FEATURES

The Object Oriented Evolution of PHP

By Zeev Suraski

One of the key ingredients in the upcoming version 5 of PHP will be the Zend Engine 2.0, with support for a brand new object-oriented programming model. This article describes the evolution of the object-oriented programming support in PHP, covering the new features and changes that are scheduled for PHP 5.

Where did it all start?

Few people know this, but when PHP as we know it today was being molded, back in the summer of 1997, there were no plans for it to have any object-oriented capabilities. Andi Gutmans and I were working to create a powerful, robust and efficient Web language loosely based on the PHP/FI 2.0 and C syntax. As a matter of fact, we got pretty far without having any notion of classes or objects – it was to be a purely structured language. One of these summer nights however, on August 27th that year, this changed.

At the time classes were introduced to the code base of what was to become PHP 3.0, they were added as syntactic sugar for accessing collections. PHP already had the notion of associative arrays collections, and the new critters were nothing but a neat new way of accessing them. However, as time has proven, this new syntax proved to have a much more far-reaching effect on PHP than originally intended.

Another thing that most people don't know is that by the time PHP 3.0 came out officially in mid-1998 and was gaining momentum at a staggering rate, Andi Gutmans and I were already determined to rewrite the language implementation. Users may have liked PHP as it existed at the time - in fact we know they liked it. But as the authors of the engine we knew what was going on under the hood and we couldn't live peacefully with that. The rewrite, which was later dubbed the 'Zend Engine' (Zend being a combination of ***Ze***ev and A***nd***i), initiated and became one of the core components of the 2nd revolution that PHP experienced in just over a year.

This revolution, however, left PHP's object model mostly unchanged from version 3 – it was still very simple. Objects were still very much syntactic sugar for associative arrays, and didn't offer users too many additional features.

Objects in the old days

So, what ***could*** one do with objects back in the days of PHP 3.0 or even with the current version of PHP 4.0? Not that much, really. Objects were essentially containers of properties, like associative arrays. The biggest difference was that objects had to belong to a **class**. Classes, as in other languages, contained a collection of properties and methods (functions), and objects could be instantiated from them using the **new** operator. Single inheritance was supported, allowing users to

extend (or specialize) the scope of an existing class without having to write it from scratch or copy it. Finally, PHP 4.0 also added the ability to call methods of a specific class, both from within and outside object contexts.

One of the biggest twists in PHP's history was the fact that despite the very limited functionality, and despite a host of problems and limitations, object oriented programming in PHP thrived and became the most popular paradigm for the growing numbers of off-the-shelf PHP applications. This trend, which was mostly unexpected, caught PHP in a sub-optimal situation. The fact that objects were not behaving like objects in other OO languages, and were instead behaving like associating arrays was beginning to show.

The limitations of the old Object Model

The most problematic aspects of the PHP 3 / PHP 4 object model was the fact that objects were passed around by value, and not by reference. What does that mean?

Let's say you have a simple, somewhat useless function, called **myFunction()**:

```
function myFunction($arg)
{
$arg = 5;
}

And you call this function:
$myArgument = 7;
myFunction($myArgument);
print $myArgument;
```

As you probably know, the call to **myFunction()** will leave **$myArgument** unchanged; Sent to **myFunction()** is a *copy* of **$myargument**'s *value*, and not **$myargument** itself. This type of argument passing is called *passing arguments by value*. Passing arguments by reference is done by most structured languages and is extremely useful, as it allows you to write your functions or call other people's functions without worrying about side effects they may have on variables outside their scope.

However, consider the following example:

```
function wed($bride, $groom)
{
    if ($bride->setHusband($groom)
&& $groom->setWife($bride)) {
    return true;
} else {
        return false;
}
}
wed($joanne, $joe);
print areMarried($joanne, $joe);
```

(The implementation of **Woman::setHusband()**, **Man::setWife()** and **areMarried()** is left as an exercise for the reader).

What will **areMarried()** return? We would hope that the two newlyweds would manage to stay married at least until the following line of code, but as you may have guessed – they wouldn't. **areMarried()** will confirm that they got divorced just as soon as they got married. Why?

The reason is simple. Because objects in PHP 3.0 and 4.0 are not 'special', and behave like any other kind of variable, when you pass **$joanne** and **$joe** to **wed()**, you don't really pass them. Instead, you pass clones or replicas of them. So, while their clones end up being married inside **wed()**, the real **$joe** and **$joanne** remain within a safe distance from the sacrament of holy matrimony, in their protected outer-scope.

Of course, PHP 3 and 4 did give you an option to force your variables to be passed *by reference*, consequently allowing functions to change the arguments that were passed to them in the outer scope. If we defined **wed()**'s prototype like this:

```
function wed(&$bride, &$groom)
```

then Joanne and Joe would have had better luck (or not, depending on your point of view).

However, it gets more complicated than that. For instance, what if you want to return an object from a function, by reference? What if you want to make modifications to **$this** inside the constructor, without worrying about what may happen when it gets copied back from ***new***'s result into the container variable? Don't know what I'm talking about? Say hallelujah.

While PHP 3 and 4 did address these problems to a certain extent by providing syntactic hacks to pass around objects by reference, they never addressed the core of the problem:

> *Objects and other types of values are not created equal, therefore,* ***Objects should be passed around by reference unless stated otherwise.***

The Answer – Zend Engine 2

When we were finally convinced that objects are indeed special creatures and deserve their own distinct behavior, it was only the first step. We had to come up with a way of doing this without interfering with the rest of the semantics of PHP, and preferably, without having to rewrite the whole of PHP itself. Luckily, the solution came in the form of a big light bulb that emerged above Andi Gutmans' head just over a year

ago. His idea was to replace objects with *object handles*. The object handles would essentially be numbers, indices in a global object table. Much like any other kind of variables, they will be passed and returned by value. Thanks to this new level of indirection we will now be moving around *handles* to the objects and not the objects themselves. In effect, this feature means that PHP will behave as if the objects themselves are passed by reference.

Let's go back to Joe and Joanne. How would `wed()` behave differently now? First, `$joanne` and `$joe` will no longer be objects, but rather, object handles, let's say 4 and 7 respectively. These integer handles point to slots in some global objects table where the actual objects sit. When we send them to `wed()`, the local variables `$bride` and `$groom` will receive the values 4 and 7; `setHusband()` will change the object referenced by 4; `setWife()` will change the object referenced by 7; and when `wed()` returns, `$joanne` and `$joe` will already be living the first day of the rest of their lives together.

What does that mean to end-users?

Alright, so the ending to the story is now more idyllic, but what does it mean to PHP developers? It means quite a number of things. First, it means that your applications will run faster, as there will be much less data-copying going around. For instance, when you send `$joe` to a function, instead of having to create a replica, and copy over his name, birth date, parents' name, list of former addresses, social security number and whatnot – PHP will only have to pass on one object handle, one integer. Of course, a direct result of this is also a significant amount of memory savings – storing an integer requires much less space than storing a full-fledged replica of the object.

But perhaps more important, the new object model makes object oriented programming in PHP much more powerful and intuitive. No longer will you have to mess up with cryptic & signs in order to get the job done. No longer will you have to worry about whether changes you make to the object inside the constructor will survive the dreaded *new*-operator behavior. No longer will you ever have to stay up until 2:00AM tracking elusive bugs! Ok, maybe I'm lying with that last one, but seriously, the new object model reduces the object-related stay-up-until-2:00AM type of bugs very significantly. In turn, it means that the feasibility of using PHP for large-scale projects becomes much easier to explain.

What else is new?

As one could expect, the Zend Engine 2 packs quite a few other features to go along with its brand new object model. Some of the features further enhance object-oriented capabilities, such as private member variables and methods, static variables and language-level aggregation. Most notable is the revolutionized interaction with external component models, such as Java, COM/DCOM and .NET through overloading.

In comparison to the Zend Engine 1 in PHP 4.0, which first introduced this sort of integration, the new implementation is much quicker, more complete, more reliable and even easier to maintain and extend. This means that PHP 5.0 will play very nicely in your existing Java or .NET based setup, as you will be able to use your existing components inside PHP transparently, as if they were regular PHP objects. Unlike PHP 4.0, that had a special implementation for such overloaded objects, PHP 5.0 uses the same interface for all objects, including native PHP objects. This feature ensures that PHP objects and overloaded objects behave in exactly the same way.

Finally, the Zend Engine 2 also brings exception handling to PHP. To date, the sad reality is that most developers write code that does not handle error situations gracefully. It's not uncommon to see sites that spit out cryptic database errors to your browser, instead of displaying a well-phrased 'An error has occurred' kind of message. With PHP, the key reason for this is that handling error situations is a daunting task – you actually have to check for the return value of each and every function. Since `set_error_handler()` was added, this issue became slightly easier to manage, as it was possible to centralize error handling – but it still left a lot to be desired. Adding exception handling to PHP will allow developers both fine-grained error recovery, but more important it will facilitate graceful application-wide error recovery.

Conclusion

The release of PHP 5.0, powered by the Zend Engine 2.0, will mark a significant step forward in PHP's evolution as one of the key Web platforms in the world today. While keeping its firm commitment to users who prefer using the functional structured syntax of PHP, the new version will provide a giant leap ahead for those who are interested in its object oriented capabilities – especially for companies developing large scale applications.

php|a

Zeev has been working for over five years on the PHP project. Along with Andi Gutmans, he started the PHP 3 and 4 projects and wrote most of their infrastructure and core components, thereby helping to forge PHP as we know it today and attracting many more developers to join the movement. Zeev is a co-founder and CTO of Zend Technologies Ltd, the leading provider of development and performance management tools for PHP-enabled enterprises. Zend's website is www.zend.com.