

Worst PHP Practice

Marcus Börger
Johannes Schlüter

Topics

- ↳ Security
- ↳ Overdesign
- ↳ Spagetthi code
- ↳ DIY – Do It Yourself
- ↳ Utilize available Tools
- ↳ Micro Optimizations
- ↳ References
- ↳ Do everything with Objects
- ↳ Include vs. Require vs `_once`
- ↳ Provide a Style Guide
- ↳ Use with Caution

Security

- M** Address security once the application is ready
- M** No hacker will ever care for my application
- M** I do not have security issues

- I** Since hackers automatically scan, they will find you

- J** Take care of security right from the beginning
 - J** Security should and will influence:
 - J** Your overall design
 - J** Your development and deployment process

Overdesign

- M** Always plan for everything
- I** Limit yourself to what you and your customer want
- I** Do not fear restarting development
- J** The more complex your design gets:
 - J** The more complex your code gets
 - J** The more bugs you have
 - J** The more the development will cost
 - J** The more likely you are to miserably fail
- J** PHP is not: Java, C++, Python, Ruby on Rails



Spaghetti code

- M** This code just needs a little bit more tweaking
- I** Modularize / Componentize your code
- J** Every day code can put in base repository
- J** Not everything you use twice belongs there

DIY – Do It Yourself

M Implementing everything yourself

M Waste of time

M Development

M Testing

M Documenting

M Maintenance

M Creating unnecessary bugs

J Prefer NIH

J Existing code should be

J Well developed

J Tested

J Documented

J Maintained

J Have very few bugs if at all

Utilize available Tools

M Designing, Testing, Versioning, Documenting . . .

... That all takes far too much time!

I Software design lets you capture errors early

I Testing obviously lets you find bugs

I Versioning helps you track down issues

I Documenting helps everyone understand the code

J Familiarize yourself with available tools

J Design: UML might be overkill, but . . .

J Testing: Run-tests, SimpleTest, PHPUnit, . . .

J Versioning: SVN, HG, GIT

Micro Optimizations

- M** Always write optimized code

- I** Optimized code usually is harder to maintain
- I** Harder to maintain code is often more error prone
- I** Writing optimized code takes longer

- J** Follow the 80 : 20 rule
 - J** 80% of the time is spent in 20% code
 - J** Optimizing the 80% by 20% gains: 4%
 - J** Optimizing the 20% by 10% gains: 8%

 - J** Use Profiling – System Profiling

References

- M** Using references to optimize code
- I** References don't do what you think they do
- I** Do not use references (avoid them like holy water)

References

```
function ConfigFramework(ARRAY $config) {  
    // . . .  
}
```

```
$config = array(...);
```

```
ConfigFramework($config);
```

```
class Application {  
    function __construct($config) {  
        $this->config = $config;  
    }  
}
```

```
$app = new Application($config);
```

References

```
function ConfigFramework(ARRAY $config) {  
    // Expensive read function  
}
```

```
$config = array(...);
```

```
ConfigFramework($config);  
// This configure stuff is somehow slow
```

```
class Application {  
    function __construct($config) {  
        $this->config = $config;  
    }  
}
```

```
$app = new Application($config);
```

References

```
function ConfigFramework(ARRAY &$config) {  
    // Expensive read function  
}
```

```
$config = array(...);
```

```
ConfigFramework($config);  
// Should be faster now, no?
```

```
class Application {  
    function __construct($config) {  
        $this->config = $config;  
    }  
}
```

```
$app = new Application($config);
```

References

```
function ConfigFramework(ARRAY &$config) {  
    // Expensive read function  
}
```

```
$config = array(...);
```

```
ConfigFramework($config);  
// Now $config is a reference
```

```
class Application {  
    function __construct($config) {  
        $this->config = $config;  
    }  
}
```

```
// And now the following is slow  
$app = new Application($config);
```


In PHP all values are zval's

```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;
```

```
IS_NULL
IS_LONG
IS_DOUBLE
IS_BOOL
IS_ARRAY
IS_OBJECT
IS_STRING
IS_RESOURCE
```

```
typedef union _zvalue_value {
    long lval;
    double dval;
    struct {
        char *val;
        int len;
    } str;
    HashTable *ht;
    zend_object_value obj;
} zvalue_value;
```

In PHP all values are zval's

```
typedef struct _zval_struct {  
    zvalue_value value;  
    zend_uint refcount;  
    zend_uchar type;  
    zend_uchar is_ref;  
} zval;
```

Userspace notion of "Reference"

0 == Not a reference

1 == Is a reference

How many "labels" are
associated with this zval?

Copy On Write

```
typedef struct _zval_struct {  
    zvalue_value value;  
    zend_uint refcount;  
    zend_uchar type;  
    zend_uchar is_ref;  
} zval;
```

- Has a value of 0 (zero)
- zval shared by 1 or more labels
- If one label wants to make a change, it must leave other labels with the original value.

`$a = 123;`

`$a`

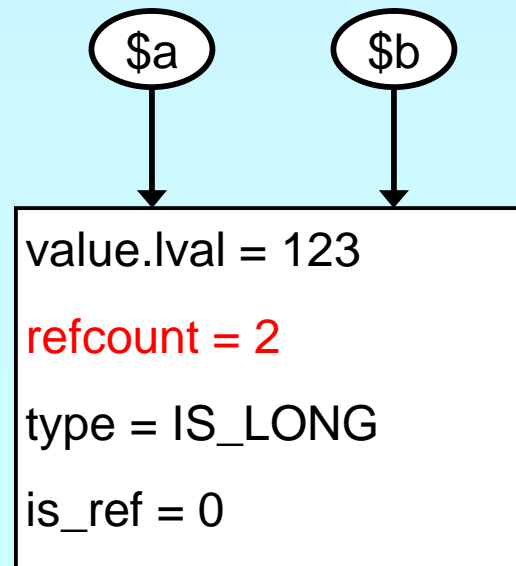
value.lval = 123
refcount = 1
type = IS_LONG
is_ref = 0

Copy On Write

```
typedef struct _zval_struct {  
    zvalue_value value;  
    zend_uint refcount;  
    zend_uchar type;  
    zend_uchar is_ref;  
} zval;
```

- Has a value of 0 (zero)
- zval shared by 1 or more labels
- If one label wants to make a change, it must leave other labels with the original value.

```
$a = 123;  
$b = $a;
```



Copy On Write

```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;
```

- Has a value of 0 (zero)
- zval shared by 1 or more labels
- If one label wants to make a change, it must leave other labels with the original value.

```
$a = 123;
```

```
$b = $a;
```

```
$b = 456;
```

\$a

```
value.lval = 123
refcount = 1
type = IS_LONG
is_ref = 0
```

\$b

```
value.lval = 456
refcount = 1
type = IS_LONG
is_ref = 0
```

Full Reference

```
typedef struct _zval_struct {  
    zvalue_value value;  
    zend_uint refcount;  
    zend_uchar type;  
    zend_uchar is_ref;  
} zval;
```

- Has a value of 1 (one)
- zval shared by 1 or more labels
- If one label wants to make a change, it does so, causing other labels to see the new value.

```
$a = 123;
```

\$a

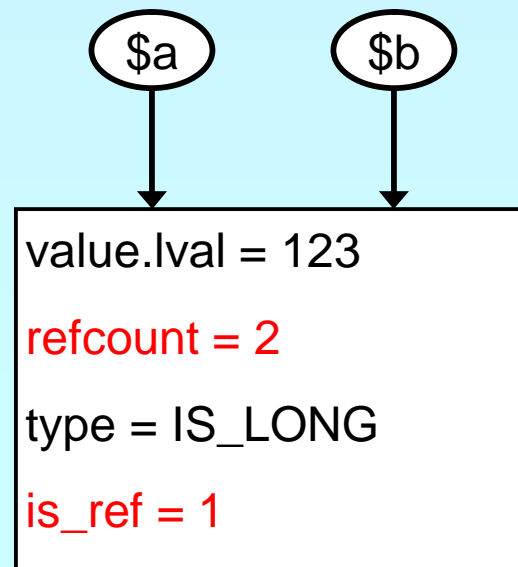
```
value.lval = 123  
refcount = 1  
type = IS_LONG  
is_ref = 0
```

Full Reference

```
typedef struct _zval_struct {  
    zvalue_value value;  
    zend_uint refcount;  
    zend_uchar type;  
    zend_uchar is_ref;  
} zval;
```

- Has a value of 1 (one)
- zval shared by 1 or more labels
- If one label wants to make a change, it does so, causing other labels to see the new value.

```
$a = 123;  
$b = &$a;
```



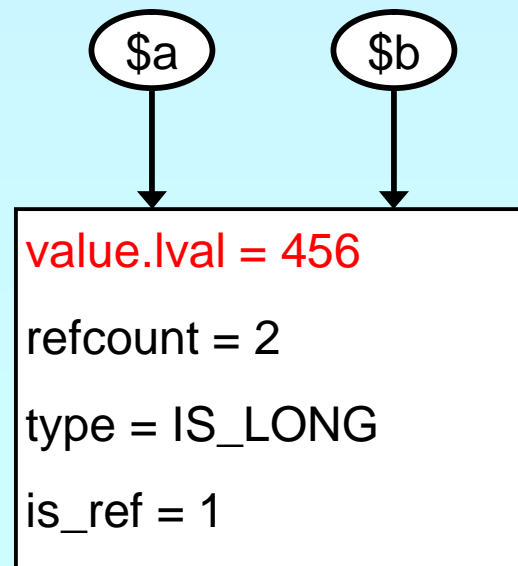
Full Reference

```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;
```

- Has a value of 1 (one)
- zval shared by 1 or more labels
- If one label wants to make a change, it does so, causing other labels to see the new value.

```
$a = 123;
$b = &$a;

$b = 456;
```



Do everything with Objects

- M** Everything must be an object
- I** PHP supports procedural code
- J** When you use a singleton factory
 - J** You could have used globals
- J** An object that simply stores values
 - J** Could simply be an array

Include vs. Require vs _once

M require_once is the safe and correct way - always

I There are four versions for a reason

J include

J require

J include_once / require_once

J fpassthru()

M eval

It Is All About Style

Provide a Style Guide

- J Provide actual coding rules (coding style)
- J Provide useful error handling
- J Always develop with `E_STRICT` + `E_NOTICE` on
- J Use your logs
- J Use `.inc` for includes + care for server config
- J Use `"` instead of `'`
- J Do not constantly switch between HTML and PHP
- J Do not use `auto_prepend_file`, `auto_append_file`
- J Do not leave debugging in production
- J Do we really need to mention `register_globals`?
- J Magic quotes: Filter on input, escape output

Use with Caution

- J `$_REQUEST`
- J `__get`, `__set`, `__isset`, `__unset`
- J `__call`, `__callStatic`
- J `__autoload`
- J `@`
- J `<? =`

Reference

- ↳ Everything about PHP
<http://php.net>
- ↳ These slides
<http://talks.somabo.de>
- ↳ George Schlossnagle
[Advanced PHP Programming](#)
- ↳ Andi Gutmans, Stig Bakken, Derick Rethans
[PHP 5 Power Programming](#)