The MVC pattern in Common Utilities - Tutorials

42066

NEWS_PDF_AUTHOR: bitcero

NEWS_PDF_DATE: 2014/11/18 21:20:00

Probably very few of you know it, but Common Utilities have included since version 2.2, a basic implementation of the pattern **MVC** (Model - View - Controller). In this article I will give you a basic explanation of its operation in Common Utilities and integrated modules. If you still do not know what MVC is, please read this article on Wikipedia to learn more about it. **How MVC works in Common Utilities** When a module uses the MVC features of Common Utilities, all requests are received through the URL and it's the job of RMCommon to receive, process and direct them to the appropriate module. To achieve that, RMCommon includes an appropriate option to specify where to receive requests for each module. This is done through the configuration, indicating the URL you use each module. For example, if the module is that we look for is located in the "inventory" directory, and RMCommon configuration has established as their path to the folder "inventory" when RMCommon receives a request to the URLhttp://sitio.com/inventorios automatically redirect the request tohttp://sitio.com/modules/inventory . This means that the module will respond to all requests made ??with misitio.com/inventarios . **URL Parameters** Once RMCommon knows where to locate each module, you can tell that we get the module by specifying the parameters of the URL. Parameters provided must be written in the form module / controller / action / other-parameters . This simple format allows all requests to the module, which are handled by RMCommon follows: Common Utilities finds the appropriate driver folder controllers within the module's directory. Take for example the URLhttp://sitio.com/ library / books / list / category / bestsellers / The process is as follows: The corresponding module is located library. Depending on the routes that have been configured, this directory could match the directory of the module or be a different one. In this sample library is the directory of the module. Common Utilities driver looking books in the directory controllers of the module library , and loads the PHP class. Now find the corresponding method to the action list and processes the request by passing the parameters category = bestsellers . These parameters must always be in pairs. After processing the data, Common Utilities get the template (view) and returns the corresponding result. **Some conventions in this** How to locate the controllers? To begin, the drivers should be located as files within the directory controllers of each module. In addition, there are certain rules for naming files that contain drivers. In our example (yes, the library) the driver file should be called books-controller.php . In addition, this file should contain a class, the controller itself, named as follows: Library_Books_Controller and must inherit from the main class RMController . Finally, the class must contain a method called list , which will be invoked by RMCommon to present a result. Until here everything is clear? So these are the rules: - The driver files must be located in the directory controllers of the module. - The file name must follow the rule -controller.php . - The controlling class within the file must be named _ _controller , and should contain as many methods as actions are to be processed. The methods / actions should be named according to

the action requested by the URL. If the action is called form , the created class must contain a method form () . If the action is called categories-form , then the class must contain a method called categories_form () . A controller class looks like this:

```
 class  Mymodule_Nombrecontrolador_Controller  extends  RMController
{
   use  RMModuleAjax ,  RMProperties ,  RMModels ;

   public  function __construct () {
      parent :: __construct ();
      $ this -> default  =  'index' ;  // default action
      $ this -> controller =  'categories' ;
   }

   public  function index () {
      // Logic index action

      $ This -> tpl -> header ()
      requires $ this -> parent -> view ;
      $ this -> tpl -> footer ();
   }
}
```

What about the models? Models are only accessible through the controller. This means that they can only be used by the methods of the controller class. The models also have some specific rules: - Must be located in the module's /models folder . - A model file must be named as -model.php . - The file must contain a class named _ _model . A typical statement from an exact model would be:

```
 class  Mymodule_Nombremodelo_Model  extends  RMActiveRecord

{
   use  RMModels ;

   public  function __construct () {

      parent :: __construct (  'model' ,  'module'  );

      / **
       * Titles table fields
       * /
      $ this -> titles = array (
         'column'  => __ (  'Column Title' ,  'module'  ),
         'column2'  => __ (  'Title column2' ,  'module'  ),
         ...
      );

   }
}
```

And the views? Eventually we get the Views. These are actually templates that are derived from the name of the action. For the above example, where the module is used library and the driver is requested books for executing the method (action) list , Common Utilities get the template list.php , because it is the one that corresponds to the action list . Easy, right? It follows that the view files (templates) should be appointed as the action (controller method) running. If our action is the name of form , so our staff must be appointed form.php . If the action is the name -form categories , also our file should be named categories-form.php . One more thing. The Views, as in any module, should be stored in the directory templates module, but not directly, but in appropriate subfolder, depending on the following cases. - Templates Folder can contain standard templates (as commonly used in the modules). - Modules can have templates for the control panel or section templates for public, therefore within the directory /templates there should be two subfolders: backend and frontend . - Within each of these subfolders there should be a new subfolder for each driver that handles Views. If the driver is called categories, then there must be a subfolder called categories where the Views will be kept for each action. This new approach of module development in XOOPS enables faster and more structured development. Furthermore, with its auxiliary objects, Common Utilities facilitates the implementation of AJAX in modules allowing more intuitive and easy to use experience for users.

Probably very few of you know it, but Common Utilities have included since version 2.2, a basic implementation of the pattern **MVC** (Model - View - Controller). In this article I will give you a basic explanation of its operation in Common Utilities and integrated modules. If you still do not know what MVC is, please read this article on Wikipedia to learn more about it. **How MVC works in Common Utilities** When a module uses the MVC features of Common Utilities, all requests are received through the URL and it's the job of RMCommon to receive, process and direct them to the appropriate module. To achieve that, RMCommon includes an appropriate option to specify where to receive requests for each module. This is done through the configuration, indicating the URL you use each module. For example, if the module is that we look for is located in the "inventory" directory, and RMCommon configuration has established as their path to the folder "inventory" when RMCommon receives a request to the URLhttp://sitio.com/inventorios automatically redirect the request tohttp://sitio.com/modules/inventory . This means that the module will respond to all requests made ??with misitio.com/inventarios . **URL Parameters** Once RMCommon knows where to locate each module, you can tell that we get the module by specifying the parameters of the URL. Parameters provided must be written in the form module / controller / action / other-parameters . This simple format allows all requests to the module, which are handled by RMCommon follows: Common Utilities finds the appropriate driver folder controllers within the module's directory. Take for example the URLhttp://sitio.com/ library / books / list / category / bestsellers / The process is as follows: The corresponding module is located library. Depending on the routes that have been configured, this directory could match the directory of the module or be a different one. In this sample library is the directory of the module. Common Utilities driver looking books in the directory controllers of the module library , and loads the PHP class. Now find the corresponding method to the action list and processes the request by passing the parameters category = bestsellers . These parameters must always be in pairs. After processing the data, Common Utilities get the template (view) and returns the corresponding result. **Some conventions in this** How to locate the controllers? To begin, the drivers should be located as files within the directory controllers of each module. In addition, there are certain rules for naming files that contain drivers. In our example (yes, the library) the driver file should be called books-controller.php . In addition, this file should contain a class, the controller itself, named as follows: Library_Books_Controller and must inherit from the main class RMController . Finally, the class must contain a method called list , which will be invoked by RMCommon to present a result. Until here everything is clear? So these are the rules: - The driver files must be located in the directory controllers of the module. - The file name must follow the rule -controller.php . - The controlling class within the file must be named _ _controller , and should contain as many methods as actions are to be processed. The methods / actions should be named according to the action requested by the URL. If the action is called form , the created class must contain a method form () . If the action is called categories-form , then the class must contain a method called categories_form () . A controller class looks like this:

```
class Mymodule_Nombrecontrolador_Controller extends RMController
{
   use RMModuleAjax , RMProperties , RMModels ;

   public function __construct () {
      parent :: __construct ();
      $ this -> default = 'index' ; // default action
```

```
        $ this -> controller =  'categories' ;
    }

    public  function index () {
        // Logic index action

        $ This -> tpl -> header ()
        requires $ this -> parent -> view ;
        $ this -> tpl -> footer ();
    }
}
```

What about the models? Models are only accessible through the controller. This means that they can only be used by the methods of the controller class. The models also have some specific rules: - Must be located in the module's /models folder . - A model file must be named as -model.php . - The file must contain a class named _ _model . A typical statement from an exact model would be:

```
 class  Mymodule_Nombremodelo_Model  extends  RMActiveRecord

{
    use  RMModels ;

    public  function __construct () {

        parent :: __construct (  'model' ,  'module'  );

        / **
         * Titles table fields
         * /
        $ this -> titles = array (
            'column'  => __ (  'Column Title' ,  'module'  ),
            'column2'  => __ (  'Title column2' ,  'module'  ),
            ...
        );

    }
}
```

And the views? Eventually we get the Views. These are actually templates that are derived from the name of the action. For the above example, where the module is used library and the driver is requested books for executing the method (action) list , Common Utilities get the template list.php , because it is the one that corresponds to the action list . Easy, right? It follows that the view files (templates) should be appointed as the action (controller method) running. If our action is the name of form , so our staff must be appointed form.php . If the action is the name -form categories , also our file should be named categories-form.php . One more thing. The Views, as in any module, should be stored in the directory templates module, but not directly, but in appropriate subfolder, depending on the following cases. - Templates Folder can contain standard templates (as commonly used in the modules). - Modules can have templates

for the control panel or section templates for public, therefore within the directory /templates there should be two subfolders: backend and frontend . - Within each of these subfolders there should be a new subfolder for each driver that handles Views. If the driver is called categories, then there must be a subfolder called categories where the Views will be kept for each action. This new approach of module development in XOOPS enables faster and more structured development. Furthermore, with its auxiliary objects, Common Utilities facilitates the implementation of AJAX in modules allowing more intuitive and easy to use experience for users.