

## Where to Start with Yii?

If you're like me, then you came to the Yii framework world with enough web coding experience to take scripts and manipulate them to your needs or even build your own applications from scratch to customize the features you want a site to have. The short story is: that you understand how to program in PHP.

Now I realize a lot of people who get into framework coding actually never learned (from a technical class at least) the art of OOP, heck majority of PHP coders didn't learn the language from a class either. Myself, I had the opportunity within school to learn OOP with C++, that being said it may be a different language but the concepts are still the same. This is a huge help in working with an OOP framework (go figure?). Now I'm not here to teach OOP, there are plenty of articles out there that have a great sense of pointing you in the right direction with learning OOP. So if you need to catch up on it or haven't a clue what I'm talking about I suggest you search about it because it's the only way to truly understand how a framework works.

However if you are like me, then this is your first time ever (period) working with a framework. This means the concepts of how the Yii framework works (or any framework for that matter) is oblivious to you. Sure everyone who can code can read code, and sure we can all dive into the framework; read about the documentation, all the functions and classes and the whole framework inside out (which I did) – but understanding how to build on it, well that just takes practice and experience.

Ok so practice then right? Practice builds experience, seems like a simple solution. Well that's easier said than done, because in order to practice anything right, you have to be taught first (either by self learning or someone showing you). Well not to discredit any of the supporters of the community but there really isn't any well documented "self learning" or "someone teaching you" articles that show you where to start and how in Yii, or many other frameworks for that matter. I'm talking about actually building your application in mind set and then showing you where to plug in your logic.

Don't get me wrong, the Yii documentation is well-established, with lots of reference to what classes are used, inherited and how they are used, everything. Yii is actually very well documented for anyone who knows how to use a framework and OOP, picking up the works of Yii is easy to them. But I wanted to learn where to actually start building my application. I have a site that I want to implement with the Yii framework but I could never figure out where to start.

This was a fork in my road to learning Yii, I wanted to do more than I could at this time. Reading documentation is great, it teaches you a lot about what you're using, but no one really wants to sit here and read a novel – let's face it we all want to get our hands dirty and start something now, we're impatient. That's the reason why we use frameworks, to make coding faster, to get the results in a more reliable, reusable and quicker fashion.

So this little guide is here to show you where to start. In my efforts to find a way to start coding with a framework, I came across a site, well a blog that the author was talking right down my alley. His article basically said, you're new to frameworks, everywhere you go tries to talk about which framework to use or why and how frameworks work but no one really tells you how to build on them. I was like YES! That's what I'm talking about. So he listed some simple solutions. His answer and when you hear it you'll be smacking your head on the wall, was to think small. Build simple (so we think) applications that as a coder you can do on your own from scratch, maybe even in multiple languages. But building it within the framework – that's how you learn to use the framework.

Here is the article if you want the full list of his suggestions, as well there's another user named Stuart who posts a comment about some more framework specific tasks that are great to move on to in continuing your learning of the framework. <http://www.onekay.com/blog/archives/35>

For the sake of keeping it simple, I'm only going to cover his first two suggestions: his first simple application (Hello World) in this tutorial and the 2<sup>nd</sup> (Calculator) in another tutorial. If you see by example those ideas, well then it becomes a little easier to actually accomplish the more in-depth tasks, and eventually build your sites.

Our first task will be to do the dreaded "Hello World" application that does nothing but print out that string. Every time we see it, we think this is so useless (at least I always did in my programming classes). Once I learned hello world in one language, moving to another language was a breeze, to do the same statement (over and over) was a joke, that I just simply hated the string "Hello World". But thinking now how to implement it with a framework, well let's just say it's not as easy as it seems, and I appreciate this statement a little more today.

Of course we could cheat; we could find some really simplistic way of printing out hello world with the views and layout, the way we want it to and all. The problem there is nothing is gained. You didn't learn how to use the framework to do your application, you just modified a script. So what do I mean then with printing out "Hello World"? Isn't that just something that simple, print it out and done?

I mean let's consider the Yii framework and how it works. It's an MVC design, which stands for Model View Controller. What that means to people who don't have a clue, well I'll tell you a bit about it but the guide made for Yii will explain these concepts well in depth for you and I would highly recommend you read it (at least the fundamentals) before you get started. Here is the guide link: <http://www.yiiframework.com/doc/guide/index>

For the sake of covering it however here is a little description. Think of your application as 2 parts. If you've ever coded a site from scratch – the design and the logic, then you might understand this concept of separating php from say css or html. An MVC design separates business logic from the presentation logic. This means that the code you used to build a form to display in html (presentation) is separated from the code you use to evaluate it (business).

Why? Why separate it and have to modify two files when you want to add an input field and thus a value to evaluate? Because let's say you wanted to modify the code, either how the form is presented or

how you transfer the data. You don't want to mess with one and inadvertently screw with the other, which can happen if all of it is built together in the same script. Sometimes you unknowingly build a script that ties both sides together and as you try to modify one, you have to accommodate for the other. Not to mention, the php files just become bulky with 2, 3 even 4 different languages coded in it.

Here is a simple way to think of this dilemma, it's like coding CSS for Firefox, and then realizing the same code doesn't work for Internet Explorer. It's a pain to have to try and come up with a solution that works for both browsers (and more if you're picky). That's why an MVC model separates the logic of the presentation from the business. Imagine if in CSS you could code it right on the spot one for Firefox and another for Internet Explorer right in the same CSS document, without having to rename your classes or IDs. That would be a perfect harmony world (so dream on). What would be better is if they all just worked the same. Now that's a perfect world. :/

Ok, so then how does the MVC model work, now that we understand why it's used? Each word of MVC represents how the design functions. Models are considered data, it's what holds and handles data. Think of a model like a giant attribute with several attributes forming it. In OOP attributes are what we use to define variables pertaining to that class. So the easy way to think of this is that a model is a giant variable that holds specific variables forming the details of that giant variable. The best part though? A model is actually a class, so you can run functions to handle that giant variable within the variable itself as well as all the other detail variables.

Here's an example to help understand, let's call our model User. So User is the entity (person, place or thing) we are manipulating. The Model (giant variable) is a class that has details built within it defining that Model. So in our case, User is a Model class that has details about it within its class, these are the smaller variables I referenced earlier. An example, our User Model will have a name variable, password variable, maybe an email variable. These are all smaller attributes defined within the model that define the model as a whole. So you think of the User Model that has a User name, User password, User email.

The next step is a Controller. A controller is what performs the actions on a Model. In OOP something that performs an action is called a method. The simple way of thinking of a method is a function; a method is simply a function. In Yii, a method defines functions within a controller manipulating an item (a Model). With Yii you'll see a lot of reference to actions, actions are simply public methods – what the user can do, where as helper methods are private or protected functions that are passed through inheritance classes only. When you see class "... extends C..." that's a class being defined that inherits properties from the parent class "C...". That means all the helper methods are available to that class to manipulate the model associated with the controller. There are more details to this in the Yii Guide.

Finally, there's View. View represents how the Controller displays the Model. The Controller manipulates the Model with methods and as such it passes the manipulated Model to the View with Methods and the View will then display it properly. With this guide however we will not be focusing much on view as it's more important to understand the business logic behind the framework than how

to manipulate the presentation logic of it. Plus you should try to avoid doing business logic within the presentation logic, just to keep things simple and separated.

So now that we have that concept under wraps we can go back to our original tasks, building a hello world application. If you don't see it now, don't kill yourself, it took me a minute to establish the "right" way of doing this, but for our application we will build a model that gets manipulated by a controller. You probably don't see this as very effective coding because that's a whole lot of waste of code and time to just simply do echo "Hello World". However by building a model to hold the entity (person, place or thing), in our case the string, we can use a controller to manipulate it (in this case display it). This is stupid for something so simple, but then you have to think that your real world applications won't be so simple and this is merely to understand how to use the framework.

Now because it seemed rather dull and stupid, I also decided to add an edit action. So while our model holds the string data, it will also hold rules to change it and the controller will not only be able to show the string but it will have an action to edit it. This just adds a little more so the simple application doesn't seem so stupid and simple.

So where to start? Well let's recap, we need a model and we need a controller. Now if you've read up on the fundamentals guide to Yii you'll understand how the naming convention works. I'm not going to go into detail about it, so if you have questions, refer to the Yii Guide or Forums.

At this point, I haven't even thought about starting to code yet, this is all written on paper for me so I can collect my logic, understand what I'm building, how and what I need to make it work. So I define a controller and a model.

MessageController.php and Message.php

MessageController has 2 actions

Action: Show message

Action: Edit message

Message defines

Variable: message

Variable: author

Rules: When action edit is called, form fields message and author are required.

This is very simple logic but it's effective and fundamental to building a proper application with Yii.

So now let's get started with some coding. If you haven't done the first application tutorial yet showing you how to use yii I would recommend it but I'm still going to show the commands on how to operate it.

First things first these are the assumptions I'm making. (Since majority of the time you can only operate the command console on your own personal computer.) You have a local folder that is your "web root"

for your “web server” in my case mines “htdocs”. Within this local root folder, you should have the yii application uploaded; this folder contains the yii framework. I will for reference call this folder yii. So here’s a small directory view of what our folder looks like:

```
Root
  yii
    framwork
```

Ok so within the root folder, we’ll be making a template web application using yiic. Open up a command prompt window (for Windows users) or terminal (for Linux/Unix). Here are some commands that you’ll want to know.

dir → lists directory within Windows

ls → lists directory within Linux

cd → changes the directory in both operating systems.

What’s important to know is that Windows is lenient with its directory changing. If you’re folder is called My Folder, typing my folder will be just fine. Linux on the other hand is case sensitive and if you have more than one word separate by a space you have to use quotations(“”) to wrap the directory. Example, a folder called My Folder would not be found with My Folder or my folder, you have to type “My Folder”. Hope that clears anything up for complete new users.

Another command to know, “../” this changes to the previous directory (a directory above the one you’re in). Navigate yourself to your root folder, the best way to do this is to go c:/whatever folder you’re in/ :> cd ../

Do that several times until you reach c:> as the directory you’re in, then using dir or ls and cd navigate to your root folder.

For me: c:/htdocs/:>

The reason why I do this is so that when I call the yiic program and get it to build an application, I don’t have to specific a root, it will simply build it within my htdocs folder ( I ran into issues before with this from spaces, so I just decided to make it simple and install right in my root folder). So let’s build a template, we’ll call it test

C:/htdocs/:> ./yii/framework/yiic webapp test

./ specifies the folder I am in, and references to access the folders within it, so within my root folder I access the yii folder and then it’s framework folder in order to run yiic. Webapp is the command to build the template, and it builds it to a folder called test, located c:/htdocs/test/.

Ok now navigate yourself to your test folder, for me: C:/htdocs/test/:> we’ll type in the next command to access the console within our web template. Type” ./protected/yiic shell”. This should

open up the shell menu where you can build templates of models and controllers (what we are going to do) among other things.

So let's build our model and controller, if you're confused on how to do this, type help within the shell to get the specifics. For the controller, add the additional actions of show and edit. Once completed, you can see where all the files were created and you can even use the test link to view your web application template in your web server. I used Message as my ID for everything but feel free to put String or HelloWorld or whatever suits you. With your Model, type Form after ID, so it looks like YourIDForm.

Now in your favourite coding software, open up the following files:

```
protected/controllers/YourIDController.php
protected/models/YourIDForm.php
protected/views/YourID/edit.php
protected/views/YourID/index.php
protected/views/YourID/show.php
```

So let's start with our Model. First things first, since we aren't using a database, we won't need active records thus we will swap the parent class. Swap extends CActiveRecords to extends CFormModel (that's why we called it YourIDForm). We don't need table name, the static function model, relational rules or the attribute labels so you can delete all that premade code. Our model looks so sweet and simple right now, but rather empty so let's fix that. Remembering from our logic we need to define the variables and set the rules, so let's do that.

Make two public variables for the string and the author, I'll call mine \$message and \$author, you can call yours whatever you like. To finish up the model we need to build our rules. Now if you read the Yii guide you'll know all about the validation functions that Yii has to offer and the features specific to them. For simplicity, I will paste out all you need to put in but feel free to read and manipulate this for your own applications. If you would like a little more of a detailed look at a model as a guide check out protected/models/ContactForm.php

Between return array( and ); of rules() make a new line and write this (replacing the names for your two strings of course):

```
// author and message are required
array('author, message', 'required'),
```

This will define the rules for when you go to edit the message, it states that the author and the message variables (input fields from the presentation) will be required and will not process unless they are filled.

And that's it; we don't need to worry about our model anymore, but don't forget to document what you're doing and why you're doing it, makes it easier to understand later. So now let's look at our controller. Opening up the controller you will see a lot of methods that probably mean nothing to you.

Since we aren't defining any filters, and we aren't overriding the action we will delete that commented code. That leaves us with our `actionShow`, `actionEdit` and `actionIndex`. Now `actionIndex` is the default action, when we go to the message routed request, that action will happen unless we specify a different action (show or edit). We'll make this page a setup to choose whether to show the string or edit the string. Once again that's quite unnecessary but we are showing how the framework works, normally you would have the index action show the string and then have a link to edit it.

Let's work with show first. Now with show, we want to evaluate one simple thing, is the message set or not. If there is no message set, then we make the default Hello World, if there is then we display our edited message. Here's my code that I would put in the show action, you'll have to adjust yours for your logic

```
$message=new MessageForm;
if(isset($message->message))
    $this->render('show',array('message'=>$message));
else
{
    $message->message = "Hello World";
    $message->author = "Default Author";
    $this->render('show',array('message'=>$message));
}
```

That was really simple. We first initiate a new instance of the model class, my model is Message thus I make a new instance of message pointing to my MessageForm Model. From that point on, I can access all attributes (like our string for the message and author) and methods (like `rules()`) with `$message->attribute` or `$message->method()`. By rendering the show view and passing it the message instance, within the view file `show.php` we can access those attributes and methods as well from `$message`.

Ok so we have our show action set up to render, however we have not set up how the view renders it. Since we'll have to do a render for `index.php` (the default action), `show.php` and `edit.php` we'll do that after we finish up our edit action. After we'll also add a main menu link so we can access our string page.

The edit action will be a little trickier. Now I based mine off the template of the contact form. All I did was modify the script to match my settings for my form and customized what actions would happen on the true or false statements. Once again we need to initiate an instance of our MessageForm thus we open up with the same `$message = new MessageForm;`. However, we are going to evaluate the `isset` function with the values of the form, and if it has input.

```
$message=new MessageForm;
if(isset($_POST['MessageForm']))
{
    $message->attributes=$_POST['MessageForm'];
    if($message->validate())
        $this->render('show',array('message'=>$message));
    else
        $this->render('edit', array('message'=>$message));
}
else
    $this->render('edit', array('message'=>$message));
```

Now there's my code for the edit function. It's quite basic, nothing fancy. The first if statement checks to see if the form was submitted with values. The next statement assigns the values from the form to our \$message instance. Then we evaluate if the values passed in the form and now stored within the attributes of the \$message instance are valid, meaning did the user actually input something into the textboxes. Remember from our model, we made the rules function where the author and string were required; this is where that comes into play.

So if everything goes well, the framework will display your show page, with the updated message. If not, then it shows the edit page marking the error inputs (whichever the user did not input). Finally, there is a default action that if the form wasn't sent and thus values aren't set, we render the form to allow the user to edit the string by inputting values for the author and the string. It's about as simple as you can get. Now we just need to create the render.

The render was a little tricky at first to understand how it works; but once again using the contact.php view file and the login.php view file made by the template I modified how I wanted my view files of index.php, show.php and edit.php within the /protected/views/message/ folder to look like. I made a simple html paragraph setup for my index page, linking to the show and the edit actions. Here:

```
<h1>Welcome</h1>
<p> To view our message go to <?php echo CHtml::link('Here',
'./index.php?r=message/show')?> </p>
<p> To edit our message go to <?php echo CHtml::link('Here',
'./index.php?r=message/edit')?> </p>
```

Very simple and my show page is pretty much the same idea, except I call reference to the \$message->attributes to display my author and message attributes. Take a look:

```
<?php $this->pageTitle=Yii::app()->name . ' - Show Message'; ?>
<p><?php echo CHtml::link('Back', './index.php?r=message')?> </p>
<h1>Our Message</h1>

<p>
<?php echo $message->author; ?>
- "
<?php echo $message->message; ?>
"
</p>
```

The php lines that say echo \$message->attribute will display the respective attribute on the page. Now to add a little style, I put the author string first, then a dash followed by wrapping the message string within quotations. This just makes the display a little classy, almost like a quote. I hope as you're customizing your render pages, you take the time to see what the pages look like. This will help you better understand what changes affect the render and how. Views aren't important until you go to style your website, however it would be a heck of a shame to go through all the trouble to code your site and then not know how or be able to customize the view of it. So remember to make every tampering experience with simple applications using Yii, a learning experience.

The final page to render would be edit. Now this page is a little trickier than the previous two, because we have to implement a form. Once again however, I used the template of the contact.php view file to simulate my form and just replaced the proper values. Here's my edit render:

```
<?php $this->pageTitle=Yii::app()->name . ' - Edit Message'; ?>
<p><?php echo CHtml::link('Back', './index.php?r=message')?> </p>

<h1>Edit Our Message</h1>

<p>
<div class="yiiForm">
<?php echo CHtml::beginForm(); ?>
<?php echo CHtml::errorSummary($message); ?>
<div class="simple">
<?php echo CHtml::activeLabel($message, 'author'); ?>
<?php echo CHtml::activeTextField($message, 'author'); ?>
</div>
<div class="simple">
<?php echo CHtml::activeLabel($message, 'message'); ?>
<?php echo CHtml::activeTextField($message, 'message'); ?>
</div>
<div class="action">
<?php echo CHtml::submitButton('Submit'); ?>
</div>
<?php echo CHtml::endForm(); ?>
</div><!-- yiiForm -->
</p>
```

Now that's it. You'll see there's not much to it. I added a link to go back to the index page of our message controller on both the show and the edit page. Even though we could just click the main menu link we are about to build, it's still nice to understand how to implement a "go back / go forward" link on your pages.

So now to add our final feature and that's to make a menu button on our main page so that we can access our message controller and its new actions. (So we can print our "Hello World".) You're probably thinking, oh boy, this is going to be insane, we have to define the link and make it accessible to all the pages and blah blah blah. No. Yii does all this work for you. All you have to do, is change 1 line, actually add 1 line to be more process. Navigate yourself to the protected/views/layout/ folder and open up the main.php. This is the main display of the template file. It shows the header, the content, the footer, the main menu – everything. It's basically our html page with template features to just pop in what we need wherever we want to show it. In this case, the template is already set up to pop in our content (from our message controller) all we have to do is point to it. So you need to find:

```
<?php $this->widget('application.components.MainMenu', array(
    'items'=>array(
        array('label'=>'Home', 'url'=>array('/site/index')),
        array('label'=>'Contact', 'url'=>array('/site/contact')),
        array('label'=>'Login', 'url'=>array('/site/login'),
            'visible'=>Yii::app()->user->isGuest),
        array('label'=>'Logout', 'url'=>array('/site/logout'),
            'visible'=>!Yii::app()->user->isGuest)
```

```
),  
)); ?>
```

This is the main menu with all the included buttons from the template. All we have to do is input a new link. Here's my link, you can customize it to your titles and locations. I put it between the contact and login buttons but feel free to place it anywhere you like within the menu.

```
array('label'=>'Our Message', 'url'=>array('/message/index')),
```

And that's it. Try out your "Hello World" application that implements the Yii controller –model – view concepts. I hope you enjoyed the tutorial and learn a great deal of how Yii functions as well as how to start with building on to the Yii framework. If you'd like to continue to the next tutorial check out my 2<sup>nd</sup> edition of "Where to Start with Yii – Project 2".

Some final words of advice, my first programming teacher told us that there were 3 types of programmers in this world: Those that Know, those that Understand and those that are the Fly in the Coke Bottle. Those that know have years of experience with their language and are the professionals in their field, they code with their eyes closed. Those that understand grasp the logic and can build it, code fairly well on their own but research functions and information they need to complete their projects.

So what's the Fly in the Coke bottle then? Every programmer at some point in their life is or was a fly in the coke bottle. It represents a problem where you are the fly who encounters the issue (fly into the coke bottle) and begin to try a whack of ideas to find something that works as a solution because you don't know how to find a solution. Basically, you keep buzzing around bouncing on the sides of the bottle until you find the opening again (metaphorically: the solution). You never want to be the Fly in the Coke Bottle and it's a bad way to program, because it leads to messy and generally bad coding or lack of understand why what you did worked as opposed to why what you were doing didn't.

However turning those experiences into a programming know why/how, the next time around you'll know where to start so you can improve the solution. Just remember, every solution can be improved, unless of course it's echo "Hellow World";.

Thanks and happy coding.