) può contenere un valore per sostituire le parole. Se non specificato, le parole censurate vengono sostituite con il simbolo di cancelletto: #####. Esempio:

```
$disallowed = array('darn', 'shucks', 'golly', 'phooey');  
  
$string = word_censor($string, $disallowed, 'Beep!');
```

- `highlight_code()` colora una stringa di codici (PHP, HTML, ecc). Per esempio:

```
$string = highlight_code($string);
```

Viene usata la funzione PHP `highlight_string()` quindi vengono utilizzati i colori usati nel file **php.ini**.

- `highlight_phrase()` mette in evidenza una frase all'interno di una stringa di testo. Il primo parametro conterrà la stringa originale, il secondo conterrà la frase che si desidera evidenziare. Il terzo e il quarto parametro conterrà i tag [HTML](#) di apertura/chiusura con cui si desidera inglobare la frase evidenziata. Esempio:

```
$string = "Here is a nice text string about nothing in particular.";

$string = highlight_phrase($string, "nice text", '<span style="color
:#990000">', '</span>');
```

- `word_wrap()` spezza un testo su una nuova linea dopo un determinato numero di caratteri. Per esempio:

```
$string = "Here is a simple string of text that will help us demonstrate
this function.";

echo word_wrap($string, 25);

// Produce:

Here is a simple string
of text that will help
us demonstrate this
function
```

- `ellipsize()` questa funzione rimuove i tag da una stringa, la divide dopo una lunghezza massima definita, e inserisce i puntini di sospensione.

Il primo parametro è la stringa, il secondo è il numero di caratteri nella stringa finale. Il terzo parametro è dove nella stringa i puntini di sospensione dovrebbe apparire (il valore va da 0 a 1, dove 0 è a sinistra, 1 a destra, .5 è al centro).

Un quarto parametro facoltativo è il tipo di puntini di sospensione. Per impostazione predefinita vengono inseriti i caratteri ...

```
$str = 'this_string_is_entirely_too_long_and_might_break_my_design.jpg';

echo ellipsize($str, 32, .5);
```

La funzione produce:

```
...
this_string_is_eak_my_design.jpg
```

## 9.19 HELPER TYPOGRAPHY

L'Helper viene caricato con il seguente codice:

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

- `auto_typography()` formatta il testo in modo che sia semanticamente e tipograficamente in [HTML](#). Si veda la classe `Typography` [8.26 a pagina 339](#) per maggiori informazioni. Per esempio:

```
$string = auto_typography($string);
```

L'uso di questa funzione può essere fonte di un intenso lavoro da parte del server, soprattutto se la quantità di dati da formattare è elevata. Se si utilizza questa funzione, è consigliabile effettuare il caching delle pagine (si veda la sezione [6.5 a pagina 94](#)).

- `nl2br_except_pre()` converte i ritorni a capo nel tag `<br />` a meno che questi non siano all'interno dei tag `<pre>`. Questa funzione è simile a quella nativa PHP `nl2br()` tranne per il fatto che questa ignora i tag `<pre>`.

```
$string = nl2br_except_pre($string);
```

## 9.20 HELPER URL

L'Helper in questione fornisce molte funzionalità che assistono nel lavoro con gli URL. Esso viene caricato con il codice:

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

`site_url()` restituisce l'URL del sito, come specificato nel file di configurazione. Il file **index.php** (o qualsiasi altro elemento sia stato impostato come `index_page` nel file di configurazione) verrà aggiunto all'URL, così come qualsiasi altro segmento URI e `url_suffix` impostati nel file di configurazione. È consigliato utilizzare questa funzione ogni volta che è necessario generare un URL locale in modo da migliorare la portabilità delle pagine in caso si modifichino gli URL in futuro.

I segmenti possono essere eventualmente passati alla funzione come una stringa o un array. Ecco un esempio che utilizza una stringa:

```
echo site_url("news/local/123");
```

L'esempio precedente restituirà: `http://example.com/index.php/news/local/123`  
Invece se si passa alla stringa un array:

```
$segments = array('news', 'local', '123');  
echo site_url($segments);
```

`base_url()` restituisce l'URL di base del proprio sito. Questa funzione fornisce lo stesso dato della funzione `site_url()`, evitando che siano aggiunti `index_page` o `url_suffix`. È anche possibile fornire un segmento come una stringa o un array. Ecco un esempio con una stringa:

```
echo base_url("blog/post/123");
```

Il codice precedente restituirà: `http://example.com/blog/post/123`

Questa funzione si rivela molto utile perché a differenza di `site_url()`, è possibile fornire una stringa ad un file, come ad esempio un'immagine o un foglio di stile. Per esempio:

```
echo base_url("/_user_guide_src_ci/images/icons/edit.png");
```

Il codice restituirà: `http://example.com/images/icons/edit.png`

`current_url()` restituisce un URL completo (compresi i segmenti) della pagina che si sta attualmente visualizzando.



`uri_string()` restituisce i segmenti URI di ogni pagina che contiene questa funzione. Per esempio, se il proprio URL è il seguente:

```
http://some-site.com/blog/comments/123
```

```
/blog/comments/123
```

`index_page()` restituisce la pagina index, come specificato nel file di configurazione del proprio sito:

```
echo index_page();
```

`anchor()` crea un link anchor in formato HTML basato sull'URL locale del sito:

```
<a href="http://example.com">Clicca qui</a>
```

Il tag ha tre parametri opzionali:

```
anchor(uri segments, text, attributes)
```

Il primo parametro può contenere ogni segmento (stringa o array) che si vuole aggiungere all'URL.

Nota: se si sta definendo un link interno alla propria applicazione, non si deve includere l'URL di base (`http://[...]`) poiché questo sarà aggiunto automaticamente in base alle informazioni specificate nel file di configurazione. Si includano pertanto i soli segmenti URI che si desidera aggiungere all'URL.

Il secondo segmento è il testo che si desidera linkare: se si lascia vuoto, verrà utilizzato l'URL. Il terzo parametro, infine, può contenere un elenco di attributi (di tipo stringa o un array associativo) da aggiungere al link. Ecco alcuni esempi:

```
echo anchor('news/local/123', 'My News', 'title="News title"');
```

Il codice produrrà: `<a href=http://example.com/index.php/news/local/123 title=News title>My News</a>`

```
echo anchor('news/local/123', 'My News', array('title' => 'The best news!'));
```

Il codice produrrà: `<a href=http://example.com/index.php/news/local/123 title=The best news!>My News</a>`

`anchor_popup()` è molto simile alla funzione `anchor()`, ma si differenzia da questa per il fatto che l'URL viene aperto in una nuova finestra. È possibile specificare gli attributi JavaScript nel terzo parametro per controllare come la finestra dovrà essere aperta. Se questo parametro non è impostato, verrà aperta una nuova finestra semplicemente in base alle impostazioni del browser. Qui di seguito viene presentato un esempio con i seguenti attributi:

```
$atts = array(  
    'width'      => '800',  
    'height'     => '600',  
    'scrollbars' => 'yes',  
    'status'     => 'yes',  
    'resizable'  => 'yes',  
    'screenx'    => '0',  
    'screeny'    => '0'  
);  
  
echo anchor_popup('news/local/123', 'Click Me!', $atts);
```

Nota: gli attributi di cui sopra sono sempre definiti nelle impostazioni predefinite: è necessario definirli con un valore solo se si vuole che assumano un valore differente. Se si desidera che la funzione utilizzi tutti i suoi parametri di default, semplicemente le si passi un array vuoto nel terzo parametro:

```
echo anchor_popup('news/local/123', 'Click Me!', array());
```

`mailto()` crea un link email in formato standard HTML. Per esempio:

```
echo mailto('me@my-site.com', 'Clicca qui per contattarmi');
```

Come con la funzione `anchor()` è possibile impostare gli attributi con il terzo parametro.

`safe_mailto()` è identica alla funzione precedente tranne per il fatto che scrive una versione offuscata del tag `mailto` usando numeri ordinali scritti con JavaScript per nascondere gli indirizzi in chiaro agli spam bot.

`auto_link()` trasforma automaticamente gli URL e gli indirizzi email contenuti in una stringa in link. Per esempio:

```
$string = auto_link($string);
```

Il secondo parametro determina se vengono convertiti sia gli URL che le email o solo uno di essi: il comportamento predefinito, se non si imposta alcun parametro, prevede la conversione di ambedue. I collegamenti di posta elettronica sono codificati come `safe_mailto()`, come mostrato sopra.

Conversione dei soli URL:

```
$string = auto_link($string, 'url');
```

Conversione dei soli indirizzi email:

```
$string = auto_link($string, 'email');
```

Il terzo parametro determina se i link saranno visualizzati nella nuova finestra. I valori possono essere TRUE/FALSE.

```
$string = auto_link($string, 'both', TRUE);
```

`url_title()` prende una stringa in ingresso e crea una stringa URL human-friendly (comprensibile). Questo è utile se, ad esempio, si ha un blog in cui si vuole usare il titolo delle voci nell'URL. Per esempio:

```
$title = "Cosa c'è di sbagliato con i CSS";  
$url_title = url_title($title);  
  
// Produce: Cosa-c'è-di-sbagliato-con-i-CSS
```

Il secondo parametro determina la parola da delimitare. Per impostazione predefinita sono usati i caratteri dash.

```
$title = "Cosa c'è di sbagliato con i CSS?";  
$url_title = url_title($title, '_');  
  
// Produces: Cosa-c'è-di-sbagliato-con-i-CSS
```

Il terzo parametro (che può assumere i valori TRUE/FALSE) determina se forzare o meno la trasformazione dei caratteri in minuscolo: per impostazione predefinita non viene effettuata alcuna trasformazione.

```
$title = "Cosa c'è di sbagliato con i CSS?";  
$url_title = url_title($title, '_', TRUE);  
  
// Produce: cosa-c'è-di-sbagliato-con-i-css
```

`prep_url()` questa funzione aggiunge **http://** alla stringa passata. Per esempio:

```
$url = "example.com";  
  
$url = prep_url($url);
```

`redirect()` effettua un header redirect per l'URI specificato. Se si fornisce l'URL completo del sito, verrà creato il relativo link. Diverso è il discorso per i link locali al proprio progetto: in questo caso si devono semplicemente fornire i segmenti URI al controller che si vuole dirigere verso il collegamento creato. La funzione costruirà l'URL in base ai vostri valori impostati nel file di configurazione.

Il secondo parametro opzionale consente di scegliere tra il metodo location (predefinito) o il metodo refresh. Location è più veloce, ma sui server Windows a volte può causare qualche problema. Il terzo parametro opzionale consente di inviare uno specifico codice di risposta HTTP (HTTP Response Code) questo potrebbe essere utilizzato per esempio per creare un redirect 301 per i motori di ricerca. Il Response Code predefinito è 302. Il terzo parametro è disponibile solo con i redirect 'location', e non per quelli 'refresh'. Per esempio:

```
if ($logged_in == FALSE)  
{  
    redirect('/login/form/', 'refresh');  
}  
  
// con redirect 301  
redirect('/article/13', 'location', 301);
```

Nota: questa funzione deve essere utilizzata prima che qualsiasi output sia inviato al browser poiché utilizza i server header (intestazioni di server). Nota: Per un maggior controllo sugli header, è necessario utilizzare la funzione `set_header()` della libreria Output.

## 9.21 HELPER XML

L'Helper viene caricato con l'istruzione:

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

- `xml_convert('string')` prende come input una stringa e converte i caratteri XML elencati qui di seguito in entità:

- e commerciale: `&`
- i caratteri maggiore e minore: `<` `>`
- apici singoli e doppi: `'` `"`
- lineetta: `-`

Questa funzione ignora la e commerciale se essa è parte di una entità carattere esistente. Per esempio:

```
$string = xml_convert($string);
```

## 9.22 ALTERNATIVE PHP CACHE

### COME FUNZIONA APC

Uno degli scopi prefissati dal linguaggio PHP e da chi ovviamente lo utilizza nei propri progetti, è quello di ottimizzare le richieste fatte al server, in modo da migliorare le prestazioni dello stesso verso i terminali host, ovvero gli utenti finali.

Il codice di scripting PHP ogni volta che viene richiamato, non viene immediatamente eseguito, ma inizialmente deve essere analizzato e precompilato in un codice intermedio, quindi ottimizzato e alla fine eseguito. Tutti questi passi, risultano ridondanti quando i contenuti della pagina sono immutati (pagine statiche) e risultano penalizzanti se il server deve soddisfare le richieste di innumerevoli client.

Computo degli “acceleratori” come APC è appunto quello di velocizzare i passaggi che precedono l’esecuzione ultima del codice PHP, per garantire al server di preservare parte della potenza computazionale. Esaminiamo i passaggi a cui il codice di scripting PHP è sottoposto, partendo dalla sua lettura dal filesystem ed il suo spostamento in memoria.

1. Lexing: L’interno del codice PHP viene convertito in token o lexicon
2. Parsing: Durante questa fase, i token vengono elaborati per ricavare delle espressioni significative
3. Compilazione: Le espressioni derivate vengono compilate in opcode
4. Esecuzione: L’opcode viene eseguito per ottenere il risultato finale

L’obiettivo di APC è di saltare i passaggi da 1 a 4, sfruttando un segmento di memoria condivisa che faccia da cache: in questa porzione di memoria i codici operativi generati vengono salvati e quando necessari, copiati nel processo di esecuzione per essere eseguiti.

APC è in definitiva un’estensione nativa per PHP che svolge il compito di pre-compilare, ottimizzare e mantenere in memoria il codice intermedio associato agli script PHP dopo la prima richiesta effettuata ad un file PHP.

### INSTALLAZIONE CON WINDOWS

L’installazione, per chi ha già utilizzato librerie esterne a quelle native del PHP non comporterà alcun problema: avviene semplicemente copiando la DLL precompilata nella directory che contiene le altre estensioni. Infine si abilita l’estensione utilizzando la seguente istruzione nel file **php.ini**:

```
extension=php_apc.dll
```

## INSTALLAZIONE CON LINUX

La libreria in questione, sotto sistemi Unix o simili, può avvenire compilando manualmente la libreria con il comando **phpsize**. Ecco i passi da compiere:

- scaricare i sorgenti della libreria dal sito di PECL <http://pecl.php.net/package/apc>
- decomprimere i file scaricati in una directory del server utilizzando i permessi di root

- a questo sarà necessario compilare la libreria.

```
phpize
./configure --enable-apc --enable-mmap
make
cp modules/apc.so /usr/local/lib/php
```

- riavviare il server web con l'istruzione:

```
service mysqld start
service httpd start
```

## CONFIGURAZIONE DI APC

- `apc.enabled` abilita/disabilita la libreria APC. Il valore 0 disabilita la libreria.
- `apc.optimization` specifica il livello di ottimizzazione da applicare al bytecode intermedio generato dallo Zend Engine
- `apc.ttl` specifica per quanti secondi il codice intermedio rimarrà memorizzato nella cache. Un valore uguale a 0 lascerà in memoria il codice per un tempo molto lungo
- `apc.filters` è possibile impostare dei filtri per indicare quali file salvare in cache e invece quali tralasciare. I filtri sono definiti in una lista contenente particolari espressioni regolari separate da virgole
- `apc.file_update_protection` alcuni comandi utilizzati per sostituire i file sul server non eseguono operazioni atomiche (non caricano in un file temporaneo e poi sostituiscono quello precedente): in questo APC può trovarsi ad accedere a file incompleti. Per evitare questo inconveniente si indica il ritardo in secondi dopo i quali si procede con l'operazione di compilazione. Il valore standard è 2, che dovrebbe andar bene quasi sempre, salvo casi in cui ci si trovi ad aver a che fare con sorgenti molto lunghi o server particolarmente lenti.

## APC AL LAVORO

Ecco i passaggi dettagliati di ciò che fa realmente APC

- Durante l'inizializzazione del modulo (MINIT), APC utilizza mmap per mappare file in memoria condivisa
- APC durante la richiesta, rimpiazza `zend_compile_file()` con i propri `my_compile_file()`

Particolarmente importante è la funzione svolta da `my_compile_file`:

- Se APC non riesce a trovare il file nella cache, APC chiamerà `zend_compile_file` per compilare il file
- Prende l'opcode che viene restituito e lo pone nella memoria condivisa che viene letta/scritta dai vari processi child di Apache
- Conserva nella memoria condivisa anche classi e function tables
- Copia l'opcode nella memoria dei child di Apache in modo che Zend possa eseguirlo nella fase di esecuzione



## ACRONIMI

### **MYSQL** Oracle MySQL

è uno dei più utilizzati e conosciuti Relational database management system (RDBMS) composto da un client a riga di comando e un server

### **MVC** Model-View-Controll

è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business

### **HTML** Hyper Text Mark-up Language

è il linguaggio di markup solitamente usato per la formattazione di documenti ipertestuali disponibili nel World Wide Web

### **URL** Uniform Resource Locator

è una sequenza di caratteri che identifica univocamente l'indirizzo di una risorsa in Internet, tipicamente presente su un host server

### **URI** Uniform Resource Identifier

è una stringa che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, un indirizzo di posta elettronica, ecc.

### **ORM** Object-relational mapping

è una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS

### **PHP** PHP: Hypertext Preprocessor

linguaggio di programmazione interpretato, originariamente concepito per lo sviluppo di pagine web dinamiche

### **XML-RPC** eXtensible Markup Language-Remote Procedure Call

è un protocollo utilizzato in informatica che permette di eseguire delle chiamate a procedure remote (RPC) attraverso la rete Internet. Questo protocollo utilizza lo standard XML per codificare la richiesta che viene trasportata mediante il protocollo HTTP. Nonostante la sua semplicità permette di trasmettere strutture dati complesse, chiederne l'esecuzione ed avere indietro il risultato.

### **PEAR** PHP Extension and Application Repository

è un framework e un sistema di distribuzione per codice scritto in PHP

### **XSS** Cross-site scripting

è una vulnerabilità che affligge siti web dinamici che impiegano un insufficiente controllo dell'input nei form

**SMTP** Simple Mail Transfer Protocol

è il protocollo standard per la trasmissione via internet di e-mail

**FTP** Simple Mail Transfer Protocol

è il protocollo standard per la trasmissione via internet di e-mail

**GD** GD Graphics Library

è una libreria scritta da Thomas Boutell e altri per la manipolazione dinamica di immagini

**HTTP** Hypertext Transfer Protocol

è un protocollo usato come principale sistema per la trasmissione d'informazioni sul web ovvero in un'architettura tipica client-server

**OOB** Object Oriented Programming

paradigma di programmazione che permette di definire oggetti software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi

**CAPTCHA** Completely Automated Public Turing test to tell Computers and Humans Apart

è un test composto da una o più domande e risposte per determinare se l'utente sia un umano (e non un computer o, più precisamente, un bot).

**RSS** Really Simple Syndication

è un formato per la distribuzione di contenuti Web basato su XML.

**MAMP** MacOS-Apache-MySQL,P (per PHP, PERL, Python)

set di programmi liberi comunemente utilizzati per realizzare un sito web locale sul sistema operativo Mac OS X della Apple

**LAMP** Linux/GNU-Apache-MySQL,P (per PHP, PERL, Python)

set di programmi liberi comunemente utilizzati per realizzare un sito web locale sul sistema operativo Linux/GNU

**WAMP** Windows-Apache-MySQL,P (per PHP, PERL, Python)

set di programmi liberi comunemente utilizzati per realizzare un sito web locale sul sistema operativo Windows di Microsoft

**SU** Super User

definizione o comando impartito da terminale con cui si acquistano i poteri dell'amministratore, con cui compiere azioni altrimenti precluse agli altri utenti (installazione/rimozione del software, ecc)

**CSS** Cascading Style Sheets

è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML

**FTP** File Transport Protocol

Il protocollo di trasferimento file, in informatica e nelle telecomunicazioni, è un protocollo per la trasmissione di dati tra host basato su TCP.

**ASCII** American Standard Code for Information Interchange

É un sistema di codifica dei caratteri a 7 bit, comunemente utilizzato nei calcolatori, proposto dall'ingegnere dell'IBM Bob Berner nel 1961, e successivamente accettato come standard dall'ISO (ISO 646)

**APC** Alternative PHP Cache

É una cache opcode libera ed open source per PHP. Il suo obiettivo è quello di fornire un framework aperto e robusto per il caching e ottimizzare il codice intermedio di PHP.

**cc** Carbon Copy

É una modalità di creazione simultanea di più copie di un documento che prevede l'aggiunta al documento dall'abbreviazione cc seguita da un elenco di destinatari, detti appunto destinatari in copia conoscenza.

**bcc** blind carbon copy

Definita anche come copia conoscenza nascosta (Ccn), detta anche copia carbone nascosta, traduzione dell'inglese blind carbon copy (Bcc), è un'intestazione dei messaggi di posta elettronica. I destinatari specificati nel campo Ccn ricevono una copia del messaggio inviato, ma il loro indirizzo viene nascosto agli altri destinatari del messaggio.

**DNS** Domain Name System

É un sistema utilizzato per la risoluzione di nomi dei nodi della rete (in inglese host) in indirizzi IP e viceversa. Il servizio è realizzato tramite un database distribuito, costituito dai server DNS.

**CSV** Comma-Separated Value srf

É un formato di file basato su file di testo utilizzato per l'importazione ed esportazione di una tabella di dati. Non esiste uno standard vero e proprio.

**XML** eXtensible Markup Language

É un linguaggio di markup che definisce e controlla il significato degli elementi contenuti in un documento o in un testo.

**CSRF** CSRF

abbreviato CSRF o anche XSRF, è una vulnerabilità a cui sono esposti i siti web dinamici quando sono progettati per ricevere richieste da un client senza meccanismi per controllare se la richiesta è stata inviata intenzionalmente oppure no. Diversamente dal cross-site scripting (XSS), che sfrutta la fiducia di un utente in un particolare sito, il CSRF sfrutta la fiducia di un sito nel browser di un utente.

**SQL** Structured Query Language

É un linguaggio standardizzato per database basati sul modello relazionale (RDBMS).

**IDE** Integrated Development Environment

É un software che, in fase di sviluppo, aiuta i programmatori nella progettazione e creazione di codice sorgente.

**HMAC** keyed-hash message authentication code

É un codice per l'autenticazione dei messaggi basata su una funzione hash.