

Making a News Module

14.12.2008

You might wonder what the word “module” means in this context. A module in CWF is a piece of application that controls or represents one type of entities, like forum topics, news items or users. A module is also a class with methods callable by the user. Such class would extend the class called Module and act as a kind of page controller.

I want to create one of those. The code will be in libs/NewsItems.php. I'll start with the following lines:

```
<?php
class NewsItems extends Module{
}
```

There are no include's or require's in the code (and there rarely will be one, since CWF uses the much dreaded `__autoload` function).

My NewsItems module should at least be able to receive a news items and display them in an orderly fashion. Therefore, I need to add the following methods.

```
<?php
class NewsItems extends Module{
    function create($target, $input){
    }

    function createDo($target, $input){
    }

    function index($target, $input){
    }
}
```

Any method of a CWF module is callable via web. When it is called in such a way, it's referred to as “action”. With the code I just wrote, going to `http://localhost/framework/?NewsItems.create` should invoke the create method. However, if I do that, I will get an error message: “Action 'NewsItems.create' is not permitted”. That's because the framework automatically checks for permissions whenever an action is invoked. Currently, I'm not logged in, so it sees me as the unknown user. Literally. That is, there is a user called “unknown” in the database, and the framework loads its settings for anyone not logged in. To be able to do anything interesting, I would need to create an account for myself:

```
INSERT INTO `user` VALUES (2, 'Admin', 'md5', '', MD5('password'),
'email@example.com', NOW(), '');
```

..and grant it administrative privileges:

```
INSERT INTO `perm` VALUES ('user', 2, '*', '*', '*');
```

After that I can log in via `http://localhost/framework/?Users.login`, and *really* call `http://localhost/framework/?NewsItems.create`, getting a blank page in response. If I call `http://localhost/framework/?NewsItems.create.xyz&k=v&k2=v2`, then the first argument (`$target`) will get the value “xyz”, while the second argument (`$input`) will get the contents of `$_GET`. To be precise, `$input` will be set to the union of `$_POST` and `$_GET` minus the garbage variable created by the part

of the query specifying what to call.

I think the reason we have `$input` argument in each of our methods is self-explanatory, but you might wonder what does the `$target` do. It's there to represent the target of an action, such as id of an article to show, or name of a document to edit. It server two purposes. One, it reduces the need for input validation, since it's only allowed to have alphanumeric values and comas. Two, it allows for transparent security check. This aspect deserves a more in-depth explanation.

The permission system of CWF relies on a set of rules in the 'perm' table. By default, all actions are prohibited, and each rule grants someone a right to perform an action or a set of actions. The first field of perm table ("type") selects between the type of subject for the permission, which can be either user, group or dynamic user category. If it's user or group, the next field ("id") will correspond to id of that user our group, respectively. In case it's set to "dynamic", the id corresponds to either to all users (then it's 1) or all registered users (then it's 2). The rest of the fields are used to specify the module, action and the target of action allowed for the particular subject.

Some examples. `Articles.show.1` would allow the subject to call show methods of the Acticles module, but only if the target is set to 1. `Articles.show.*` would allow to call the same action with any id. `Articles.*.*` would allow to call any action in the module. Finally, `*.*.*` would grant the subject permission to invoke any action or any module.

It's important to note, though, that the permission system of CWF is advisory, not mandatory. Every module has it's own `permitted()` method, that may or may not consult with the database and may or may not use it the way I described above. For example, if I grant someone permission for `NewsItems.create`, the module will also allow them to invoke `NewsItems.createDo`, even though that rule is not in the database. Why? It's a common pattern to have two methods for every HTML form: one that displays it and another that processes the data on submit. The convention is to call the latter with the same name as former, appending "Do" at the end. Therefore, the default `permitted()` method in the Module class treats them as having similar permissions - it cuts off "Do" from method names when it does permission checks.

Okay, let's get back to coding `NewsItems` module. The first thing I will do is to write code for `create()`.

```
<?php
class NewsItems extends Module{
    function create($target, $input){
?>
<h2>Create a News Item</h2>
<form action="?NewsItems.createDo" method="POST">
<span class="label">Title:</span><input type="text" name="title" /><br />
<span class="label">Text:</span><textarea name="text"></textarea><br />
<input type="submit" value="Post It" />
</form>
<?php
    }

    function createDo($target, $input){

    }

    function index($target, $input){
    }
}
```

As you can see, output of the method will be sent back to the user. There is no need to do something special about it - the framework will surround it with HTML boilerplate code by via output buffering. Of course, this is not MVC, but I don't see any point in using templates for an application as simple as this one. If need be, the HTML code can be easily moved to a template afterwards.

The next step - create a table to store news items:

```
CREATE TABLE IF NOT EXISTS `newsitem` (  
  `id` int(11) NOT NULL auto_increment,  
  `title` varchar(200) NOT NULL,  
  `text` text NOT NULL,  
  `time` timestamp NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `time` (`time`)  
);
```

To work with this table, I want to use a record class. The convention is to make record name singular and module name plural forms of the same word. That is not to say that a module will *always* work with only one record. It's just a convenient way to name files in a simple application. The code below will go to `libs/ NewsItem.php`.

```
<?php  
class NewsItem extends Record{  
    protected $attrs = array(  
        'title' => NULL,  
        'text' => NULL,  
        'time' => NULL,  
    );  
}
```

The `$attrs` field holds an array representing table rows, with the exception of `id`. The latter is a special case - every record has an `id` field, and the default name for it is, not surprisingly, "id". That's why I did not need to list it explicitly. The table name is deduced from the class name.

Here is how an instance of this class can be used:

```
<?php  
class NewsItems extends Module{  
    function create($target, $input){  
?>  
<h2>Create a News Item</h2>  
<form action="?NewsItems.createDo" method="POST">  
<span class="label">Title:</span><input type="text" name="title" /><br />  
<span class="label">Text:</span><textarea name="text"></textarea><br />  
<input type="submit" value="Post It" />  
</form>  
<?php  
    }  
  
    function createDo($target, $input){  
        $ni = new NewsItem();  
        $ni->title = $input['title'];  
        $ni->text = $input['text'];  
        $ni->create();  
    }  
}
```

```
        echo 'News item created.';
    }

    function index($target, $input){
        $ni = new NewsItem();
        $items = $ni->selectAll(3);
        foreach ($items as $item) {
            echo "<div>
<h3>$item->title</h3>
<b>$item->time</b><br />
$item->text
</div>";
        }
    }
}
```

The code should be mostly self-explanatory. In the `index()` action, I'm creating a new `NewsItem` object only to call `selectAll` method one line later. This method would make more sense if it was static, but the language limitations of PHP 5.1 don't allow that. You need to create a throwaway object to get the benefits of inheritance. (This is likely to change after PHP 5.3 becomes stable.)

Select methods of a record return arrays of objects of the same type. In this particular case I want to get 3 news items. You might think that the first argument to `selectAll` is a limit, but that's not entirely correct. It's a page size. If I were to provide a second argument, it would act as a page number.

Since this code provides the minimum functionality needed from a simple news application, I'll stop here.