Cradlecore MVC 1.0 Documentation

February, 2012
Written by A. Soto.

Table of Contents

Requirements

- PHP 5 >= 5.2, tested only in PHP 5.2 and 5.3
- Apache with mod_rewrite module enabled. For ubuntu mod_rewrite configuration check this links : http://www.ghacks.net/2009/12/05/enable-mod_rewrite-in-a-ubuntu-server/ and http://httpd.apache.org/docs/2.2/howto/access.html
  Most hosting providers provides this configuration.
- PHP **memcache** extension installed and enabled, if you are going to use the cache addon, you will need to have this extension enabled. Please check the cache section.

Installation

To install cradlecore mvc you will need to check if your apache **mod_rewrite** module is installed and enabled and configured to let you use **.htaccess** file in linux environments or any other file in windows environments, see **.htaccess on windows** that let you use rewrite rules in specific folders in your server root.

The next step is download the compressed file from sourceforge. Extract the contents in a location where the php include_path can reach. For example in the PEAR directory (recommended when you need to share the library between multiple applications) or relative location to your future application in your apache server.

As you can see in the file downloaded, there are two files named: cradlecore and cradlecore.bat, the first one, is a cli tool for linux environments and the second one a cli tool for windows. This tools can help in the creation of projects and projects modules.

To use the command line tools you will need to register the path where those files are located in your operative system PATH environment variables.

For **linux**, you have to execute in command line the following commands, this change will be temporal if you type this command directly in the command line:

export PATH=$PATH:*library location*

example: **export PATH=$PATH:*/var/www/sourceforge/cradlecoremvc***

Or to make this change permanent try to add the same command line to the hidden **.bashrc** or **.bash_profile** (filename depends on linux distro) files located in your user home directory. Try closing the commandline and loading again, when the **cradlecore** command be typed it will be recognized.

In **windows** environment is a little bit different, execute the following command in command line:

SET PATH=%PATH%;*library location*

example: **SET PATH=%PATH%;*C:\wamp\www\cradlecoremvc***

To make this change permanent on windows operative systems, the cradlecoremvc path should be added in environment variables. For windows xp, check this link http://support.microsoft.com/kb/310519 .

With those configurations you can execute from command line the command **cradlecore**

Create an application

To easy create an application use the following command inside your apache http root directories as following:

cradlecore create project *ProjectName*

This command will generate a directory called ***ProjectName*** with the neccesary files to start creating the application.

Application structure

The application structure follows a very simple archetype easy to understand.

```
ProjectName
        ├─assets
        ├─configuration
        ├─modules
        ├─.htaccess
        ├─htaccess.txt
        └─index.php
```

- **assets =** Folder to store files such as static files, css, js and many others.
- **configuration =** Where application.json and routes.json are located.
- **modules =** Folder where application modules are created.
- **htaccess =** Apache rewrite rules for the index.php in order to redirect the traffic to it
- **htaccess.txt =** Apache rewrite rules alternative for the index.php in order to redirect the traffic to it.
- **index.php =** File to listen the traffic coming to the application path, **if you see in line 5**, there is the **absolute path to reach the library**, when the application is moved to different environment, this path should be changed where the library is located.

Application file

The application.json file is located in the application configuration folder, is one of the main configuration files that is required to use the framework. Can be configured at parent module level and child module level.

For example:

```
{
    "name": "Webapp",
    "environment": "development",
    "environments": {
        "development": {
            "db_host": "127.0.0.1",
             "db_name": "dev_employees"
        },
        "production": {
            "db_host": "domain.org",
            "db_name": "employees"
        }
    },
    "modules_configuration": {

        "base": {
            "frame": "html5",
            "title": "Main Page",
            "type": "MainLayout",
            "config": {
                "requiresLoggedUser": false,
                "children": {
                    "header": {
                        "type": "Header"
                    },
                    "footer": {
                        "type": "Footer"
                    }
                }
            }
        },

        "index": {
            "extends": "base",
         "config": {
            "children": {
              "content": {
                 "type": "Login"
              }
            }
         }
      },

      "admin_index": {
         "extends": "index",
         "title": "Welcome Admin",
         "config": {
            "children": {
              "subcontent": {
                 "type": "Admin",
                 "config": {
                    "adminUsers": ["admin", "editor", "boss"]
                 }
              }
            }
         }
      },

      "admin": {
         "extends": "base",
         "config": {
            "requiresLoggedUser": true,
            "children": {
              "content": {
                 "type": "Admin",
                 "action": "validator"
              }
            }
         }
      }


   }

}
```

The following table shows the meaning of the objects and fields of the application.json

FIELD	TYPE	DESCRIPTION
name	string	Name used when application is created with the command line tool
environment	string	Represents the current application environment based on application environments defined in **environments** object
environments	object	Represents all the available environments by default when application is created are 2 available environments objects called **development** and **production** . Configuration values in the environments are user defined except the cache object. See cache section.
modules_configuration	object	Specified all the modules configurations exposed for parent modules or for routes mapping. A module is parent if its is on a high level than other modules relative to its configuration tree.
{instance_name_key}	string	Is a custom name user defined for the module that will be instanced **e.g :** In the application.json showed above, the keys “index”, “base”, “content”, “header” and “footer”

Module instance configuration:

FIELD	TYPE	DESCRIPTION
type	string	Is the module name of the module to be implemented by the instance.
config	object	Instance configuration object

title	string	Represents the page title, is user defined, and it only can be set only for top parent modules instances in application.json or in the top parent instances that extends from other parent instance
frame	string	Defining this field and passing a value: ***html5*** or ***xhtmlStrict,*** the base view will use an html5 or xhtml markup layout. And also the **developer will be able to add css, js or other blob data dinamically from the controller.** Is used in modules that are located in the top of the tree as shows the application.json above.
extends	string	Extends another instance to be used as a base for the current instance, is a good practice to avoid objects reconfigurations.

The config object :

FIELD	**TYPE**	**DESCRIPTION**
children	string	Specifies the children modules for the current module. Those childrens specify also the type and config for each one, children is a reserve word in configuration file.
{configuration_key}	any	Key is user defined, the same as the value that will contain.

Create a module

To create an application module, first enter in console in the application folder, use the following commands inside your apache http root directories as following for both cases, linux and windows:

Enter into project folder:

cd *ProjectName*

And then :

cradlecore create module *ModuleName*

This command will generate a directory called ***ModuleName***

Modules structure

```
ModuleName
        ├─assets
        ├─models
        ├─views
        └─controller.php
```

- **assets =** Folder to store files such as static files, css, js and many others for an specific module.
- **models =** Folder to store specific module models by default module.php is created an associated to the controller's property model . By default is created a method called getData, where you can include some of existing logic.
- **views =** Where the views are located by default index.view.php is created and associated to the controller's index action method.
- **controller.php = The module's controller by default is created an index action method.**

Routes file

The routes.json file in the application configuration directory is used to map the urls to application.json modules instances for example:

```
{
   "entry_point": "/Webapp/",
   "url_mappings": {

      "index_page": {
         "verbs": [ "get" ],
         "path": "/",
         "call": "index.index"
      },

      "index_page2": {
         "verbs": [ "get" ],
         "path": "/login/:user/",
         "call": "index.index"
      },

      "admin_page": {
         "verbs": [ "get" ],
         "path": "/admin",
         "call": "admin.index"
      }

   }
}
```

Routes mapping object fields and values:

FIELD	**TYPE**	**DESCRIPTION**
entry_point	string	Entry point (url path or path section) where the application will be accessed for example if the application is located in the root's top location of the hosting or local apache the entry point should be “” . If the application is located in a

		htdoc folder called **“Webapp”** the entry point should be **“/Webapp/” .**
url_mappings	object	Group of routes mapping, each route has to have an user defined key. For example in the routes.json showed above the values "index_page", "index_page2" and "admin_page" .

Mapping object:

FIELD	**TYPE**	**DESCRIPTION**
verbs	string array	Array of http methods that will support the mapping. For example “get”, “post”, “put”, etc.
path	string	The relative path to the entry point where route is going to be mapped
call	string	It makes reference to the instance defined in application.json and also makes reference to the controllers action method that will be called.

Views

The view has a naming convention:

{*controller_action_method*}.{*mobile_device_id* => optional }.view.php

- **controller_action_method =** Is the action method that the view will make reference, if an action method called **login** is used it will make reference to a view called **login.view.php .**
- **mobile_device_id =** The device id based on the defined **devices.json** file entries, see devices section.

To see how to implement mobile devices views check mobile devices section.

To create the view manually, take care of this convention. When a module is created by default the index.view.php view is created. This view is referencing the controller's action method called index that also is created when module is created.

The following code snippets illustrates how to pass values from controller to view:

As the snipped shows below you have to call the method done and pass an associative array with the keys and values that you want to pass to the view

```
<?php

/**
 * Description of AdminController
 *
 * @author alejandro.soto
 */
class AdminController extends CradleCoreController {

    public function __construct() {

    }

    /**
     * Index action method
     *
     */
    public function index() {
        $data = $this->model->getData();
        $this->done(array('name' => 'Someone', 'somedata' => $data));
    }

}

?>
```

In the view, just only call the method get specifying the key of the value that will be echoed

```
<h1>Admin</h1>
<?php echo $this->get('name') ?>
```

Frames

Frames are a kind of html layout, it enables to add assets such as javascripts, css, metadata, etc.

Composite

A composite module instance is a module configured with one or more children modules in the application.json configuration file. Also it requires to be called from the parent view.

As the following snippet illustrates, the composite module requires one or more children, in this case the composite module is the module instance called “base” because contains 2 children module instances called “header” and “footer” . Also those 2 modules can be composite modules.

```
{
    "name": "Webapp",
    "modules_configuration": {

        "base": {
         "frame": "html5",
         "type": "MainLayout",
         "config": {
            "children": {
               "header": {
                  "type": "Header"
               },
               "footer": {
                  "type": "Footer"
               }
            }
         }
      }

   }
}
```

In the parent's(“base” type “MainLayout”) module view, the children shoud be called with the function called **child** , this function receive the user defined key for the child instance that is on application.json .

For example:
MainLayout view

```
<div>
   <?php echo $this->child('header') ?>

   <?php echo $this->child('content') ?>

   <?php echo $this->child('footer') ?>
</div>
```

Extending configurations

Extend from another module instance is very useful when **reuse** the same layout is required, many times the content of the page is the only section that will change between all the site's pages, and the header and footer will be repeated, in this case this can be configured in the **application.json**

The following snippets illustrates how module instances “index” and “admin” extends from the module instance named “base”. This means that index will show index contents but extending header and footer from “base” instance. The same case applies for “admin” instance.

```
{
   "name": "Webapp",
   "modules_configuration": {

      "base": {
         "frame": "html5",
         "type": "MainLayout",
         "config": {
            "children": {
               "header": {
                  "type": "Header"
               },
               "footer": {
                  "type": "Footer"
               }
            }
         }
      },

      "index": {
         "extends": "base",
         "config": {
            "children": {
              "content": {
                 "type": "Login"
              }
            }
         }
      },

      "admin": {
         "extends": "base",
         "config": {
            "children": {
                "content": {
                    "type": "Admin"
                }
            }
        }
    }


    }

}
```

Assets

Assets stuff like css, javascript and others are very important in every web application. In our application structure there are to places to store assets, in the application root **/assets** and in the module folder ***/modules/ModuleName/assets*** The controller provide us an addon to add assets in run-time which is called **assets** object. It can be referenced as the following snippet illustrates:

Where the first parameter is the path inside the application or also it can be a full url pointing to external asset. In addBlob function case the first parameter should be a metadata markup for example.
The second parameter is to specify the location where the tag should be positioned **'top'** or **'bottom'** .

```
public function index() {
      $data = $this->model->getData();
      $this->assets->addCss('/modules/Login/assets/index.css');
      $this->assets->addJs('/modules/Login/assets/index.js');
      $this->assets->addJs('/modules/Login/assets/index.js', 'bottom');
      $this->assets->addBlob('<meta http-equiv="cache-control" content="no-cache" />');
      $this->done(array('name' => 'Someone', 'somedata' => $data));
 }
```

To view an API reference of this functionality see this
http://sourcetek.org/projects/cradlecore-mvc/api-docs/

HTTP

URLs

Internal links in web applications using mvc are very common. The framework expose an addon method to deal with this issue.

For example, the following snippet print an url link in a view to the index page, the method **url** receives the path as the parameter, this method can also be called from the controller.

```
<h1>Footer</h1>
<a href="<?php echo $this->http->url('/') ?>">test</a>
```

Redirects

Redirects can be done with the controller's **http** addon function called **redirect**, it receives the url as parameter.

```
$this->http->redirect('http://google.com');
```

Headers

Headers can be retrieved and set using the **http** addon. To retrieve the current request headers the function **getHeaders** can be called and it will return an associative array with the request headers.

```
$headers = $this->http->getHeaders();
```

To set a response header the following function can be used :

```
$this->http->setHeader('Cache-Control: no-cache, must-revalidate');
```

Params

As you know parameters can be passed within the url or in the POST request body, the framework also exposes the addon called params which let you read the parameters passed in the query string, in the path or in the request body.

For example if you want to pass a parameter called user in the path, the path should be specificied in the route definition in routes.json as the following snippet illustrates:

```
"index_page2": {
    "verbs": [ "get" ],
    "path": "/login/:user/",
    "call": "index.index"
}
```

And in the controller just the addon method **getParamFromUrl** should be called :

```
<?php

/**
 * Description of LoginController
 *
 * @author alejandro.soto
 */
class LoginController extends CradleCoreController {

    public function __construct() {

    }

    /**
     * Index action method
     *
```

```
   */
   public function index() {
      $this->done(array('name' => 'Someone', 'username' => $this->params-
>getParamFromUrl('user')));
   }

}

?>
```

To read the query string params and read values from the body you should use the function :

```
$param = $this->params->getParam('key')
```

To view an API reference of this functionality see this
http://sourcetek.org/projects/cradlecore-mvc/api-docs/

Environments configuration

Environments variables are values that can be set in the application.json file, environments are defined in the **environments** object as you can see in the snippet below. Differents values for the same keys can be defined changing between the available **environments** with the **environment** value

Environments variables are shared between all the controllers in the application and it can be read as the following snippet illustrates how to retrieve the value in development environment called **db_host** :

Controller code snippet :

```
$db_host = $this->AppConfig->db_host;
```

Application.json code snippet :

```
{
   "name": "Webapp",
    "environment": "development",
    "environments": {
        "development": {
            "db_host": "127.0.0.1",
             "db_name": "dev_employees"
        },
        "production": {
            "db_host": "domain.org",
            "db_name": "employees"
        }
    },
    "modules_configuration": {

          ...

     }
}
```

Module configuration

Also the values defined for an specific module can be read from the controller, for example using a module , requiresLoggedUser is configuration variable user defined

```
{
    "name": "Webapp",
    "environment": "development",
    "environments": {
        "development": {

                    ...

        },
        "production": {

                    ...

        }
    },
    "modules_configuration": {

        "admin": {
            "type": "Admin",
            "config": {
                "requiresLoggedUser": true,
                }
            }
        }


    }

}
```

Those configuration variables that can be configured from instance definition in the **config** object in the **application.json** can be read in the controller for example using a similar code :

```
$requiresLoggedUser = $this->config->requiresLoggedUser;
```

Cache

Use memcached server is a common practice in websites that require high performance. This framework provides the addon called cache**.** It requires the php **memcache extension** enabled and also requires an additional configurations in the **environments** objects in the application.json to get it working. To see how to install the memcache extension in php check this http://php.net/manual/en/memcache.installation.php .

As you can see in the code application.json below, a **cache** object has to be defined in the application configuration, this object requires two properties the memcached server host and the port where the server is running. If the framework detect that extension is not available it or can connect to memcached server the framework will throw an error . The cache object in application configuration is exclusive and cannot be used for other configuration value.

```
{
    "name": "Webapp",
    "environment": "development",
    "environments": {
        "development": {
            "cache": {
```

```
            "host": "127.0.0.1",
            "port": 11211
         }
      },
      "production": {
         "cache": {
            "host": "127.0.0.1",
            "port": 11211
         }
      }
   },
   "modules_configuration": {

        …

   }
}
```

As you read there is an addon called cache exposed to cache data in the application controller and can be explicitly used as the following code shows:

To store data the following function can be used passing a key, data, and the timeout in seconds as optional parameter.

```
$stored = $this->cache->store($key, $data, 60);
```

To retrieve data the following function can be used passing the data key.

```
$value = $this->cache->retrieve($key);
```

The following function will clean and remove all the data stored in memcached server.

```
$flushed = $this->cache->reset();
```

Mobile Devices

Cradlecore MVC also supports mobile devices, it provides a simple json configuration named **devices.json** located inside the application **configuration** directory. The mobile supports covers a controller addon to provide mobile detection manually based on the devices.json configuration and covers the views, for example if there is a logic for an index module, it is independent from its rendering so the only component that will change are the views.

File devices.json:

This file is located in configuration directory and it is created when the project is created, can be change whenever the developer needs, it contains the **mobile_device_id** and for each id one or more user agents (also can be a part of the device user agent. For example **iPhone;** is a section of the iphone's user agent and it will match when the framework receives an iphone's request, the same happened with the other definitions).

```
{
   "iphone": [
      "iPhone;"
           ],
   "ipad": [
       "iPad;"
         ],
   "android": [
       "Android"
            ],
   "blackberry": [
       "BlackBerry"
              ],
   "webos": [
       "webOS"
          ]
}
```

Devices table :

FIELD	TYPE	DESCRIPTION
mobile_device_id	string	Identifies each mobile configuration. It is used to create the mobile views. See views section.

mobile_device_useragents	string array	Array of user agents or portions of user agents to validate if the requestor device is mobile or not.

Mobile Views:

The views for mobile devices have to be created manually relative to the default **index.view.php.** Based on the above **devices.json** sample you can create a view for iphone, ipad, androd, blackberry and webos called for example **index.iphone.view.php**, **index.ipad.view.php**, **index.android.view.php**, **index.blackberry.view.php** and **index.webos.view.php**, as you read this file can be modified and more devices can be added. With this functionality the developer can create different markups for each devices. Take care of the **views convention**, see views section. If no view was created for a devices definition by default the default view will be used.

Controller device addon:

In order to detect which devices is making the request the addon called device can be used for retrieve the **mobile_device_id**, if is not a valid devices the getDeviceName() function will return null but if it is valid the **mobile_device_id** will be returned

```
$deviceName = $this->device->getDeviceName();
```

Another controller addon function that can be called is the isDevice() function, it return false or true.

```
$isDevice = $this->device->isDevice();
```

Appendix

.htaccess on windows
Windows by default does not accept files without name so we will need to use a file with different name, for example htaccess.txt

To use htaccess.txt as an .htaccess file you need to setup your httpd.conf file located in the folder of the apache installation and add the following line and save:

```
AccessFileName htaccess.txt
```