

Oh My!

Iterators, ArrayAccess & Countable

Eli White

Vice President — One for All Events

@eliw



Drinking Game



1st: The SPL (Standard PHP Library)

A standard set of interfaces & classes for PHP5

Designed to solve standard problems
and provide efficient data access.

SPL Features

SPL includes numerous types of features:

- **Data Structures** (Linked Lists, Stacks, Queues, Heaps, ...)
- **Iterators** (w/ Filtering, Caching, Recursion, ...)
- **Various standard objects** (FileObject, ArrayObject, ...)
- **Subject/Observer Interface**
- **Exceptions**
- **and more ...**

This talk ...

Will focus on a core set that are generically useful for all your data access objects.

Iterators, ArrayAccess, Countable

(and maybe a few others)

Why?

Allows your user-space objects to be treated like native 1st class types.

Starting off

You have to start from somewhere ...

Features we want to duplicate

All features built in to arrays

foreach iteration:

```
foreach ($array as $key => $value) {  
    echo "{$key}: {$value}\n";  
}
```

direct item access:

```
echo $array[5];
```

countability:

```
echo count($array);
```


A Basic Class

```
class Set
{
    protected $_set;

    public function __construct(Array $parameters = NULL) {
        $this->_set = $this->_loadFromCache($parameters);
        if ($this->_set === NULL) {
            $this->_set = $this->_loadFromDatabase($parameters);
        }
        if ($this->_set === NULL) {
            $this->_set = [];
        }
    }

    protected function _loadFromCache(Array $parameters = NULL) {
        // Pull data from cache, returning an array of arrays or objects
    }

    protected function _loadFromDatabase(Array $parameters = NULL) {
        // Pull data from DB, returning an array of arrays or objects
    }
}
```

But you need to access the data...

So you need to implement some access method:

```
class SetAccess extends Set
{
    public function getAll() {
        return $this->_set;
    }

    public function get($index) {
        if (array_key_exists($index, $this->_set)) {
            return $this->_set[$index];
        } else {
            return NULL;
        }
    }
}
```

Inelegant solution

Leaves you accessing data in these ways:

```
$myset = new SetAccess();  
print_r($myset->get(3));
```

```
$myset = new SetAccess();  
$all = $myset->getAll();  
foreach ($all as $item) {  
    print_r($item);  
}
```

Iterators

Natively use foreach on your objects

Iterator Interface

5 methods to define, revolve around remembering state:

- `current()`: Returns the current value
- `key()`: Returns the current value's access key
- `next()`: Moves the internal pointer to the next item
- `rewind()`: Needs to reset the internal pointer to the first item
- `valid()`: Returns whether the internal pointer is at a valid data item

```
interface Iterator extends Traversable {  
    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 )  
}
```

Easy to implement with Arrays

```
class SetIterable extends SetAccess implements Iterator
{
    public function current() {
        return current($this->_set);
    }

    public function key() {
        return key($this->_set);
    }

    public function next() {
        next($this->_set);
    }

    public function rewind() {
        reset($this->_set);
    }

    public function valid() {
        return (key($this->_set) !== NULL);
    }
}
```

Now have direct access...

Now we can access the object directly in a foreach loop!

```
$myset = new SetIteratable();  
foreach ($myset as $key => $item) {  
    echo "{$key}: ", print_r($item, true), "<br/>\n";  
}
```

ArrayAccess

Treat your object like it was an array.

ArrayAccess Interface

4 methods to define, to gain direct key access:

`offsetExists()`: Does the provided key exist?

`offsetGet()`: Return the value at provided key

`offsetSet()`: Set the value at the provided key

`offsetUnset()`: Remove the value (and key) provided

```
interface ArrayAccess {  
    abstract public boolean offsetExists ( mixed $offset )  
    abstract public mixed offsetGet ( mixed $offset )  
    abstract public void offsetSet ( mixed $offset , mixed $value )  
    abstract public void offsetUnset ( mixed $offset )  
}
```

Again
easy to
code with
builtins ...

```
class SetArray extends SetIterable implements ArrayAccess
{
    public function offsetExists($offset) {
        return array_key_exists($offset, $this->_set);
    }

    public function offsetGet($offset) {
        return $this->_set[$offset];
    }

    public function offsetSet($offset, $value) {
        if (is_null($offset)) {
            $this->_set[] = $value;
        } else {
            $this->_set[$offset] = $value;
        }
    }

    public function offsetUnset($offset) {
        unset($this->_set[$offset]);
    }
}
```

Treat it like an array...

You can now directly treat the object like an array:

NOTE:

You don't have to implement everything. Create blank `offsetSet()` and `offsetUnset()` if you don't want to allow modification!

```
$myset = new SetArray();  
  
print_r($myset[3]);  
  
if (isset($myset['bob'])) { echo "Smith"; }  
  
$myset['Eli'] = 'White';  
echo '<p>', $myset['Eli'], '</p>';  
unset($myset['Eli']);  
  
$myset[] = [2010, 2011, 2012, 2013, 2014];  
$myset[] = 'php[tek] 2014';
```

Countable

And while we are at it ...

Countable Interface

Just one method:
count(): How many items in this object?

```
class SetCountable extends SetArray implements Countable
{
    public function count() {
        return count($this->_set);
    }
}
```

```
$myset = new SetCountable();
echo count($myset);
```

Return whatever you want though...

Like all these, what you return is up to you!

```
class SetCountable extends SetArray implements Countable
{
    public function count() {
        return count($this->_set, COUNT_RECURSIVE);
    }
}

$myset = new SetCountable();
echo count($myset);
```

Serializable

Another bit of magic...

Serializable Interface

2 methods to let you custom define serialization:
serialize(): Returns a serialized form of your object
unserialize(): Instantiates an object, given the serialized form

```
interface Serializable {  
    abstract public string serialize ( void )  
    abstract public void unserialize ( string $serialized )  
}
```


A simple example, just saving data

```
class SetSerial extends SetArray implements Serializable
{
    public function serialize() {
        return serialize($this->_set);
    }

    public function unserialize($serialized) {
        $this->_set = unserialize($serialized);
    }
}
```

Simple, and serialization works as normal!

```
$myset = new SetSerial();
$myset['magazine'] = 'php[architect]';
$save = serialize($myset);
```

```
$restore = unserialize($save);
echo $restore['magazine'];
```

But you can return whatever...

Only save what data you want.
Encode in whatever format you want:

```
class SetSerialFunky extends SetArray implements Serializable
{
    public function serialize() {
        $copy = array_filter($this->_set, function ($val) { return !is_array($val); });
        return json_encode($copy);
    }

    public function unserialize($serialized) {
        $this->_set = json_decode($serialized, TRUE);
    }
}
```

Putting it all together

So what does this look like...

```

class SetFull implements Iterator, ArrayAccess, Countable, Serializable
{
    // Iterator:
    public function current() { return current($this->_set); }
    public function key() { return key($this->_set); }
    public function next() { next($this->_set); }
    public function rewind() { reset($this->_set); }
    public function valid() { return (key($this->_set) !== NULL); }

    // ArrayAccess:
    public function offsetExists($key) { return array_key_exists($key, $this->_set); }
    public function offsetGet($key) { return $this->_set[$key]; }
    public function offsetUnset($key) { unset($this->_set[$key]); }
    public function offsetSet($key, $value) {
        if (is_null($key)) { $this->_set[] = $value; }
        else { $this->_set[$key] = $value; }
    }

    // Countable:
    public function count() { return count($this->_set); }

    // Serializable
    public function serialize() { return serialize($this->_set); }
    public function unserialize($data) { $this->_set = unserialize($data); }
}

```

Get creative!

This only scrapes the surface of what is possible!

None of the methods need to return 'basic' information like this.

Get as creative as needed for your situation!

More Iterator Fun!

If we have time, let's play!

InfiniteIterator

Causes an iterator to automatically rewind:

```
$forever = new InfiniteIterator(new SetFull());  
$count = 100;  
foreach ($forever as $item) {  
    print_r($item);  
    if (!($count--)) break;  
}
```

LimitIterator

Allows you to set a start index & max iterations:

```
foreach (new LimitIterator(new SetFull(), 0, 3) as $item) {  
    print_r($item);  
}
```

```
$forever = new InfiniteIterator(new SetFull());  
foreach (new LimitIterator($forever, 0, 100) as $item) {  
    print_r($item);  
}
```


FilterIterator

Apply your own filtering to what items are returned

```
class ArrayFilter extends FilterIterator
{
    public function accept() {
        return is_array($this->getInnerIterator()->current());
    }
}

foreach (new ArrayFilter(new SetFull()) as $item) {
    print_r($item);
}
```

RegexIterator

Predefined instance of FilterIterator with regex

```
$regex = new RegexIterator(new SetFull(), '/^tek[0-9]+/', RegexIterator::MATCH);  
foreach ($regex as $item) {  
    print_r($item);  
}
```

Lots of options/flags:

- RegexIterator::MATCH
- RegexIterator::GET_MATCH
- RegexIterator::ALL_MATCHES
- RegexIterator::SPLIT
- RegexIterator::REPLACE
- RegexIterator::USE_KEY

MultipleIterator

Allows iterating over multiple iterators at once

Stops whenever any one runs out of items

```
$multiple = new MultipleIterator();
$multiple->attachIterator(new SetFull());
$multiple->attachIterator(new SetFull(['other', 'parameters']));

foreach ($multiple as $both) {
    $setOne = print_r($both[0], TRUE);
    $setTwo = print_r($both[1], TRUE);
    echo "One: {$setOne} | Two: {$setTwo} <br/> \n";
}
```

RecursiveIterator & RecursiveIteratorIterator

2 methods to you define to allow recursion:

hasChildren(): Does the current item have any children?
getChildren(): If so, return a RecursiveIterator to iterate them.

```
class SetRecursable extends SetFull implements RecursiveIterator {
    public function hasChildren() {
        return is_array(current($this->_set));
    }

    public function getChildren() {
        return new RecursiveArrayIterator(current($this->_set));
    }
}

foreach (new RecursiveIteratorIterator(new SetRecursable()) as $item) {
    echo " {$item} ";
}
```

And so much more...



Pardon a brief commercial interruption



PHP[TEK] 2017

The Premier PHP Conference
12th Annual Edition

MAY 24-26 — ATLANTA

tek.phparch.com

For this presentation & more:
eliw.com

Twitter: @EliW

One for All Events:
www.oneforall.events



*One for All
Events*