PHPDiff Documentation

Release 1.0

Máté Bartus

Jul 30, 2017

Contents

1 Two-way diff **3**
- 1.1 Examples . . . 3

2 Three-way diffs **5**
- 2.1 Weave merge . . . 5
- 2.2 Three-way merge . . . 5
- 2.3 Examples . . . 5

3 Diff structure **7**
- 3.1 Diff change blocks . . . 7

4 Longest Common Subsequence **9**

5 Custom comparison **11**
- 5.1 Example . . . 11

6 Custom Sequencing strategy **13**
- 6.1 Example . . . 13

7 Indices and tables **15**

Contents:

CHAPTER 1

Two-way diff

Two-way diffs are the simplest diffs there are. It simply produces a change set between two documents. You can create two-way diffs by using the `Differ` class.

Examples

The example below shows a how to create a simple diff between two documents:

```
use CHItA\PHPDiff\DifferBase;
use CHItA\PHPDiff\Differ;

$differ = new Differ();
$diff = $differ->diff(
    array('a', 'b', 'b', 'a', 'c', 'b', 'b', 'a'),
    array('a', 'b', 'a', 'c', 'c', 'c', 'c', 'a', 'd')
);

// $diff will contain an array with the following structure:
//
// array(
//     array(
//          'type' => DifferBase::UNCHANGED,
//          array('a', 'b')
//     ),
//     array(
//          'type' => DifferBase::REMOVED,
//          array('b')
//     ),
//     array(
//          'type'  => DifferBase::UNCHANGED,
//          array('a', 'c')
//     ),
//     array(
//          array(
```

```
//              'type'  => DifferBase::REMOVED,
//              array('b', 'b')
//          ),
//          array(
//              'type'  => DifferBase::ADDED,
//              array('c', 'c', 'c')
//          )
//      ),
//      array(
//          'type'  => DifferBase::UNCHANGED,
//          array('a')
//      ),
//      array(
//          'type'  => DifferBase::ADDED,
//          array('d')
//      )
// )
```

Note: For more information about the diff structure generated by the library, please read the *Diff structure section*.

CHAPTER 2

Three-way diffs

Three-way diffs are the set of changes between a parent and two child documents. There are multiple algorithms for computing three-way diffs which are producing different results. PHPDiff implements the **Weave merge** and **Three-way merge** algorithms out of the box, while providing an interface for creating other implementations as well.

Weave merge is the **default** three-way merge implementation provided by this library, however, it can be easily changed either by providing an other three-way diff algorithm implementation in `Differ3`'s constructor, or by calling the setter function on that class.

Description of the algorithms can be found on Wikipedia <https://en.wikipedia.org/wiki/Merge_(version_control)>.

Weave merge

Weave merge is a simple algorithm, which produces a diff that contains all units that are present in both modified versions of the files, and none that were deleted from either of them.

This algorithm only produces a merge conflict when the order of the units in the merged documents cannot be determined.

Three-way merge

Three-way merge is maybe the most common three-way merge algorithm out there. It basically produces an output which contains all the changes that are present in either of the modified versions of the document, however, when both documents containing changes at the same place, it produces a conflict.

Examples

Simple three-way diff example, using the three-way merge algorithm:

```
use CHItA\PHPDiff\Diff3Algorithm\ThreeWayMerge;
use CHItA\PHPDiff\Differ;
use CHItA\PHPDiff\Differ3;
use CHItA\PHPDiff\DifferBase;
use CHItA\PHPDiff\LongestCommonSubsequence\Algorithm\Hirschberg;

$differ = new Differ3(
    new ThreeWayMerge(new Differ(null, new Hirschberg()))
);
$diff = $differ->diff(
    array('a', 'b', 'c', 'd', 'e', 'f'), // Original document
    array('a', 'b', 'w', 'x', 'y', 'e'), // Modified 1
    array('a', 'q', 'r', 's', 'b', 'f')  // Modified two
);

// $diff will contain an array with the following structure:
//
// array(
//     array(
//        'type' => DifferBase::UNCHANGED,
//        array('a')
//     ),
//     array(
//        'type' => DifferBase::ADDED,
//        array('q', 'r', 's')
//     ),
//     array(
//        'type' => DifferBase::UNCHANGED,
//        array('b')
//     ),
//     array(
//         array(
//            'type' => DifferBase::REMOVED,
//            array('c', 'd', 'e', 'f')
//         ),
//         array(
//            'type' => DifferBase::ADDED,
//            array('w', 'x', 'y')
//         )
//     )
// )
```

Note: For more information about the diff structure generated by the library, please read the *Diff structure section*.

CHAPTER 3

Diff structure

The library returns diffs in an array. These arrays contain the changes, which are grouped by the type of the changes in the following format:

```
array(
    $change1,
    $change2,
)
```

Note: Where `$changeN` can be any of the change blocks listed below.

Diff change blocks

Unchanged block is used when all of the two (or three) documents have the same units in common.

```
array(
    'type' => DifferBase::UNCHANGED,
    array('all', 'common', 'units', 'until', 'the', 'next', 'change type')
)
```

Added block is used when the original version did not contain some units that are present in the modified version(s).

```
array(
    'type' => DifferBase::ADDED,
    array('all', 'common', 'units', 'until', 'the', 'next', 'change type')
)
```

Removed block is used when the original version did contain some units that are not present in the modified version(s).

```
array(
    'type' => DifferBase::REMOVED,
```

```
    array('all', 'common', 'units', 'until', 'the', 'next', 'change type')
)
```

Edit block is used when the modified version(s) contain both additions and deletions compared to the original version.

```
array(
   array(
      'type' => DifferBase::REMOVED,
      array('removed', 'units')
   ),
   array(
      'type' => DifferBase::ADDED,
      array('added', 'units')
   )
)
```

Conflict block is used when a merge conflict cannot be resolved. This could only occur in three-way diffs. Note that because these units are neither present nor removed from the document, in general no removed lines are returned before this block (even where some units are removed from both modified versions).

```
array(
   'type' => DifferBase::CONFLICT,
   array('conflicting', 'units', 'in', 'version1'),
   array('conflicting', 'units', 'in', 'version2'),
)
```

CHAPTER 4

Longest Common Subsequence

To generate diffs we solve the longest common subsequence problem for the documents to determine which lines are the ones that did not changed.

This library provides two implementations out of the box for the longest common subsequence problem, one that is time efficient (dynamic programming approach) and another that is memory efficient (Hirschberg's algorithm).

There is also an option to implement a strategy that selects the solver based on the inputs. For this, you need to implement the `CHItA\PHPDiff\LongestCommonSubsequence\Strategy\StrategyInterface`.

CHAPTER 5

Custom comparison

By implementing `CHItA\PHPDiff\Comparison\ComparisonInterface`, you may add a custom data comparison implementation to the algorithm. This could be useful if you would like to compare the trimmed versions of the units for example.

> **Warning:** Comparison algorithm never alters the content of the passed elements. Any manipulation of the input data in the comparison algorithm will not be present in the output.

However, please be aware that when you use this option, the returned units from two-way diffs will contain the units from the modified version (second parameter of the Differ::diff() method). Please also note, that when using custom comparisons with three-way diffs, the units in the output could be from either of the modified documents.

Example

An example comparison algorithm that trims whitespaces from the end of the units before comparing them.

```
use CHItA\PHPDiff\Comparison\ComparisonInterface;

class TrimCompare implements ComparisonInterface
{
    public function compare($value1, $value2)
    {
            return rtrim($value1) === rtrim($value2);
    }
}
```

```
use CHItA\PHPDiff\DifferBase;
use CHItA\PHPDiff\Differ;

$differ = new Differ();
$differ->setComparisonAlgorithm(new TrimCompare());
```

```
$diff = $differ->diff(
    array('a', 'b', 'b', 'c'),
    array(' a', 'b  ', 'b   ', 'c')
);

// $diff will contain an array with the following structure:
//
// array(
//     array(
//          array(
//              'type'  => DifferBase::REMOVED,
//              array('a')
//          ),
//          array(
//              'type'  => DifferBase::ADDED,
//              array(' a')
//          )
//     ),
//     array(
//          'type'  => DifferBase::UNCHANGED,
//          array('b  ', 'b   ', 'c')
//     )
// )
```

CHAPTER 6

Custom Sequencing strategy

By default, the diff algorithms expect arrays as the input, in which case it is assumed, that each element of the array is a unit (an element of the sequence and a single letter of the alphabet which the sequence is generated from).

However, you can implement `CHItA\PHPDiff\SequencingStrategy\SequencingStrategyInterface` and handle any type of input yourself.

Example

In the example below, the sequence is generated from a string where the units are the bytes of the string (ascii characters).

```
use CHItA\PHPDiff\SequencingStrategy\SequencingStrategyInterface;

class MySequencer implements SequencingStrategyInterface
{
    public function getSequence($dataSet)
    {
        return str_split($dataSet);
    }
}
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search