

New Features in PHP 5.3

Bradley Holt (<http://bradley-holt.com/>)



Introduction

Are you using PHP 5.3?

Note that I'm using PHP 5.3 in development but not in production.

Past, Present & Future

- ⦿ PHP 5.0 brought us a new object model
- ⦿ PHP 5.3 brings us namespaces, closures, late static binding & more
- ⦿ PHP 5.3.99—huh?

So what?

- ⦿ Speed & memory improvements
- ⦿ Some problems are easier to solve in PHP 5.3
- ⦿ Zend Framework 2.0, Symfony 2.0, Lithium & Doctrine 2.0 will require PHP 5.3

The problems that are easier to solve tend to be those that are common in frameworks.

New Features

Namespaces, Late Static Binding & Closures

Namespaces

Namespaces provide you with another level of encapsulation and allow you to better organize your code.

Namespaced Classes

```
<?php  
namespace btvphp\stuff;  
class Foo {}  
$a = new \btvphp\stuff\Foo();  
$b = new Foo(); // Same as above
```

```
$p = new Foo(); // Same as above  
$q = new \btvphp\stuff\Foo();
```

Namespaces are analogous to directories when it comes to the backslash (\).

Namespaced Functions

```
<?php
namespace btvphp\stuff;
function sayHi ()
{
    return 'Hi';
}
echo \btvphp\stuff\sayHi () ; // Hi
echo sayHi () ; // Hi
echo sayHi () ; \\ Hi
echo /pфлбпб/сфлтт/сялгт () ; \\ Hi
```

Namespace Aliasing

Examples courtesy of Matthew Weier O'Phinney
[\(http://weierophinney.net/matthew/\)](http://weierophinney.net/matthew/)



Aliasing Classes

```
<?php  
namespace Zend\SignalSlot {  
    class Signals {}  
}  
namespace { // global namespace  
    use Zend\SignalSlot\Signals;  
    $signals = new Signals();  
}  
}
```

signals = new Signals();

use Zend\SignalSlot\Signals;

The bracketed syntax is recommended when more than one namespace is in a file. The “namespace” without any qualifier references the global namespace.

Changing the Name

```
<?php  
namespace Zend\Loader {  
    class PluginBroker {}  
}  
namespace {  
    use Zend\Loader\PluginBroker as Broker;  
    $broker = new Broker();  
}  
}  
  
$broker = new Broker();
```

Global Resolution

```
<?php  
namespace Doctrine {  
    class Manager {  
        public static function load() {}  
    }  
}  
namespace {  
    \Doctrine\Manager::load();  
}  
}  
/Doctrine/Manager::load();  
/Doctrine/Manager::load();
```

Late Static Binding

The Problem

Parent Class

```
<?php  
class Foo {  
    protected static function speak() {  
        return 'Hi';  
    }  
    public static function sayHi() {  
        return self::speak();  
    }  
}  
}  
}
```

What happens if I call Foo::sayHi()?

Child Class

```
<?php  
class Bar extends Foo {  
    protected static function speak() {  
        return 'Hello';  
    }  
}  
}  
}
```

What happens if I call Bar::sayHi()?

“Hi” or “Hello”?

```
<?php  
echo Bar::sayHi();  
ECHO Bar::sayHi();
```

“Hi” or “Hello”?

```
<?php  
echo Bar::sayHi(); // Hi  
ECHO Bar::sayHi(); // Hi
```

Static references to the current class (self) are resolved using the class in which the function was defined.

The Solution

Parent Class

```
<?php  
class Foo {  
    protected static function speak() {  
        return 'Hi';  
    }  
    public static function sayHi() {  
        return static::speak();  
    }  
}  
}
```

Use “static” keyword instead of “self”.

Child Class

```
<?php  
class Bar extends Foo {  
    protected static function speak() {  
        return 'Hello';  
    }  
}  
}  
}
```

The child class remains unchanged for this example.

“Hi” or “Hello”?

```
<?php  
echo Bar::sayHi(); // Hello  
echo Bar::sayHi(); // Hello
```

The “static” keyword references the class that was initially called at runtime, in this case “Bar”.

Closures / Lambda Functions

See: <http://bit.ly/9LYP3r>



Variable Assignment

```
<?php
$sayHi = function () {
    return 'Hi';
}
echo $sayHi(); // Hi
echo $sayHi(); // Hi
};
```

Scope

```
<?php
$sayWhat = 'Hi';
$say = function ($toWhom) use ($sayWhat) {
    return $sayWhat . ', ' . $toWhom;
};
echo $say('Bradley'); // Hi, Bradley
echo $say('Biggles'); // Hi, Biggles
};
```

The “use” parameters are passed in when the closure is created.

Anonymous Functions

```
<?php
$values = array(3, 7, 2);
usort($values, function ($a, $b) {
    if ($a == $b) { return 0; }
    return ($a < $b) ? -1 : 1;
}) ;
/* [0] => 2
   [1] => 3
   [2] => 7 */
```

```
[S] => \ * \
[T] => 3
```

This is a contrived example, there are better ways to sort this particular array.

Other Neat Stuff

New Bundled Extensions

- ⦿ Phar (PHP Archive)
- ⦿ Internationalization Functions
- ⦿ Fileinfo: guesses content type & encoding
- ⦿ SQLite version 3
- ⦿ Enchant spelling library

Extension Improvements

OpenSSL

- ⦿ More OpenSSL functionality available natively within PHP
- ⦿ Faster than do-it-yourself or system calls
- ⦿ Useful if you're working with OpenID, etc.

DateTime Object

- ⦿ Add or subtract date intervals
- ⦿ Calculate the difference between two dates
- ⦿ Get or set unix timestamp
- ⦿ See: <http://bit.ly/5pDpWI>



SPL Data Structures

See: <http://bit.ly/bz6pqY>



Matthew Turland did a presentation on this at TEK and has performance tests you can run for yourself.

SplStack

- ⦿ Push & Pop
- ⦿ Last In, First Out (LIFO)
- ⦿ Uses less memory than arrays for big stacks
(greater than 5,000 elements)

As with other SPL data structures, SplStack provides a specialized alternative to using an array.

SplQueue

- ⦿ Enqueue & Dequeue
- ⦿ First In, First Out (FIFO)
- ⦿ Faster and uses less memory than arrays for most queues

SplHeap

- ⦿ Insert & Remove
- ⦿ Reorders elements based on comparisons
- ⦿ Faster and uses less memory than arrays for most heaps

goto



Courtesy of xkcd: <http://xkcd.com/292/>



Think of GOTO as a more flexible break statement.

New Syntax

__invoke()

```
<?php  
class Foo {  
    public function __invoke($x) {  
        return $x + $x;  
    }  
}  
  
$foo = new Foo();  
echo $foo(2); // 4  
  
echo $foo(); // 4  
$foo = new Foo();
```

This allows you to call an object as a function.

__callStatic()

```
<?php  
class Foo {  
    public static function __callStatic  
($name, $args) {  
        return $name . ' called  
statically';  
    }  
}  
echo Foo::bar(); // bar called statically  
echo Foo::par(); // par called statically  
}
```

This is similar to `_call` but for static methods.

DIR

```
<?php  
echo dirname(__FILE__);  
echo __DIR__; // Since PHP 5.3
```

```
echo __DIR__; // Since PHP 5.3
```

This gets you the directory of the current file without needing to use the dirname function on __FILE__.

Miscellaneous

- ⦿ Nowdocs: “Nowdocs are to single-quoted strings what heredocs are to double-quoted strings.”
- ⦿ Improved ternary (?:) operator

I'm sold, what now?

Platform Support

Linux

- ⦿ Ubuntu 10.10
- ⦿ Fedora 12+
- ⦿ openSuse 11.2+
- ⦿ Red Hat Enterprise Linux (RHEL) 6

Since RHEL will have PHP 5.3 soon, so will CentOS.

Mac OS X

- ⦿ Bundled with Snow Leopard
- ⦿ MacPorts
- ⦿ Homebrew

Windows

- ⦿ Binary packages available
- ⦿ WebMatrix Beta 3

PHP 5.3 Hosting

- ServerGrove
- WebMatrix
- A2 Hosting
- Hostek

Resources

- ⦿ PHP Manual (<http://php.net/>)
- ⦿ PHP 5.3.0 Release Announcement (http://php.net/releases/5_3_0.php)
- ⦿ Migrating from PHP 5.2.x to PHP 5.3.x (<http://www.php.net/manual/en/migration53.php>)

Questions?

Thank You

Bradley Holt (<http://bradley-holt.com/>)





New Features in PHP 5.3 by Bradley Holt is licensed under a
Creative Commons Attribution 3.0 Unported License.

