



# SPL Iterators

Elliott White III – *Eli*

<http://eliw.com/>

# *What is SPL?*

## Standard PHP Library

A collection of standard interfaces & classes designed to solve standard problems and implement efficient data access as part of the standard PHP installation.

# *What an Iterator?*

A design pattern that is a generic solution to the problem of iterating over data in a consistent manner.

# *Basic Interface*

When using/creating an Iterator, you have the following methods available to you:

- `current()` - Returns the current element
- `key()` - Returns the key of the current element
- `next()` - Advances the to the next element & returns it
- `rewind()` - Rewinds to the first element in the collection
- `valid()` - Whether the current position is valid

# *But really what you wanna do...*

The real power in doing all this?

The ability to use foreach on your objects.

# *A basic example: Directory Access*

Traditional method of listing a directory:

```
<?php
$dir = opendir(".");
while ($entry = readdir($dir) !== false) {
    echo "<p>{$entry}</p>";
}
closedir($dir);
?>
```

# *A basic example: Directory Access*

Object method of listing a directory:

```
<?php
$dir = dir('.');
while (false !== ($entry = $dir->read())) {
    echo "<p>{$entry}</p>";
}
$dir->close();
?>
```

# *A basic example: Directory Access*

Using Iterator and foreach:

```
<?php
$dir = new DirectoryIterator('.');
foreach ($dir as $entry) {
    echo "<p>{$entry}</p>";
}
unset($dir); // Release the resource
?>
```

# *A basic example: Directory Access*

But wait, there is more, it's returning Objects!

```
<?php
$dir = new DirectoryIterator('.');
foreach ($dir as $entry) {
    echo "<p>{$entry} - {$entry->getSize()} -
        {$entry->getType()}</p>";
}
unset($dir); // Release the resource
?>
```

# *Some Benefits*

So what have we seen?

- Consistent interface with foreach, next(), etc.
- Building additional functionality than a basic loop

What else can this do?

- Extend Iterators to do what you need
- Chain Iterators: An Iterator iterating an Iterator

# *Extending & Chaining Iterators*

Let's start with the most basic Iterator: ArrayIterator

```
<?php
$arr = array('Mimi', 'Wiley', 'Tux', 'Taz', 'Iyi', 'Yowler');
$arrIt = new ArrayIterator($arr);
foreach ($arrIt as $entry) {
    echo "<p>{$entry}</p>";
}
?>
```

# *Extending & Chaining Iterators*

Now we want to filter those entries.

## Use FilterIterator

An Iterator prototype that has one method you must define, `accept()`, which returns a boolean value denoting if you wish to accept that entry or not.

(Actually a form of `OuterIterator`, which is an Iterator wrapper, and has the same methods as an Iterator, but also has `getInnerIterator()` to wrap the Iterator with)

# Extending & Chaining Iterators

## Implementation of a Regex based FilterIterator:

```
class RegexFilterIterator extends FilterIterator {
    private $criteria; // Holds the filter criteria
    public function __construct(Iterator $i, $regex) {
        parent::__construct($i);
        $this->criteria = $regex;
    }
    public function accept() {
        return preg_match($this->criteria, parent::current());
    }
}
```

# *Extending & Chaining Iterators*

Use our new RegexFilterIterator to only see names with 'y'

```
$arr = array('Mimi', 'Wiley', 'Tux', 'Taz', 'Iyi', 'Yowler');  
$it = new RegexFilterIterator(new ArrayIterator($arr), '/y/i');  
  
foreach ($it as $entry) {  
    echo "{$entry}<br />\n";  
}
```

# *Building your own Iterator*

As an example, let's build a new version of ArrayIterator, as the default one doesn't return Objects, but just values.

Meaning you can't use the foreach access method if you need the keys as well as values.

# *Building your own Iterator*

```
class KeyArrayIterator implements Iterator {
    protected $data;

    public function __construct($v) {
        $this->data = $v;
    }
    public function rewind() {
        return reset($this->data);
    }
    public function next() {
        return next($this->data);
    }
}
```

# *Building your own Iterator*

```
public function valid() {
    return (key($this->data) === NULL) ? false : true;
}

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

public function current() {
    return new KeyArrayItem(key($this->data),
        current($this->data));
}
}
```

# *Building your own Iterator*

```
class KeyArrayItem {
    public $key;
    public $value;

    public function __construct($k, $v) {
        $this->key = $k;
        $this->value = $v;
    }

    public function __toString() {
        return (string) $this->value;
    }
}
```

# *Building your own Iterator*

Now use it to display both key & value:

```
<?php
require 'KeyArrayIterator.php';
$arr = array('Owen' => 'P', 'Time' => 'D', 'Kevin' => 'B');

$arrIt = new KeyArrayIterator($arr);

foreach ($arrIt as $entry) {
    echo $entry->key, ' - ', $entry, "<br />\n";
}
?>
```

# *Topics to explore*

## LimitIterator

- An OuterIterator like FilterIterator, but that uses a start and stop index to only get a slice of data.

## InfiniteIterator

- Automatically calls rewind() at the end

## RecursiveIterator

- Adds methods getChildren() and hasChildren() to allow for recursive parsing of hierarchies.

# *Blow your mind*

## IteratorIterator

- Takes any class with Iterator methods defined and returns a new Iterator wrapped around it.

## RecursiveIteratorIterator

- Wraps a RecursiveIterator implementing class with an Iterator that will automatically dive into each child.

# *Look at the power!*

Ever want a recursive directory listing in 2 lines?

```
<?php
$rdir = new RecursiveIteratorIterator(
    new RecursiveDirectoryIterator('.') );
foreach ($rdir as $entry) { echo "{$entry}<br />\n"; }
?>
```

# *And so much more...*

Far more info than you ever wanted:

The official SPL docs:  
<http://php.net/book.spl>

The REAL SPL docs:  
<http://www.php.net/~helly/php/ext/spl/>

# *Digg is Hiring!*

Digg.com is hiring PHP programmers!

<http://digg.com/jobs>  
[jobs@digg.com](mailto:jobs@digg.com)

# Any Questions?

- For this presentation and more:

<http://eliw.com/>

<http://digg.com/users/EliW>

The Digg logo, consisting of the word "digg" in a white, lowercase, sans-serif font, centered within a blue rectangular background.