Understanding the Basics of Zend Framework

The Zend Framework

A couple of years ago, PHP sat at the top of the powerful-but-easy-to-use scripting languages heap — at least as far as popularity was concerned. It was installed on most UNIX®- and Linux®-based Web servers. And if you were a programmer, it was easy to get a hosting account that would let you use it. Ruby had been around for quite some time, but not many people were using it. If you wanted to build a Web site using dynamically generated content, but you weren't sure that you needed to go so far as to use an application server like J2EE, you would very likely use PHP. It was fast, easy to learn, convenient, and you didn't have to learn Perl.

And then — suddenly, it seemed — the landscape changed. Ruby on Rails hit the programming world like a truck. Object-oriented and based on the Model-View-Controller (MVC) paradigm, Ruby on Rails presented a way to do what we all want to do: create a Web site with virtually no effort. Of course, there were still two problems. For one thing, you had to learn a new programming language. That's not a trivial task, no matter what the language. And for another thing, if you found a host that would let you run Ruby on Rails, you were very lucky. Most wouldn't. If you've had the same account for a decade (as I have), you might be a little slow to switch just because they don't have a new programming language. Then, of course, there was the issue of all of the existing PHP code you've written over the years. Did you really want a ditch it all and start over? Of course not!

What is an enterprising PHP programmer to do? Create a new framework that incorporates many of these new advantages, that's what. And, thus, the Zend Framework was born.

The Zend Framework provides clean, stable code, complete with — and perhaps most importantly — clean intellectual property rights. PHP is gaining ground in the enterprise space, but if you're a Fortune 500 company, you don't want to take a chance on a module submitted to a repository that may or may not be some other company's intellectual property.

What is the Zend Framework, exactly? The Zend Framework:
- Is based on PHP
- Is object-oriented
- Uses the MVC paradigm
- Has open source contributors
- Has contributors who take responsibility for the fact that their code is not the intellectual property of someone else

It also aims to make your programming life easier, not just in general, by instituting the MVC pattern, but also for specific things you tend to do all the time, like access databases or output to a PDF file. (OK — you probably don't output to a PDF file all the time. But I'll bet you would if it were easier.)

Zend Framework components include: Zend_Controller This module provides the overall control for the application. It translates requests into specific actions and makes sure they get executed. Zend_Db This is based on PHP Data Objects (PDO) and provides access to databases in a generic way. Zend_Feed This makes it easy to consume RSS and Atom feeds. Zend_Filter This provides string-filtering functions, such as isEmail() and getAlpha(). Zend_InputFilter To Zend_Filter, this is designed to work with arrays such as form inputs. Zend_HttpClient This enables you perform HTTP requests easily. Zend_Json This enables you to easily translate PHP objects into JavaScript Object Notation, and vice-versa. Zend_Log This provides general-purpose logging functionality. Zend_Mail This enables you to send text and multipart MIME e-mail. Zend_Mime This is used by Zend_Mail to help decode MIME messages. Zend_Pdf This enables you to create new PDF documents, and load and edit existing PDF documents. Zend_Search This enables you to perform sophisticated searches on your own text. For example, you can build a search engine that returns results based on relevancy or other factors. Zend_Service_Amazon, Zend_Service_Flickr, and Zend_Service_Yahoo These provide easy access to these Web service APIs. Zend_View This handles the "view" portion of the MVC pattern. Zend_XmlRpc This enables you to easily create an XML-RPC client. (Server capabilities are planned for the future.)

Now let's take a look at where we're going and what we're going to do.Setting up

The Zend Framework doesn't require any particular installation, but you need to keep some requirements in mind. The Zend Framework requires PHP V5. It is compatible with V5.0.4 and above, so you can use V5.1, but you don't have to. You must, however, make sure that the library directory, where the framework expects to find all of its files, is included in

the include_path. To do that, be sure to set it in the php.ini file, as in: ; Windows: "\path1;\path2" include_path = ".;c:\php\includes;e:\sw\zendframework\library"

That's it.

The MVC pattern

When it comes to graphic design, I cannot draw my way out of a wet paper bag with lighted exit signs. It's just not something I do well. However, give me a design to put together by somebody with talent, and I can make that interface do anything you want. So it's very helpful to be able to break out the work involved in designing a Web-based application so designers can do what they do well, coders can do what they do well, and DBAs can do what they do well. That is the essence of the MVC pattern, which breaks a project down into three tiers. Defining Model-View-Controller

Patterns are, by definition, ways of doing things that have evolved over years because they're a good solution to a problem. The Model-View-Controller (MVC) pattern is no exception. Because of this other organic means of creation, you will occasionally find differences in definitions. For example, some definitions of the MVC pattern define the controller's role as simply to change state, with all of the business logic in the model tier. In the end, the only thing that's important is that the application does what you want it to do most efficiently.

The model tier consists of the representation of the actual data. For example, in our feed-reader project, we have users, feeds, and feed entries. Their representation in the database conceptually "models" their structure and, thus, consists of the model tier.

The view tier consists of the logic that actually defines how the displayed data looks. It doesn't decide what the displayed data is — just how it looks. In an ideal world, this template contains no logic. It simply takes the information it's given and displays it.

The controller is what actually defines what the data is. The controller, in fact, is where all of the logic resides. In the Zend Framework, this tier controls the actions to be executed. For example, if I want to display a single feed-item object, the responsibility is parsed like this: The feed-item object has a controller, which defines what happens when the display action is called. That action calls back to get the desired data from the model (in other words, the database or other persistent store), then feeds those fields — such as title, content, permalink, and so on — to the view, which simply displays it in the browser.

Granted, this is a departure for most PHP programmers. PHP has always been a procedural language, despite the presence of objects. Nevertheless, the trade-offs are definitely worth it. Building to the MVC pattern is much easier than building an application in which everything is mashed together.

Now let's look at what's expected of you.

Coding guidelines

When you are coding for the Zend Framework, or even with the Zend Framework, it is expected that you will follow certain guidelines. These are designed to make group projects easier. In other words, by defining coding conventions, you will not only avoid problems later but you will also make it easier for others to read your code. The Zend Framework documentation includes several pages of guidelines, including:

- Make sure the files are clean. In other words, no leading or trailing spaces that can call a Web server to unexpectedly send content before headers, standard indents of four spaces, etc.
- Start your class name with Zend_ if and only if it is intended to be part of the Zend Framework itself, rather than just an application that uses the framework.
- Underscores are forbidden in function names. Use camel-style lowercase (as in getTodaysDate()), instead.
- Start your variable names with an underscore only if they are private or protected.
- Declare all variables as private, protected, or public. Don't use var.
- Use the standard PHP tag, as in <?php ?> — not the short form (<? ?>).
- Make sure your code is easy to read. In other words, when you use a period (.) to concatenate text, be sure to put spaces before and after the period to make it more readable. The same holds for adding spaces after commas when declaring an array.
- If you must pass-by reference, do it only in the declaration of a function. Call-time pass-by reference is prohibited.
- Every PHP file must include documentation that can be read by PhpDocumentor, and the coding guidelines specify

certain minimum tags.

This is not a complete list of guidelines, of course, but it should give you an idea of the types of requirements. Check the documentation for the full list, so your code more easily fulfills the promise of making PHP projects easier to share.