(Re)discovering the SPL

Joshua Thijssen

jaytaph

PHPUK 2014
London - UK

Joshua Thijssen

Freelance consultant, developer and trainer @ NoxLogic

Founder of the Dutch Web Alliance

Development in PHP, Python, C, Java.
Lead developer of Saffire.

Blog: http://adayinthelifeof.nl
Email: jthijssen@noxlogic.nl
Twitter: @jaytaph

NOX LOGIC

Q: Have you ever used the SPL?

Q: Have you ever used the SPL and didn't went nuts?

SPL Documentation

http://www.php.net/spl

Introduction

The Standard PHP Library (SPL) is a collection of interfaces and class
meant to solve common problems.

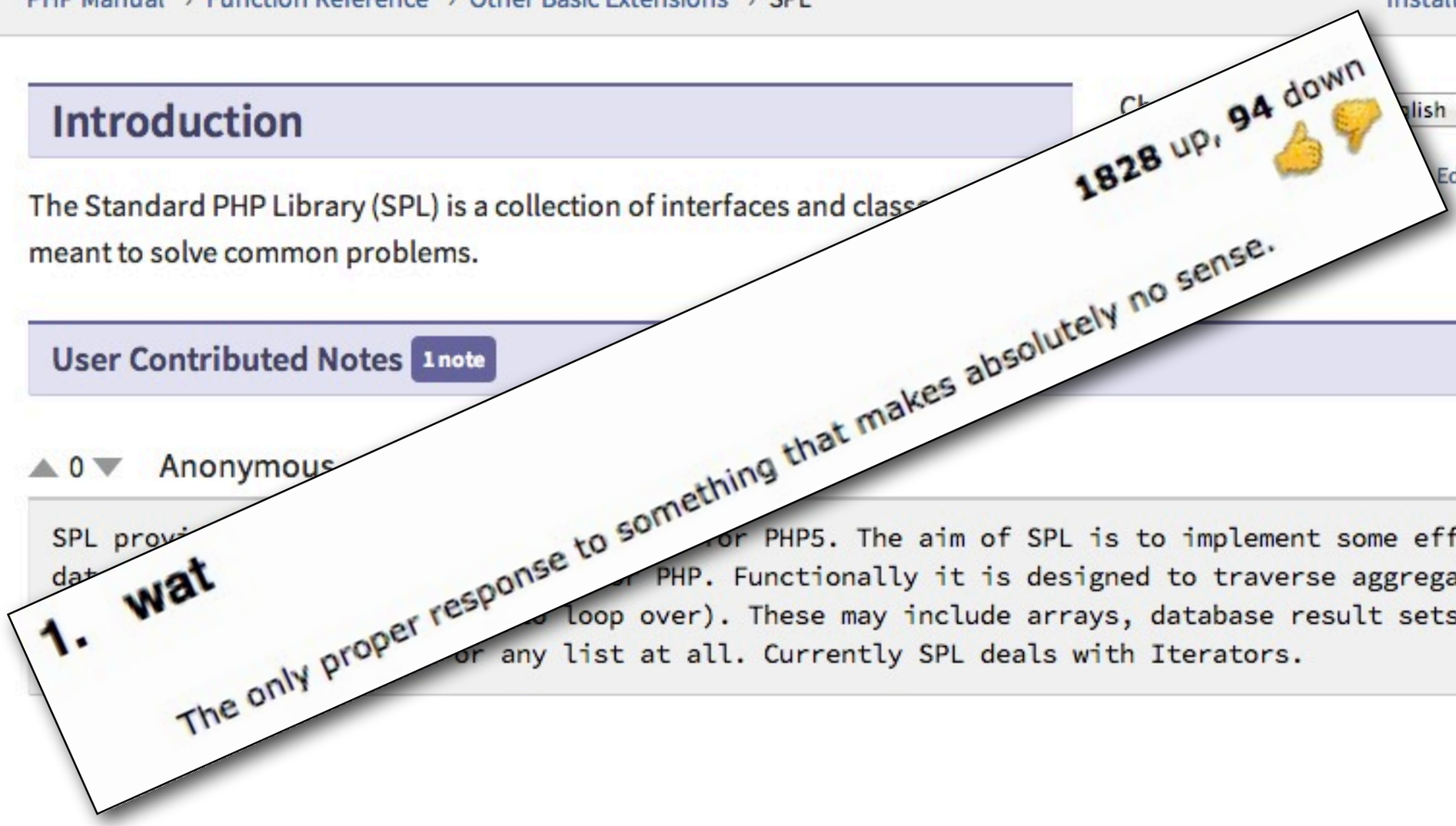
User Contributed Notes 1 note

add a note

▲ 0 ▼ Anonymous 1 year ago

SPL provi or PHP5. The aim of SPL is to implement some efficient dat PHP. Functionally it is designed to traverse aggregate o loop over). These may include arrays, database result sets, xml or any list at all. Currently SPL deals with Iterators.

add a note



- Not enough documentation.
- Very few examples.
- Wrong / missing in some cases.

- Interfaces
- Iterators
- Data structures
- Exceptions
- Miscellaneous functionality

Don’t get scared!
The SPL is awesomesauce!

5. awesomesauce — **50** up, **21** down

a word meaning awesome, only cooler-be prepared to be looked at funny when you use this word.

Girl 1: Woah, that movie was awesomesauce!!
Girl 2: Dude, you spend way too much time on Urban Dictionary.

INTERFACES

Traversable

(not an “spl interface”)

- Traversable cannot be implemented.
- Traversable can be detected (instanceof).
- foreach() detects traversable interfaces and does magic.

Iterator

(still not an “spl interface”)

Userland interface to make an object traversable

Iterator interface:

```
Iterator extends Traversable {
   /* Methods */
   abstract public mixed current ( void )
   abstract public scalar key ( void )
   abstract public void next ( void )
   abstract public void rewind ( void )
   abstract public boolean valid ( void )
}
```

- Iterator
  - FilterIterators
    - “Chains” iterators together
- IteratorAggregate

```
$dir = opendir(".");
while (($file = readdir($dir)) !== false) {

    # Business logic happens here
    print "file: $file\n";

}
```

```
$dir = opendir(".");
while (($file = readdir($dir)) !== false) {

    # hack: only display mp3 files
    if (! preg_match('|\.mp3$|i', $file)) {
        continue;
    }

    # Business logic happens here
    print "file: $file\n";
}
```

Filter on *.mp3

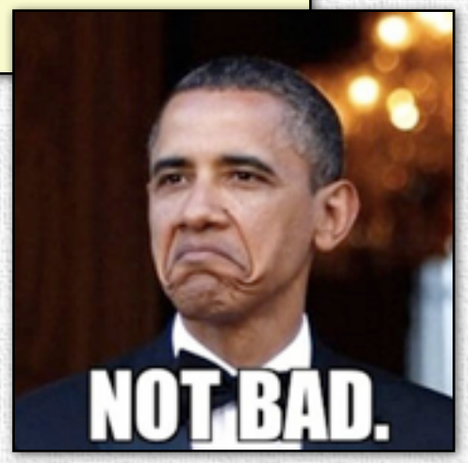
- Filter all MP3 and all JPG files.
- Filter all MP3 files that are larger than 6MB.
- Do not filter at all.
- Search sub-directories as well.
- Search multiple directories.

- How to test? (we can't)
- How to maintain? (we can't)
- How to reuse? (we can't)

```
$it = new DirectoryIterator(".");
foreach ($it as $fi) {
    print "File: ".$fi->getPathname()."\n";
}
```

```
$it = new DirectoryIterator(".");
$it2 = new RegexIterator($it, "/\.mp3$/i");
foreach ($it2 as $fi) {
    print "File: ".$fi->getPathname()."\n";
}
```

```
$it = new DirectoryIterator(".");
$it2 = new RegexIterator($it, "/\.mp3$/i");
$it3 = new FilesizeIterator($it2, 0, 6 * 1024 * 1024);
$it4 = new LimitIterator($it3, 10, 5);

foreach ($it4 as $fi) {
    print "File: ".$fi->getPathname()."\n";
}
```



- ✓ Reusable
  We can use iterators where ever we want.
- ✓ Testable
  Iterators can be tested separately.
- ✓ Maintainable
  No need to adapt our business logic.

Countable

(hurrah! An “spl interface”!)

```
class myIterator implements \Iterator {
   ...
}

$a = array(1, 2, 3);
$it = new myIterator($a);

print count($it);
```

```
1
```

```
class myCountableIterator extends myIterator implements Countable
{
    function count() {
          return count($this->_elements);
    }
}

$a = array(1, 2, 3, 4, 5);
$it = new myCountableIterator($a);

print count($it);
```

```
5
```

```
class myCountableIterator extends myIterator implements Countable
{
    function count() {
          return count($this->_arr);
      }
}

$a = array(1, 2, 3, 4, 5);
$it = new myCountableIterator($a);
$it2 = new limitIterator($it, 0, 3);

print count($it2);
```

```
1
```

SeekableIterator

- It’s not an iterator, it’s an interface.
- seek()
- Implementing “seekableIterator” can speed up other iterators.
- LimitIterator makes use of “seekableIterator”

ITERATORS

SPL Iterators

- AppendIterator
- ArrayIterator
- CachingIterator
- CallbackFilterIterator
- DirectoryIterator
- EmptyIterator
- FilesystemIterator
- FilterIterator
- GlobIterator
- InfiniteIterator
- IteratorIterator
- LimitIterator
- MultipleIterator
- NoRewindIterator
- ParentIterator
- RecursiveArrayIterator
- RecursiveCachingIterator
- RecursiveCallbackFilterIterator
- RecursiveDirectoryIterator
- RecursiveFilterIterator
- RecursiveIteratorIterator
- RecursiveRegexIterator
- RecursiveTreeIterator
- RegexIterator
- SimpleXMLIterator

NOX LOGIC

SPL Iterators

http://lxr.php.net/xref/PHP_5_5/ext/spl/internal/

{OpenGrok

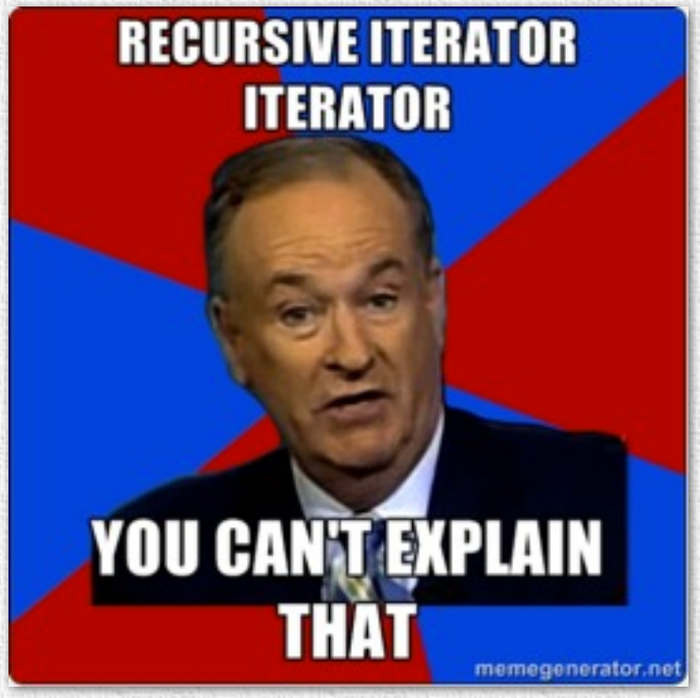
Cross Reference: appenditerator.inc

xref: /PHP_5_5/ext/spl/internal/appenditerator.inc

Home | History | Annotate | Line# | Navigate | Download Search only in /PHP_5_5/ext/spl/internal/

```
 1
 2
 3 /** @file appenditerator.inc
 4  * @ingroup SPL
 5  * @brief class AppendIterator
 6  * @author  Marcus Boerger
 7  * @date    2003 - 2009
 8  *
 9  * SPL - Standard PHP Library
10  */
11
12 /** @ingroup SPL
13  * @brief   Iterator that iterates over several iterators one after the other
14  * @author  Marcus Boerger
15  * @version 1.0
16  * @since PHP 5.1
17  */
18 class AppendIterator implements OuterIterator
19 {
20     /** @internal array of inner iterators */
21     private $iterators;
22
23     /** Construct an empty AppendIterator
24      */
25     function __construct()
26     {
27         $this->iterators = new ArrayIterator();
28     }
29
30     /** Append an Iterator
31      * @param $it Iterator to append
32      *
33      * If the current state is invalid but the appended iterator is valid
34      * the AppendIterator itself becomes valid. However there will be no
35      * call to $it->rewind(). Also if the current state is invalid the inner
36      * ArrayIterator will be rewound und forwarded to the appended element.
37      */
38     function append(Iterator $it)
39     {
```

- IteratorIterator?
- RecursiveIterator?
- RecursiveIteratorIterator?
- RecursiveCallbackFilterIterator?



IteratorIterator

Turns traversable “things” into an iterator

```
$it = new myIterator();
if ($it instanceof \IteratorAggregate) {
  $it = $it->getIterator();
}
$it2 = new \LimitIterator($it, 5, 10);
```

```
$it = new myIterator();
$it2 = new \IteratorIterator($it);
$it3 = new \LimitIterator($it2, 5, 10);
```

Recursive*Iterator

```
$it = new ArrayIterator(
            array(“foo”, “bar”, array(“qux”, “wox”), “baz”));

foreach ($it as $v) {
  print $v . “\n”;
}
```

```
foo
bar
Array
baz
```

```
$it = new RecursiveArrayIterator(
            array(“foo”, “bar”, array(“qux”, “wox”), “baz”));

foreach ($it as $v) {
  print $v . “\n”;
}
```

```
foo
bar
Array
baz
```

```
$it = new RecursiveArrayIterator(
            array(“foo”, “bar”, array(“qux”, “wox”), “baz”));
$it2 = new RecursiveIteratorIterator($it);

foreach ($it2 as $v) {
  print $v . “\n”;
}
```

```
foo
bar
qux
wox
baz
```

"Recursive" iterators add the POSSIBILITY to recursively iterate over data.

You still need to implement it!

RecursiveCallbackFilterIterator

- Enables recursivity
- Is a filter iterator (does not necessarily return all the elements)
- Filters through a callback function.

CachingIterator

2 for the price of 1

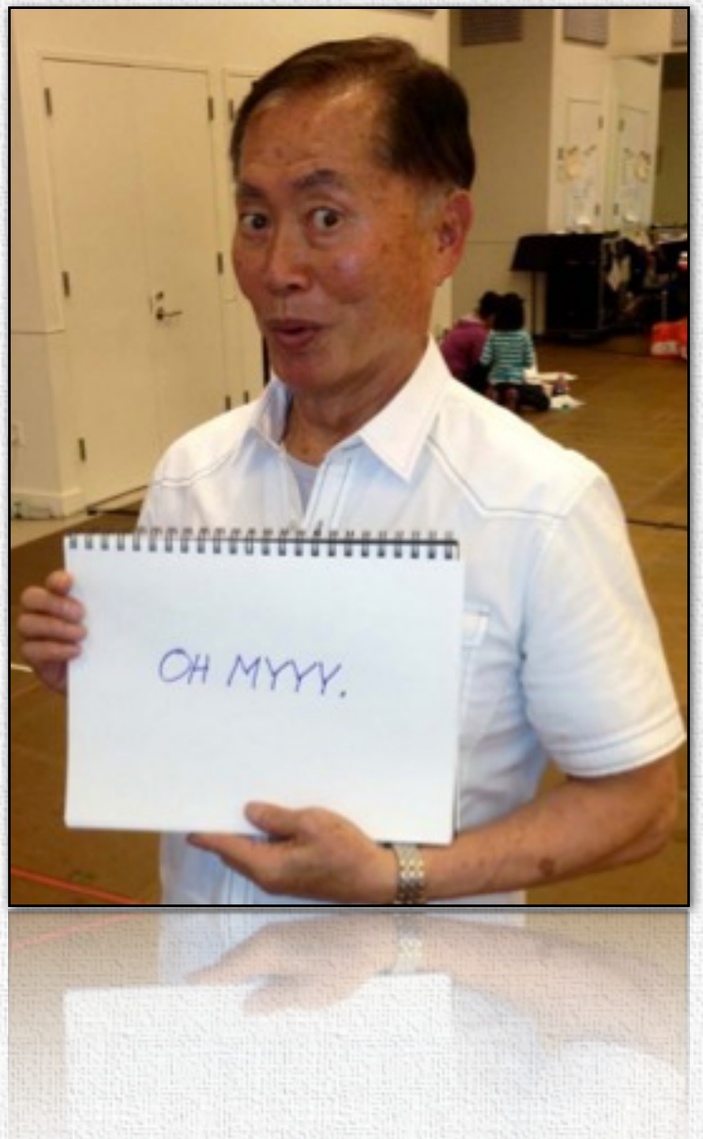
- Lookahead iterator
- Caches values, but not really :(
- Powerful __tostring() functionality (which probably no one uses)

```
$alphaIterator = new ArrayIterator(range("A", "Z"));
$it = new CachingIterator($alphaIterator);

foreach ($it as $v) {
  if (! $it->hasNext()) {
    print "last letter: ";
  }
  print $v . "\n";
}

// A
// ...
// Y
// last letter: Z
```

```
$alphaIterator = new ArrayIterator(range("A", "Z"));
$it = new CachingIterator($alphaIterator);

foreach ($it as $v) {
  if (! $it->hasNext()) {
    print "last letter: ";
  }
  print $v . "\n";
}

print "The fourth letter of the alphabet is: ".$it[3]."\n";
```

- Don’t change cached data (you could, but don’t)
- It doesn’t use the cached data on consecutive calls to the iterator.
- It clears the cache on a rewind() (and thus, a next foreach() loop)

SPL Iterators,..



- It has “quirks” that are easily solvable (but breaks BC)
- Documentation is not always up to date.
- Naming is VERY confusing (caching iterator, recursiveIterator, seekableIterator)
- But the iterators are worth it!

DATA STRUCTURES

- SplDoublyLinkedList
- SplStack
- SplQueue
- SplHeap
- SplMinHeap
- SplMaxHeap
- SplPriorityQueue
- SplFixedArray
- SplObjectStorage

- Every data structure has its strength and weaknesses.
- Big-Oh O(1), O(n), O(log n) etc...
- Balance between time (CPU) and space (memory)
- PHP arrays are quite good!
- But sometimes other data structures are better.

NOX LOGIC

PubQuiz

http://www.flickr.com/photos/sostark/26697448/


SplQueue

SpI(Doubly)LinkedList

http://www.flickr.com/photos/blogchef/4764248456/sizes/z/in/photostream/

http://www.flickr.com/photos/pediddle/54326823/

SplPriorityQueue

http://farm4.static.flickr.com/3224/3072017987_ee454536da.jpg

http://en.wikipedia.org/wiki/File:USA.NM.VeryLargeArray.02.jpg

- Use wisely:
  - Don’t use SplStack / SplQueue for random reads.
  - Don’t use FixedArrays when you need speed boosts.

SplObjectStorage

splObjectStorage as a map

```
$map = new SplObjectStorage();
$map[$obj1] = $info1;
$map[$obj2] = $info2;
print_r ($map[$obj2]);
```

splObjectStorage as a set

```
$set = new SplObjectStorage();
$set->attach($obj1);
print_r ($set->contains($obj1));
```

Defining what to store:

```
class MyStorage extends SplObjectStorage {
  function getHash($object) {
    return $object->type;
  }
}

$obj1 = new StdClass();   $obj1->type = “foo”;
$obj2 = new StdClass();   $obj2->type = “bar”;
$obj3 = new StdClass();   $obj3->type = “foo”;

$store = new MyStorage();
$store->attach($obj1);     // Added
$store->attach($obj2);     // Added
$store->attach($obj3);     // Not added:same type (thus hash) already present!
```

EXCEPTIONS

- BadFunctionCallException
- BadMethodCallException
- DomainException
- InvalidArgumentException
- LengthException
- LogicException
- OutOfBoundsException
- OutOfRangeException
- OverflowException
- RangeException
- RuntimeException
- UnderflowException
- UnexpectedValueException

Logic Exceptions

- BadFunctionCallException
  - BadMethodCallException
- DomainException
- InvalidArgumentException
- LengthException
- OutOfRangeException

Runtime Exceptions

- OutOfBoundsException
- OverflowException
- RangeException
- UnderflowException
- UnexpectedValueException

```
function foo($str) {
  if ($str == “The Spanish Inquisition”) {
    throw new \UnexpectedValueException(“Nobody expects ”.$str);
  }
  ...
}
```

```
function foo($str) {
  if ($str == “The Spanish Inquisition”) {
    throw new \InvalidArgumentException(“Nobody expects ”.$str);
  }
  ...
}
```

Logic, not runtime

```
function foo($str, $int) {
  if (! is_string($str)) {
    throw new \InvalidArgumentException(“Invalid type”);
  }
  if ($int < 0 || $int > 10) {
    throw new \OutOfRangeException(“should be between 0 and 10);
  }
  ...
}
```

Never throw “\Exception”

Always catch “\Exception”

MISC

- SPL Autoloading
- SplFileInfo class
- Spl(Temp)FileObject
- ArrayObject
- SplObserver / SplSubject

Autoloader

```
spl_autoload_register(“spl_autoload_call”);
```

Throws logicException

```
spl_autoload_unregister(“spl_autoload_call”);
```

- Removes ALL the autoloaders!
- Destroys the autoload stack.
- Set your house on fire.

ArrayObject

ArrayObjects are objects that acts like arrays

ArrayObjects are <u>not</u> objects that acts like arrays

```php
$a = array("foo", "bar");
$b = $a;
$b[] = "baz";

print_r ($a);
print_r ($b);
```

```
Array
(
    [0] => foo
    [1] => bar
)
Array
(
    [0] => foo
    [1] => bar
    [2] => baz
)
```

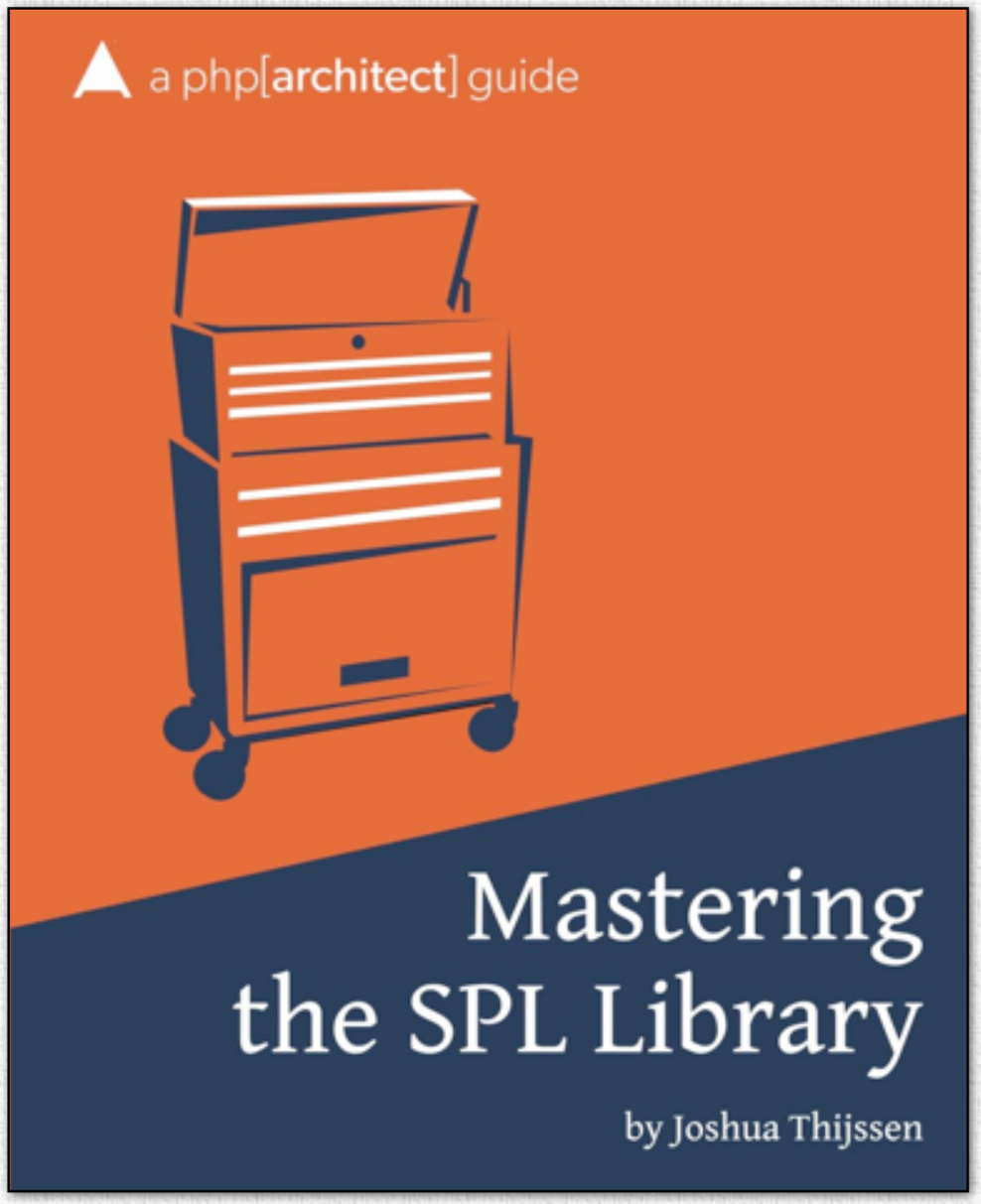
```php
$a = new ArrayObject();
$a[] = "foo";
$a[] = "bar";

$b = $a;
$b[] = "baz";

print_r (iterator_to_array($a));
print_r (iterator_to_array($b));
```

```
Array
(
    [0] => foo
    [1] => bar
    [2] => baz
)
Array
(
    [0] => foo
    [1] => bar
    [2] => baz
)
```

How can we make using the SPL easier?

http://noscope.com/photostream/Albums/various_I_Want_To_Believe_01.jpg

- The (first and only) book about the SPL.
- Written by me (so you know it's good :P)
- Fixes the documentation problem of the SPL (or a wobbly table)



- Adaption of the SPL will only happen when developers can become familiar with it.
- There is currently no real way to familiarize yourself with the SPL.

- BLOG!
- Update documentation.
- Find the quirks (and maybe even solve them).

joind.in

https://joind.in/10699

Find me on twitter: @jaytaph

Find me on email: jthijssen@noxlogic.nl

Find me for blogs: www.adayinthelifeof.nl

Find me for development and training: www.noxlogic.nl