# 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?

Is anyone not on PHP 5 yet?

# So what?

- Some problems are easier to solve in PHP 5.3
- Speed & memory improvements
- 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
<?php
namespace btvphp\stuff;
class Foo {}
$a = new \btvphp\stuff\Foo();
$b = new Foo(); // Same as above
```

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

# Namespaced Functions

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

# Using Other Namespaces

```php
<?php
namespace btvphp\stuff {
    class Foo {}
}
namespace btvphp\misc {
    use btvphp\stuff as stuff;
    $a = new stuff\Foo();
}
```

The bracketed syntax is recommended when more than one namespace is in a file.

# Late Static Binding

# The Problem

# Parent Class

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

```
}
    }
```

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

# “Hi” or “Hello”?

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

# “Hi” or “Hello”?

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

```
}
    }
```

The child class remains unchanged for this example.

# “Hi” or “Hello”?

```php
<?php
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

Link is to Vance Lucas' article on Practical Uses for PHP 5.3 Closures

# Variable Assignment

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

# Scope

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

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

# Anonymous Functions

```php
<?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 */
```

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

Brandon Savage has a good article on this.

# 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

- Nowdocs: "Nowdocs are to single-quoted strings what heredocs are to double-quoted strings."
- Improved ternary (`?:`) operator
- Added `__callStatic()` allowing for dynamic static method overloading

I'm sold, what now?

# Platform Support

- Linux
  - Ubuntu 10.10
  - Fedora 12+
  - openSuse 11.2+
- OS X Snow Leopard (MacPorts & Homebrew)
- Binary packages available for Windows

Others?

# PHP 5.3 Hosting? Hmm...

Anyone?

# 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/)