DOCUMENTATION

By: Gabriel Osuobiem

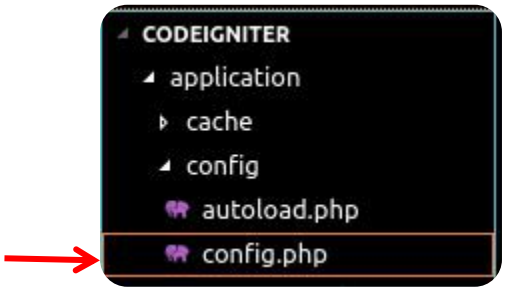
Table of Contents

Introduction

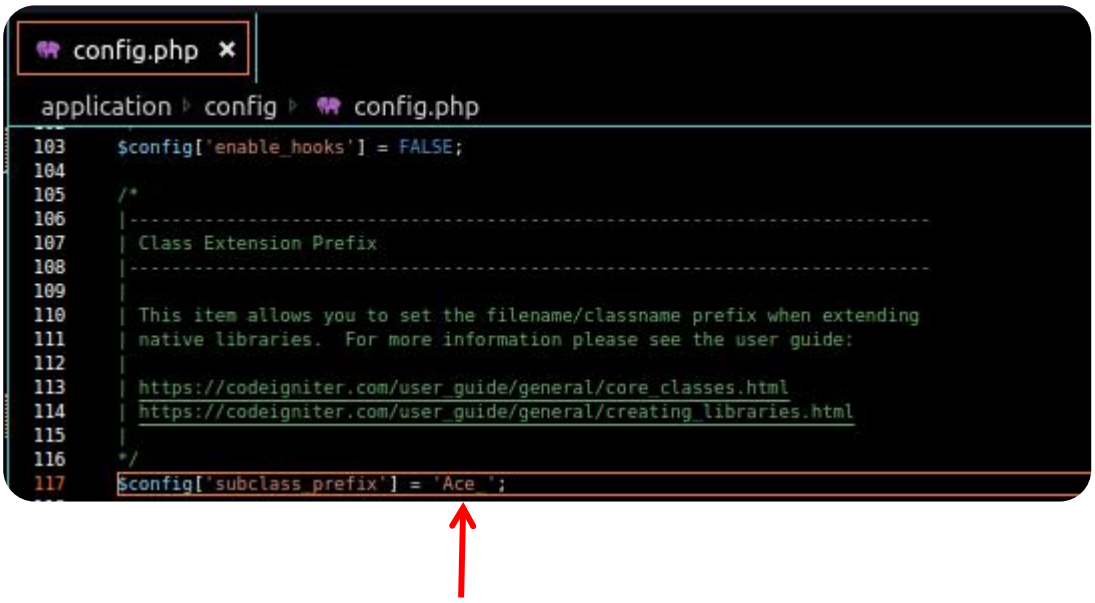
Ace CI (Ace CodeIgniter) is a database abstraction layer built atop CodeIgniter. With it's clean and well-documented code Ace CI eliminates overhead, code repetitions, bulky and dirty code. It enhances program and delivery speed, programmer efficiency and throughput. Hence, making life incredibly easy for the programmer.

Setup and Configuration

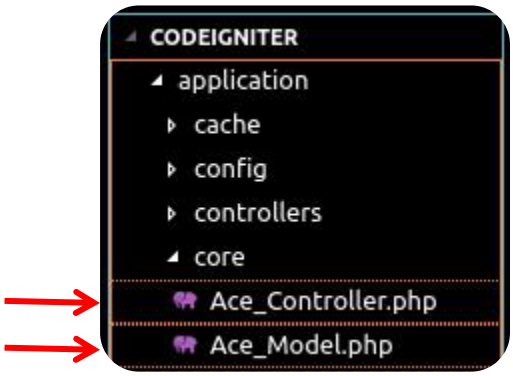
1. You have to make CodeIgniter aware of ACI (Ace CI). Navigate to the config folder in your project directory and open **config.php.** Like so:



In the **config.php** file, find **$config['subclass_prefix']** (in should be on line **117** or thereabout), change '**MY_'** to **'Ace_'**. Like so:



2. Copy the files **Ace_Controller.php** and **Ace_Model.php** to **your_project > application > core**. Like so:



3. When creating your models and controllers make sure they extend **Ace_Controller** and **Ace_Model** respectively, instead of **CI_Controller** and **CI_Model**. Like so:

```
class My_controller extends Ace_Controller
```

and

```
class My_model extends Ace_Model
```

Usage

This section will use some code examples to explain how you can use ACI.

Model Structure

All your models must have this structure.

```
class My_model extends Ace_Model {

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

  protected $table = 'database_table_name';

}
```

That's the beginning and the end of any model. ACI will handle the rest. Simple right? Yeah I know.

Selecting Data

The following operations allow you to retrieve records from a table. We'll use an example table called **users** with it's corresponding model **users_model**.

- **Selecting all records**

  Use the **get()** function without supplying any parameters.

  Like so:

```
$all_users = $this->users_model->get();
```

- **Looking for Specific Data**

where

The example below attempts to fetch users who live in Lagos.

```
$lagos_users = $this->users_model->get('city', 'Lagos');
```

The example below attempts to fetch users who live in Lagos and are above 22 years of age.

```
$where = array('city' => 'Lagos', 'age' => 22)
$lagos_users = $this->users_model->get($where);
```

OR

```
$where = "city='Lagos' AND age=22";
$lagos_users = $this->users_model->get($where);
```

or_where

The example below attempts to fetch users who are 22 years old and live in Lagos or Texas.

```
$users = $this->users_model->get([
  'where' => ['age' => 22, 'city' => 'Lagos'],
  'or_where' => ['city' => 'Texas']
]);
```

Fetch single record

To retrieve a single user record use:

```
$single_users = $this->users_model->getOne(2);
```

OR

```
$single_user = $this->users_model->getOne(['id' => 2]);
```

*// SELECT * FROM users WHERE id = 2*

- **Looking for Specific Data**

  like, or_like, not_like, and or_not_like

  Generate LIKE clauses, useful for doing searches.

```
$users = $this->users_model->get([
  'like' => ['name' => 'Gab'],
  'or_like' => ['name' => 'rie'],
  'not_like' => ['city' => 'Texa'],
  'or_not_like' => ['city' => 'Los A']
]);
```

  distinct, having, or_having

```
$users = $this->users_model->get([
  'distinct' => 'city',
  'having' => ['city' => 'Lag'],
  'or_like' => ['city' => 'Tex']
]);
```

// SELECT DISTINCT `city` FROM `user` HAVING `city` = 'Lag' OR `city` = 'Tex'

- **Ordering Results**

  order_by

```
$users = $this->users_model->get([
  'where' => ['city' => 'Lagos'],
  'order_by' => ['name' => 'DESC']
]);
```

*// SELECT * FROM `users` WHERE `city` = 'Lagos' ORDER BY `name` DESC*

- **Limiting or Counting Results**

limit

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

```
$users = $this->users_model->get([
  'where' => ['city' => 'Lagos'],
  'limit' => 12
]);
```

*// SELECT * FROM `users` WHERE `city` = 'Lagos' LIMIT 12*

```
$users = $this->users_model->get([
  'where' => ['city' => 'Lagos'],
  'limit' => [12, 3]
]);
```

*// SELECT * FROM `users` WHERE `city` = 'Lagos' LIMIT 3, 12*

Counting results

Determine the number of rows in a particular Active Record query.

```
$lagos_users = $this->users_model->getCount('city', 'Lagos');
```

// SELECT COUNT(*) AS `numrows` FROM `users` WHERE `city` = 'Lagos'

```
$lagos_users = $this->users_model->getCount([
  'where' => ['age' => 22, 'city' => 'Lagos'],
  'or_where' => ['city' => 'Texas']
]);
;
```

// SELECT COUNT(*) AS `numrows` FROM `users` WHERE `age` = 22 AND `city` = 'Lagos' OR `city` = 'Texas'

TIP: ***getCount()*** works like the ***get()*** function. The only difference is that ***getCount()*** returns the number of rows fetched.

Inserting Data

The following operations allow you to insert records in a table. We'll use an example table called **users** with it's corresponding model **users_model**.

create

```
$new_user = array(
  'name' => 'Gabriel Osuobiem',
  'age' => 22,
  'city' => 'Lagos',
);
$this->users_model->create($new_user);
```

// INSERT INTO `users` (`name`, `age`, `city`) VALUES ('Gabriel Osuobiem', 22, 'Lagos')

If you want the **last insert id** to be returned after insert then use:

```
$last_id = $this->users_model->create($new_user, true);
```

createBatch

If you wish to insert more than one record at a time.

```
$users = array(
     array(
       'name' => 'Gabriel Osuobiem',
       'age' => 22,
       'city' => 'Lagos',
     ),
     array(
       'name' => 'Albert Einstein',
       'age' => 76,
       'city' => 'Ulm',
     ),
     array(
       'name' => 'Steve Jobs',
       'age' => 56,
       'city' => 'San Francisco',
     )
);
$this->users_model->createBatch($users);
```

Updating Data

The following operations allow you to update records in a table. We'll use an example table called **users** with it's corresponding model **users_model**.

update

```
$user = array(
            'filter' => array('id' => 2),
            'data' => array(
                        'name' => 'Gabriel Igelle Osuobiem',
                        'age' => 23,
                        'city' => 'Manchester',
                      )
);
$this->users_model->update($user);
```

//UPDATE `users` SET `name` = 'Gabriel Igelle Osuobiem', `age` = 23, `city` = 'Manchester' WHERE `id` = 2

Deleting Data

The following operations allow you to delete records from a table. We'll use an example table called **users** with it's corresponding model **users_model**.

delete

```
$this->users_model->delete('id', 2);
```

OR

```
$this->users_model->delete(['id' => 2]);
```

TIP: You can use either **[]** or **array()**, they both work fine.

Entity Relationships

Ever wished CodeIgniter had entity relationships? Your wish has been granted. Just like magic **ACI** allows you to create relationships between database table entities and also carry out operations using these relationships. It supports **one-to-one, one-to-many, many-to-one,** and **many-to-many** relationships.

- **One-to-one**

  This section will explain how you can create **one-to-one** relationship and execute operations with it.

  In a university setting, a department/unit is headed by one HOD (Head of Department), and a HOD heads only one department/unit. This is a typical example of a **one-to-one** relationship, this will be our example.

  For clarity, a **one-to-one** relatonship must have what is known as the **host entity**. The table of this entity will carry a foreign key column that links it to the other entity. The **unit entity** will be our **host entity** for our example.

  Creating the relationship

  *** Example code is in the next page ***

Unit_model.php *// The model can be named anything*

```
class Unit_model extends Ace_Model {

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

  protected $table = 'units';

  public function hod() {
    $hod = [
      'table' => 'hods', // Table name of other entity
      'key' => 'id', // Name of referenced column in the units
                                                          table
      'host' => true, // true since this entity is the host
    ];

    return $this->hasOne($hod);
  }
}
```

HOD_model.php

```
class HOD_model extends Ace_Model {

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

  protected $table = 'hods';

  public function unit() {
    $unit = [
      'table' => 'units', // Table name of other entity
      'key' => 'hod_id', // Name of foreign key column in the
                                                     units table
      'host' => false, // false since this entity is not the host
      'ref_key' => 'id', // Name of referenced column in
                                                     hods table
    ];

    return $this->hasOne($unit);
  }
}
```

create

The following example code will show you how to **create one-to-one** related records. We will continue with our **unit/HOD** example. The code should be in the **controller**.

HOD_controller.php

```
class HOD_controller extends Ace_Controller {
                        .
                        .
                        .

  public function createUnit() {
    $unit_data = [
      'name' => 'Computer Science',
      'faculty' => 'Physical Science',
      'hod_id' => 1 // If not provided ACI will do it automatically
    ];

    $hod = $this->hod_model->getOne(1);  // Do not forget to
                                                          do this
    if($hod) {
      $this->hod_model->unit()->create($unit_data);
    }
  }
}
```

TIP: The **unit entity** will not be able to create a HOD because the **unit entity** is the **host entity.** It therefore has a field called **hod_id** which implies that a HOD has to exist in the database before the unit is created.

get

The following example code will show you how to **retrieve one-to-one** related records. We will continue with our **unit/HOD** example. The code should be in the **controller**.

HOD_controller.php

```
class HOD_controller extends Ace_Controller {
                        .
                        .
                        .

  public function getUnit() {

    $hod = $this->hod_model->getOne(1);  // Do not forget to
                                                  do this
    if($hod) {
      $unit = $this->hod_model->unit()->get();
    }
  }
}
```

Unit_controller.php

```
class Unit_controller extends Ace_Controller {
                        .
                        .
                        .

  public function getHOD() {

    $unit = $this->unit_model->getOne(1, 'hod_id');
                                  // Do not forget to do this
    if($unit) {
      $hod = $this->unit_model->hod()->get();
    }
  }
}
```

TIP: The **getOne()** function has an optional parameter called **$preserve**. It is used to specify the value of a field that should be stored for future use. If this is not specified **ACI** will automatically store the **id** field.

For example `$this->unit_model->getOne(1, 'hod_id');` will store **hod_id**

update

The following example code will show you how to **update one-to-one** related records. We will continue with our **unit/HOD** example. The code should be in the **controller**.

HOD_controller.php

```
class HOD_controller extends Ace_Controller {
                        .
                        .
                        .

  public function updateUnit() {

    $hod = $this->hod_model->getOne(1);  // Do not forget to
                                                  do this
    $unit_data = [
      'name' => 'Software Engineering'
    ];
    if($hod) {
      $this->hod_model->unit()->update($unit_data);
    }
  }
}
```

Unit_controller.php

```
class Unit_controller extends Ace_Controller {
                        .
                        .
                        .

  public function updateHOD() {

    $unit = $this->unit_model->getOne(1, 'hod_id');
                                        // Do not forget to do this
    $hod_data = [
      'name' => 'Gabriel I Osuobiem',
      'age' => 22
    ];
    if($unit) {
      $this->unit_model->hod()->update($hod_data);
    }
  }
}
```

delete

The following example code will show you how to delete **one-to-one** related records. We will continue with our **unit/HOD** example. The code should be in the **controller**.

HOD_controller.php

*** Example code is in the next page ***

```
class HOD_controller extends Ace_Controller {
                        .
                        .
                        .

  public function deleteUnit() {

    $hod = $this->hod_model->getOne(1);  // Do not forget to
                                                  do this
    if($hod) {
      $this->hod_model->unit()->delete();
    }
  }
}
```

Unit_controller.php

```
class Unit_controller extends Ace_Controller {
                        .
                        .
                        .

  public function deleteHOD() {

    $unit = $this->unit_model->getOne(1, 'hod_id');
                                  // Do not forget to do this
    if($unit) {
      $this->unit_model->hod()->delete();
    }
  }
}
```

- **<u>One-to-many & Many-to-one</u>**

This section will explain how you can create **one-to-many** and **many-to-one** relationships and execute operations with them.

In this section, we'll use an example of the relationship between a **Father** to his **Children.** This is a typical example of **one-to-many** and **many-to-one** relationships **as one Father** has **many Children** and **many Children** have **one Father.**

Creating the relationship

Father_model.php *// **one-to-many** relationship*

```
                    .
                    .
public function children() {
  $children = [
    'table' => 'children', // Table name of the child entity
    'foreign_key' => 'father_id', // Name of foreign key
                                  column in the children table
  ];

  return $this->hasMany($children);
}
                    .
                    .
```

Child_model.php *// **many-to-one** relationship*

```
                    .
                    .
public function father() {
  $father = [
    'table' => 'fathers', // Table name of the child entity
    'ref_key' => 'id', // Name of referenced column in the
                                                fathers table
  ];

  return $this->belongsTo($father);
}
                    .
                    .
```

create

The following example code will show you how to **create one-to-many** related records. We will continue with our **father/child** example.

The code should be in the **controller**.

Father_controller.php

```
                    .
                    .

public function createChildren() {
  $children_data = array(
    [
      'name' => 'Gabriel Osuobiem',
      'age' => 22,
      'father_id' => 1  // If not provided ACI will do it automatically
    ],
    [
      'name' => 'Precious Osuobiem',
      'age' => 12
    ]
  );

  $father = $this->father_model->getOne(1);   // Do not
                                               forget to do this
  if($father) {
    $this->father_model->children()->create($children_data);
  }
}
                    .
                    .
```

TIP: As you already know, the **child entity** will not be able to create a father because the father has to exist in the database before the child is created.

get

The following example code will show you how to **retrieve one-to-many** and **many-to-one** related records. We will continue with our **father/child** example.

The code should be in the **controller**.

Father_controller.php

```
                    .
                    .

public function getChildren() {

  $father = $this->father_model->getOne(1);  // Do not forget
                                                to do this
  if($father) {
    $children = $this->father_model->children()->get([
      'order_by' => ['age' => 'DESC']  // This is optional
    ]);
  }
}
                    .
                    .
```

Child_controller.php

```
                    .
                    .

public function getFather() {

  $child = $this->child_model->getOne(1, 'father_id');
                                          // Do not forget to do this
  if($child) {
    $father = $this->child_model->father()->get();
  }
}
                    .
                    .
```

update

The following example code will show you how to **update many-to-one** related records. We will continue with our **father/child** example.

The code should be in the **controller**.

Child_controller.php

```
                    .
                    .

public function updateFather() {

  $child = $this->child_model->getOne(1, 'father_id');
                                        // Do not forget to do this
  $father_data = [
    'name' => 'Ferdinand I Osuobiem',
    'age' => 50
  ];
  if($child) {
    $this->child_model->father()->update($father_data);
  }
}
                    .
                    .
```

TIP: As you may have guessed, allowing a father to update children data will make the code dirty and kind of confusing. This feature was not included for this reason.

delete

The following example code will show you how to **delete one-to-many** and **many-to-one** related records. We will continue with our **father/child** example.

The code should be in the **controller**.

Father_controller.php

```
                    .
                    .

public function deleteChildren() {

  $father = $this->father_model->getOne(1);   // Do not
                                                forget to do this
  if($father) {
    $this->father_model->children()->delete();
  }
}
                    .
                    .
```

Child_controller.php

```
                    .
                    .

public function deleteFather() {

  $child = $this->child_model->getOne(1, 'father_id');
                                          // Do not forget to do this
  if($child) {
    $this->child_model->father()->delete();
  }
}
                    .
                    .
```

- **Many-to-many**

This section will explain how you can create **many-to-many** relationship and execute operations with them.

In this section, we'll use an example of the relationship between **Customers** and **Products.** This is a typical example of **many-to-many** relationship **as Customers** can puchase various **Products** and **Products** can be purchased by **many Customers.**

For clarity, a **many-to-many** relatonship occurs when multiple records in a table are associated with multiple records in another table. Therefore, there has to be a **pivot table** that relates both tables.

Creating the relationship

Customer_model.php

```
                    .
                    .
public function products() {
  $products = [
    'relative' => 'products', // Table name of the product entity
    'pivot' => 'customers_products', // Name of the pivot table
    'relative_key' => 'product_id', // Name of product foreign
                                      key column in the pivot table
    'foreign_key' => 'customer_id', // Name of customer foreign
                                      key column in the pivot table
    'ref_key' => 'id' // Name of referenced column in the products
                                                              table
  ];

  return $this->belongsToMany($products);
}
                    .
                    .
```

Product_model.php

```
                    .
                    .
public function customers() {
  $customers = [
    'relative' => 'customers', // Table name of the customer entity
    'pivot' => 'customers_products', // Name of the pivot table
    'relative_key' => 'customer_id', // Name of customer foreign
                                       key column in the pivot table
    'foreign_key' => 'product_id', // Name of prodct foreign key
                                       column in the pivot table
    'ref_key' => 'id' // Name of referenced column in the customers
                                       table
  ];

  return $this->belongsToMany($customers);
}
                    .
                    .
```

create

The following example code will show you how to **create many-to-many** related records. We will continue with our **customer/product** example.

The code should be in the **controller**.

Customer_controller.php

*** Example code is in the next page ***

TIP: Operations of a **many-to-many** is the same between both entities. Hence, we'll only use the **customer** entity to show examples.

```
                    .
                    .

public function buyProducts() {
  $products_data = array(134, 18, 4, 1); // Array|Object of products ids
  $customer = $this->customer_model->getOne(1);  // Do not forget to do
                                                                    this
  if($customer) {
    $this->customer_model->products()->create($products_data);
  }
}
                    .
                    .
```

get

The following example code will show you how to **retrieve many-to-many** related records. We will continue with our **customer/product** example.

The code should be in the **controller**.

Customer_controller.php

```
                    .
                    .

public function getProducts() {

  $customer = $this->customer_model->getOne(1);  // Do not
                                                  forget to do this
  if($customer) {
    $products = $this->customer_model->products()->get([
      'order_by' => ['age' => 'DESC']  // This is optional
    ]);
  }
}
                    .
                    .
```

update and delete

The features above will be present in the next version of **Ace CI**. Meanwhile, proceed to the next section for other fun tweaks.

- **Other Tweaks**

This section will teach you how to use other functionalities in **Ace CI**.

Using CodeIgniter functions

ACI has no restrictions in case you wish to use any CodeIgniter's core function in your code. For example, if you want to call a core **db** function in your code you don't have to bother loading **database(). ACI** has already loaded it in it's **Ace_Model** class, you can just proceed and call any core function you wish to use. Like so:

```
$this->db->where('city', 'Lagos');
```

Loading Helpers and Libraries

ACI has provided a technique such that **libraries** and **helpers** will be loaded once yet can be available to every controller.

In the **Ace_Controller.php** file, you can load all your **libraries** and **helpers** in the **class constructor** and they will be available for every controller to access at once. Like so:

```
class Ace_Controller extends CI_Controller {

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

    $this->load->library('session');
    $this->load->helper('form');
  }
                    .
                    .
}
```

Data Dump

This functionality dumps the value and type of any variable passed as a parameter to its function **dd($var)**, then it kills the script. It works only in controllers. Like so:

```
$this->dd($array);
```

Hash

This functionality will hash the value of any variable passed as a parameter to its function **hash($var)**. The function works with an **encryption key** which has to be specified in **the config.php** file. To make the **encryption key** available follow the steps listed below:

- Open the **config.php** file
- Paste this `$config['encryption_key'] = 'your_key_here';` at the end of the file (the last line).
- Replace `your_key_here` with a randomly generated string. It can be as lenthy as you wish.

After carrying out those steps you can now proceed to use the **hash()** function, like so:

```
$this->hash($password);
```

It works only in controllers.

Support

For enquiries, thoughts, complaints, support, or guidelines outside the scope of the scope of this document please contact me.

CONTACT

Phone: +234-706-959-7156

Gmail: osuobiem@gmail.com

LinkedIn: https://www.linkedin.com/in/gabriel-osuobiem-b22577176/

Skype: live:osuobiem

Github: https://github.com/osuobiem

I am available for freelance jobs.