Writing a model for Portal

> This tutorial assumes you have portal and registry already installed correctly and the portal is accessible from http://localhost

In this tutorial, we will try to extend the portal functionality to not only able to view registry object from the API, but able to display our own data model, for the sake of simplicity, we'll use cars. The portal will be able to display metadata of a car based on it's model name, (for eg, going to http ://localhost/wrx should give us metadata for the WRX car

We'll start with creating a `car` directory under `applications/portal` so all of our controllers, models and views can reside in

```
mkdir applications/portal/cars
mkdir application/portal/cars/controllers
mkdir application/portal/cars/models
mkdir application/portal/cars/views
```

We'll start with the controller

```
<?php
class Cars extends MX_Controller {
        function view() {
                echo 'Hello World!';
        }
}
```

Now we need to change the Portal configuration to use this controller for any view, so we'll change the `applications/portal/core/config /config.php` to reflect that

```
$config['default_model'] = 'cars'; //instead of registry_object, this
tells the portal to use the cars model to be the default model
```

Now, by going to http://localhost/wrx you can see Hello World, because the dispatcher `applications/portal/core/controller /dispatcher.php` now recognize `cars@view` as the default controller to view any requests

We'll continue by creating a car model by creating a file called *car.php located a*t `applications/portal/cars/models/_car.php`

```
<?php if (!defined('BASEPATH')) exit('No direct script access allowed');
class _car {
        public $prop;
        function __construct($make, $model) {
                $this->init($make, $model);
        }
        function init($make, $model) {
                $this->prop = array(
                        'make'         => $make,
                        'model' => $model
                );
        }
        /**
         * Magic function to get an attribute, returns property within the
$prop array
         * @param  string $property property name
         * @return property result
         */
        public function __get($property) {
                if(isset($this->prop[$property])) {
                        return $this->prop[$property];
                } else return false;
        }
        /**
         * Magic function to set an attribute
         * @param string $property property name
         * @param string $value    property value
         */
        public function __set($property, $value) {
                $this->prop[$property] = $value;
        }
}
```

This is the base model for the car, we now need a way to create new cars, we'll have another model called `cars_factory` and will be in charge of creating new `_car`

```
<?php
class Cars_factory extends CI_Model {
        function newCar($make, $model) {
                return new _car($make, $model);
        }
        function __construct() {
                parent::__construct();
                include_once("_car.php");
        }
}
```

So now, if we go back to the cars controller, we can create new a new car with

```
$this->load->model('cars_factory');
$car = $this->cars_factory->newCar('Subaru', 'WRX');
```

Now, we need to handle requests coming in from the server, http://localhost/wrx and http://localhost/jazz should construct 2 different `_car` model and be ready to view.

> If there is only 1 parameter passed in to the dispatcher, the request is seen as $_GET['any'], if there are 2 parameter, the request is seen as $_GET['slug'] and $_GET['id']. This can be changed in the dispatcher if need be. For now we'll regard our 2 requests as being $_GET['any']

We'll continue with our cars controller to pass in the request and construct the car model, and then pass a view back to the client like so

```
<?php
class Cars extends MX_Controller {
        function view() {
                $this->load->model('cars_factory');
                $model = $this->input->get('any');
                $car = $this->cars_factory->getCar($model);
                $data['car'] = $car;
                $this->load->view('car', $data);
        }
}
```

And we add the `cars_factory@getCar` function along with a `cars_factory@blueprint` function to generate more metadata for the request like so

```
<?php
class Cars_factory extends CI_Model {
        function newCar($make, $model) {
                return new _car($make, $model);
        }
        function getCar($model) {
                $blueprint = $this->blueprint($model);
                $car = new _car($model, $blueprint['make']);
                $car->cylinder = $blueprint['cylinder'];
                $car->engine = $blueprint['engine'];
                return $car;
        }
        function blueprint($model) {
                $blueprints = array(
                        'jazz' => array(
                                'make' => 'Honda',
                                'model' => 'Jazz',
                                'cylinder' => 4,
                                'engine' => '4cyl 1.5L'
                        ),
                        'wrx' => array(
                                'make' => 'Subaru',
                                'model' => 'WRX',
                                'cylinder' => 4,
                                'engine' => '4cyl 2.5L'
                        )
                );
                return $blueprints[$model];
        }
        function __construct() {
                parent::__construct();
                include_once("_car.php");
        }
}
```

Lastly, we'll generate a view to display a single car with the file car.php located at `applications/portal/cars/views/car.php`

```
<h1><?php echo $car->make.', '.$car->model; ?></h1>
<dl>
        <dt>Make</dt><dd><?php echo $car->make; ?></dd>
        <dt>Model</dt><dd><?php echo $car->model; ?></dd>
        <dt>Cylinders</dt><dd><?php echo $car->cylinder; ?></dd>
        <dt>Engine</dt><dd><?php echo $car->engine; ?></dd>
</dl>
```

Now, if we go to http://localhost/wrx we should see the following HTML rendered correctly

```
<h1>wrx, Subaru</h1>
<dl>
        <dt>Make</dt><dd>wrx</dd>
        <dt>Model</dt><dd>Subaru</dd>
        <dt>Cylinders</dt><dd>4</dd>
        <dt>Engine</dt><dd>4cyl 2.5L</dd>
</dl>
```

And the response for http://localhost/jazz should be

```
<h1>jazz, Honda</h1>
<dl>
        <dt>Make</dt><dd>jazz</dd>
        <dt>Model</dt><dd>Honda</dd>
        <dt>Cylinders</dt><dd>4</dd>
        <dt>Engine</dt><dd>4cyl 1.5L</dd>
</dl>
```