

# Metaprogramming with PHP

David Sklar

New York PHP Conference 2006

June 15, 2006

<http://www.sklar.com/>

# ¿ metaprogramming ?

- Programs that write programs that write programs that write programs...
- Writing a **little language** for better expression of a particular problem
- (the **Domain Specific Language**)

# Why should you care?

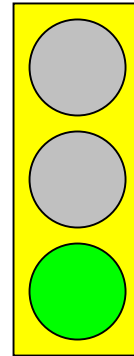
- **Genuinely useful** in some common situations
- **Helpful** framework for thinking about problem solving
- Often, just **nifty**

# Today

- **magic methods**
- **stream wrappers**
- **SPL interfaces**
- **various experimental extensions**

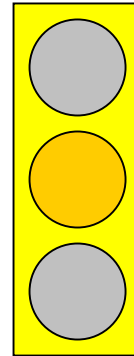
# Watch the Traffic Lights

- Green means **go**



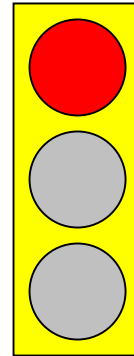
# Watch the Traffic Lights

- **Yellow: caution**



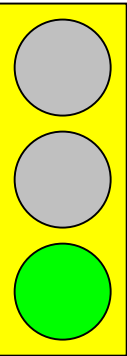
# Watch the Traffic Lights

- Red: **just for fun**



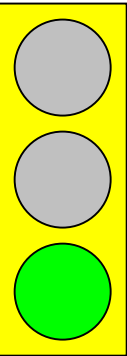
# \_\_magic()

```
class Zoltar {  
    function __get($prop) { }  
    function __set($prop, $value) { }  
    function __call($method, $args) { }  
}
```



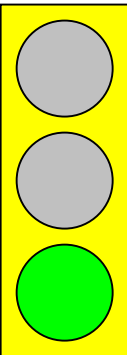
# \_\_get() and \_\_set()

- Read-only (or write-once) properties



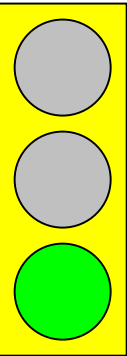
```
class WORM {
    protected $_props = array();
    public function __set($prop, $value) {
        if (! array_key_exists($prop, $this->_props)) {
            $this->_props[$prop] = $value;
        }
    }
    public function __get($prop) {
        if (array_key_exists($prop, $this->_props)) {
            return $this->_props[$prop];
        } else {
            return null;
        }
    }
}
```

```
$w = new WORM();
$w->boogaloo = 'electric';
print "It's $w->boogaloo \n";
$w->boogaloo = 'hydrodynamic';
print "It's $w->boogaloo \n";
```



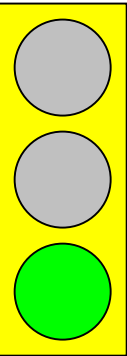
```
$w = new WORM();  
$w->boogaloo = 'electric';  
print "It's $w->boogaloo \n";  
$w->boogaloo = 'hydrodynamic';  
print "It's $w->boogaloo \n";
```

It's electric  
It's electric



# \_\_get() and \_\_set()

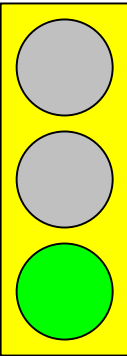
- Properties can be **tied** to external resources



```

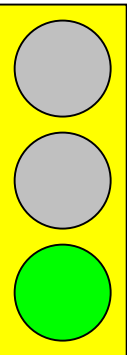
class FileBucket {
    protected $_dir = null;
    protected $_data = array();
    public function __construct($dir) {
        if (! (is_dir($dir) && is_readable($dir))) {
            throw new Exception("Can't read directory $dir");
        } else {
            $this->_dir = $dir;
        }
    }
    public function __get($prop) {
        if (isset($this->_data[$prop])) {
            return $this->_data[$prop];
        } else {
            $file = $this->propToFile($prop);
            if (is_file($file) && is_readable($file)) {
                $this->_data[$prop] = file_get_contents($file);
                return $this->_data[$prop];
            } else {
                throw new Exception("Can't read file for $prop");
            }
        }
    }
}

```



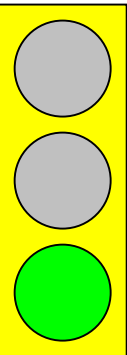
```
public function __set($prop, $value) {
    $this->_data[$prop] = $value;
    file_put_contents($this->propToFile($prop),
        $this->_data[$prop]);
}
protected function propToFile($prop) {
    return $this->_dir . DIRECTORY_SEPARATOR . $prop;
}
}
```

```
$fb = new FileBucket('/etc');
print $fb->passwd;
$fb->{'passwd.bak'} = $fb->passwd;
```

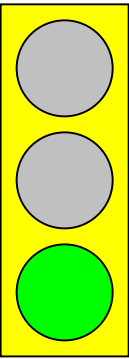


# \_\_call()

- Call methods that didn't exist when you wrote your code!



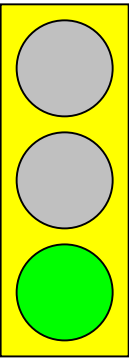
```
class Formatter {
    protected $_target = null;
    public function __construct($target) {
        $this->_target = $target;
    }
    public function __call($method, $args) {
        if (class_exists("OutputFormatter_{$method}")) {
            $fmt = array("OutputFormatter_{$method}",
                'format');
            array_push($args, $this->_target);
            return call_user_func_array($fmt, $args);
        } else {
            throw new Exception("No {$method} formatter");
        }
    }
}
```



```
class Sandwich {
    public $format;
    public function __construct() {
        $this->format = new Formatter($this);
    }
    public function getIngredients() {
        return array('peanut butter', 'banana', 'bread');
    }
}

class OutputFormatter_html {
    public static function format($target) {
        print '<ul>';
        foreach ($target->getIngredients() as $ingredient) {
            print '<li>'.htmlspecialchars($ingredient).'</li>';
        }
        print '</ul>';
    }
}

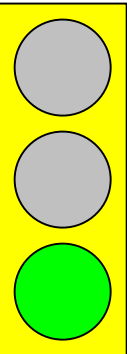
class OutputFormatter_text {
    public static function format($target) {
        foreach ($target->getIngredients() as $ingredient) {
            print '* '.$ingredient."\n";
        }
    }
}
```



```
$s = new Sandwich;  
print $s->format->html();  
print $s->format->text();
```

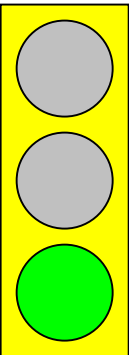
```
<ul><li>peanut butter</li><li>banana</li>  
<li>bread</li></ul>
```

- \* peanut butter
- \* banana
- \* bread



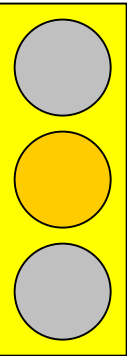
# Properties vs. Methods

- Properties and methods can be used as two **axes** of data on the same structure
- Properties are **concise**, but methods can have **lots of info** passed to them.



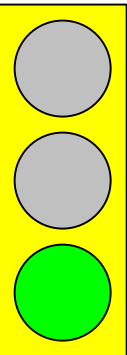
# Leaks

- `__get()` and `__set()` **ignored** if property exists
- No `$sandwich->meats[] = 'Tongue';`
- `__call()` **ignored** if method exists
- No magic with **static** properties or methods



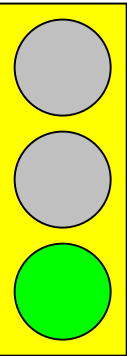
# Stream Wrappers

- Use's PHP's I/O functions for anything that you can represent as a stream of bytes



# You could be a stream if...

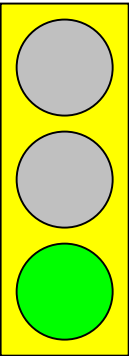
- You can be represented as some pile of data in a **hierarchy** with node content that is a **bytestream**.



# Example: WebDAV

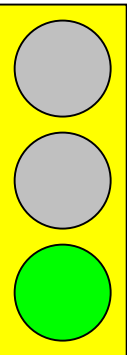
- PEAR HTTP\_WebDAV client implements webdav and webdavstreams:

```
require 'HTTP/WebDAV/Client.php';  
$path =  
'webdav://dav.example.org/tla.txt';  
$doc = file_get_contents($path);
```



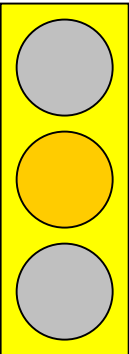
# Writing a Stream Handler

- Implement functions listed at [http://php.net/stream\\_wrapper\\_register](http://php.net/stream_wrapper_register)
- Register the stream with `stream_wrapper_register()`
- Enjoy!



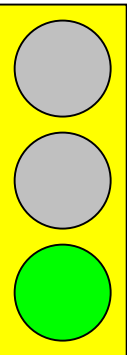
# Leaks

- Only works with functions that hook into PHP's streams interface, not functions that directly call system functions
- No `glob()`; no `tempnam()`;
- Bug 27508 (`stream_feof()` in < 5.1.0)



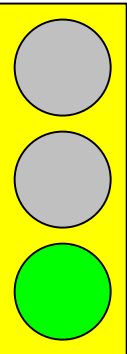
# SPL

- **Classes and interfaces that make PHP 5 more OOPy**
- `http://www.php.net/~helly/php/ext/spl/`



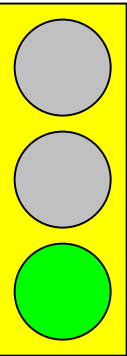
# interface `ArrayAccess`

- Pretend that an **object** is an **array**
- Load elements **lazily**
- **Hide complexity** behind square brackets
- Use a **third axis** for data structures



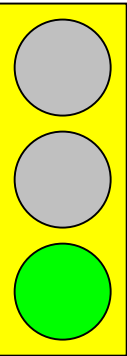
# Load on Demand

- Report total size up front
- Parse/load elements as requested
- Go back for more when they're needed



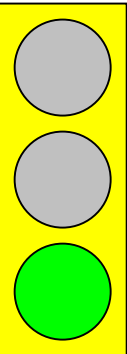
# Remote Data as Array

- Apps from Ning Content Store
- Returned in 100-app chunks
- Iteration is transparent



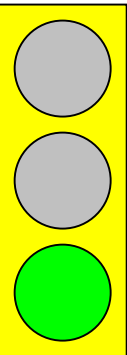
# interface ArrayAccess

```
public function offsetExists($offset);  
public function offsetGet($offset);  
public function offsetSet($offset,  
                           $value);  
public function offsetUnset($offset);
```



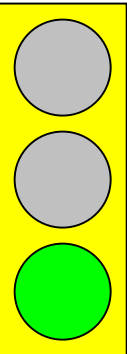
# interface Countable

```
public function count();
```



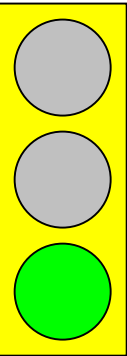
# interface Iterator

```
public function current();  
public function key();  
public function next();  
public function rewind();  
public function valid();
```



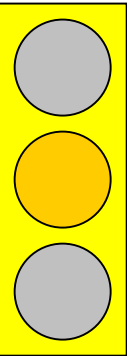
# "Array" as Data Axis

- Similar to SimpleXML - properties are element children, array elements are attributes.



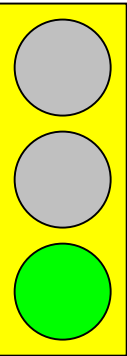
# Leaks

- Countable is  $\geq 5.1.0$
- `is_array()` doesn't work



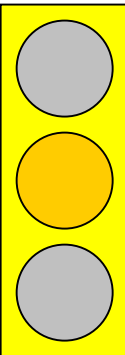
# Reflection

- Information at runtime about code structure
- `call_user_func()`, `class_exists()`,  
`get_class_vars()`,  
`get_declared_classes()`, etc...
- Reflection\* classes



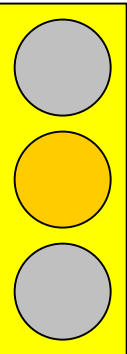
# Reflection + Docblock = Annotations

- Represent "models" as PHP classes with annotations in Docblock comments
- `docblock` extension speeds/eases Docblock parsing but not strictly necessary.



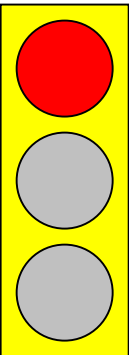
# Leaks

- No Docblock comments saved on consts
- Reflection classes have many changes and additions with different PHP versions



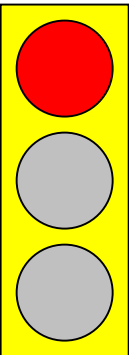
# Operator Overloading

- Why should `+` always mean "add two numbers"?
- **Drastic** syntactical change.
- <http://pecl.php.net/package/operator>



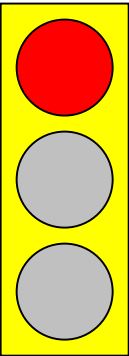
# Object Addition

```
$f = new Feed('test');  
$i = new Item();  
$i->title = 'News flash!';  
$i->link = 'http://www.example.com';  
$i->description =  
    'Breaking news in this item!!!';  
$f += $i;  
$f = $f + new Item('Cheese stolen!',  
    'http://cheese.example.org',  
    'Who moved my cheese?');  
print $f->toXml();
```



# Fun With Regular Expressions

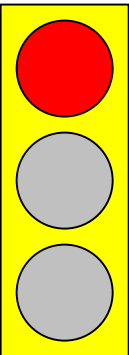
```
$r = new Regex( '/^(\\d{5})(-\\d{4})?$/' );
$codes = array( '19096', '06520-5657',
                'SW1A 1AA' );
foreach ( $codes as $code ) {
    print $code .
        ( $r == $code ? ' is' : ' is not' ) .
        ' valid';
}
foreach ( $codes as $code ) {
    $subpatterns = $r & $code;
}
```



# Fun with Strings

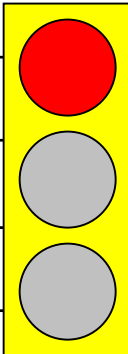
```
$str = new String('Foo');  
//possible but stupid  
$str = $str + "Bar";  
echo $str + PHP_EOL;  
//missed this?  
$str = $str * 2;  
echo $str + PHP_EOL;  
// greetings from python  
$str = new String("int(%d) float(%f)");  
$str = $str % array(23, 2.0815);  
echo $str + PHP_EOL;
```

From: <http://blog.ghosthowwalksinside.org/archives/2006/programming/php/operator-magic>



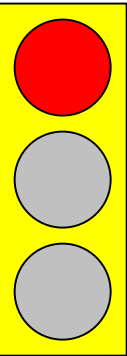
# Operator `__magic()`

<code>+</code>	<code>__add()</code>	<code>-</code>	<code>__sub()</code>
<code>*</code>	<code>__mul()</code>	<code>/</code>	<code>__div()</code>
<code>%</code>	<code>__mod()</code>	<code>.</code>	<code>__concat()</code>
<code>&gt;&gt;</code>	<code>__sr()</code>	<code>&lt;&lt;</code>	<code>__sl()</code>
<code> </code>	<code>__bw_or()</code>	<code>&amp;</code>	<code>__bw_and()</code>
<code>^</code>	<code>__bw_xor()</code>	<code>===</code>	<code>__is_identical()</code>
<code>==</code>	<code>__is_equal()</code>	<code>!==</code>	<code>__is_not_identical()</code>
<code>&lt;</code>	<code>__is_smaller()</code>	<code>!=</code>	<code>__is_not_equal()</code>
<code>~</code>	<code>__bw_not()</code>	<code>(bool)</code>	<code>__bool()</code>
<code>++</code>	<code>__pre_inc()</code>	<code>!(bool)</code>	<code>__bool_not()</code>
<code>++</code>	<code>__post_inc()</code>	<code>--</code>	<code>__pre_dec()</code>
<b>Also</b>	<b><code>__assign_XXX()</code></b>	<code>--</code>	<code>__post_dec()</code>



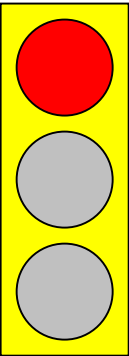
# Leaks

- Easy to make unreadable code
- Can't overload  $>$   $\geq$  (without a patch)



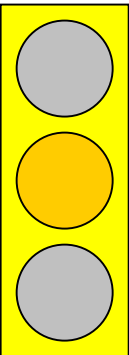
# runkit

- **Replace functions, methods, constants at runtime**
- **Change class ancestry**
- **Run sandboxed code**
- **Mint your own autoglobals**
- <http://pecl.php.net/package/runkit>
- <http://www.sebastian-bergmann.de/AspectPHP/>



# pdo\_user

- Write a PDO backend -- in PHP
- What data source would you like an SQL interface to?
- [http://pecl.php.net/package/pdo\\_user](http://pecl.php.net/package/pdo_user)



# Oz(es) Behind the Curtain

- Thanks:
  - Wez Furlong (streams, PDO)
  - Marcus Börger (SPL)
  - Sara Golemon (operator, pdo\_user, runkit)

