Controllers

Overview

The controller is responsible for responding to user actions. In the case of a web application, a user action is (generally) a page request. The controller will determine what request is being made by the user and respond appropriately by triggering the model to manipulate the data appropriately and passing the model into the view. The controller does not display the data in the model, it only triggers methods in the model which modify the data, and then pass the model into the view which displays the data.

Most components have two controllers: one for the front-end and one for the back-end.

Creating the Front-end Controller

(preferred) HUBzero Method

Most HUBzero component controllers will differ from Joomla! in some important ways. Some changes are made to aid in development while others may simply be a difference in philosophy. Note, however, that no differences require hacking or altering Joomla! in any way and HUBzero methodologies will run on any stock Joomla! install.

```
<?php
// No direct access
defined('_JEXEC') or die('Restricted access');

ximport('Hubzero_Controller');

class HelloControllerOne extends Hubzero_Controller
{
 public function displayTask()
 {
  // Pass the view any data it may need
  $this->view->greeting = 'Hello, World!';

  // Set any errors
  if ($this->this->getError())
  {
   foreach ($this->getErrors() as $error)
   {
    $view->setError($error);
   }
  }

  // Output the HTML
  $this->view->display();
 }
}
```

There doesn't appear to be much going on here due to some of the auto-setup the Hubzero_Controller class brings.

The first, and most important, difference to note is that we're extending Hubzero_Controller rather than JController. Since we're not employing JController, how tasks and views are determined, built, and executed differ from standard Joomla components.

Note: Hubzero_Controller extends JObject, so all its methods and properties are available.

One key difference is how the execute() method is handled. In Joomla!, any public method is assumed to be an executable task. In the HUBzero method, the list of available tasks is built from only methods that are 1) public and 2) end in "Task". When calling a task, the "Task" suffix should be left off. For example:

```
// This route
JRoute::_('index.php?option=com_example&task=other');

// Refers to
....
public function otherTask()
{
 ...
}
....
```

If no task is supplied, the controller will default to a task of "display". The default task can be set in the controller:

```
<?php
// No direct access
defined('_JEXEC') or die('Restricted access');

ximport('Hubzero_Controller');

class HelloControllerOne extends Hubzero_Controller
{
 public function execute()
 {
  // Set the default task
  $this->registerTask('__default', 'mydefault');

  // Set the method to execute for other tasks
  // The following can be called by task=delete and will execute the r
emoveTask method
  $this->registerTask('delete', 'remove');  // (task, method name);

  parent::execute();
 }
 ...
}
```

Each controller extending Hubzero_Controller will have the following properties available:

- _option - String, component name (e.g., com_example)
- _controller - String, controller name
- view - Object (JView)
- config - Object (JParamter), component config
- database - Object (JDatabase)
- juser - Object (JUser)

```
class HelloControllerOne extends Hubzero_Controller
{
 public function displayTask()
 {
  $this->view->userName = $this->juser->get('name');
  $this->view->display();
 }
}
```

Auto-generation of views

The Hubzero_Controller automatically instantiates a new JView object for each task and assigns the component and controller names as properties (for use in your view). Controller names map to view directory and task names directly map to view names.

```
  /{component}
```

```
    /views
      /one  (controller name)
        /tmpl
          /display.php
          /remove.php
```

Joomla! 1.5 Method

Our component only has one task - greet the world. Therefore, the controller will be very simple. No data manipulation is required. All that needs to be done is the appropriate view loaded. We will have only one method in our controller: display(). Most of the required functionality is built into the JController class, so all that we need to do is invoke the JController::display() method.

The code for the base controller com_hello/controller.php is:

```
<?php
// No direct access
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport('joomla.application.component.controller');

/**
 * Hello World Component Controller
 *
 * @package    Joomla.Tutorials
 * @subpackage Components
 */
class HelloController extends JController
{
    /**
     * Method to display the view
     *
     * @access    public
     */
    public function display()
    {
        parent::display();
    }
}
```

The JController constructor will always register a display() task and unless otherwise specified (using the registerDefaultTask() method), it will set it as the default task.

This barebones display() method isn't really even necessary since all it does is invoke the parent constructor. However, it is a good visual clue to indicate what is happening in the controller.

The JController::display() method will determine the name of the view and layout from the request and load that view and set the layout. When you create a menu item for your component, the menu manager will allow the administrator to select the view that they would like the menu link to display and to specify the layout. A view usually refers to a view of a certain set of data (i.e. a list of cars, a list of events, a single car, a single event). A layout is a way that that view is organized.

In our component, we will have a single view called hello, and a single layout (default).