Writers

Write nested arrays in place, create levels, remove keys, append to lists, merge values and defaults, and write in a scope of the array

Anonymous · 05/11/2026

ArrayWriter

`ArrayWriter` writes the keys of an array in place, as many times as needed. It is the counterpart of `ArrayReader`: it takes keys the same way and reports errors with the same messages and paths.

```
use Derafu\Array\ArrayWriter;

$data = ['order' => ['totals' => ['net' => 1000]]];
$writer = ArrayWriter::fromReference($data);

$writer->set('order.id', 10);
$writer->getWriter('order.totals')->set('total', 1190);

// $data is now:
// ['order' => ['totals' => ['net' => 1000, 'total' => 1190], 'id' => 10]]
```

Creating a Writer

Factory	Input	Behavior
`fromReference(array &$data, ?string $source = null)`	A variable holding an array.	Creates a writer at the root that writes in that variable.
`new ArrayWriter($data, $path = '', $source = null)`	A variable holding an array.	Creates a writer with an explicit path and source.

The writer does not copy the array: every write changes the variable it was created from. `source` tells whoever reads an error which array it is (a file, a request, a document), as in `ArrayReader::fromArray()`, and `path` says where the array is inside a bigger one.

`toArray()` returns the current array.

Writing

Method	Behavior
`set($key, $value)`	Sets a value, creating the levels that are missing.
`remove($key)`	Removes a key. If the path does not exist, nothing happens.
`append($key, $value)`	Adds a value at the end of the list at the key. The list is created if the key is absent or `null`.
`merge($values)`	Merges recursively: the values replace what is already there.
`mergeDefaults($defaults)`	Merges recursively: what is already there is kept and the defaults only fill what is missing.

All of them return the writer, so calls can be chained. Values are not typed when written: `null` is stored as `null`, and the type is what the reader asks for when the value is read.

```
$data = ['items' => ['a']];
$writer = ArrayWriter::fromReference($data);

$writer
    ->append('items', 'b')
    ->append('tags', 'new')
    ->set('meta.owner', 'Alice')
    ->remove('items.0');

// ['items' => [1 => 'b'], 'tags' => ['new'], 'meta' => ['owner' => 'Alice']]
```

Keys

Keys follow the rules of the reader and of `ArrayPath`: an existing literal key matching the complete string takes precedence; otherwise dots separate nested levels.

```
$data = ['profile.name' => 'Alice'];
$writer = ArrayWriter::fromReference($data);

$writer->set('profile.name', 'Bob'); // writes the literal key
// ['profile.name' => 'Bob']
```

Levels

`set()`, `append()` and `getWriter()` create the levels of the path that are absent or `null` as arrays. A level that exists and is not an array is an error: it is not replaced.

```
$data = ['order' => ['id' => 33]];
ArrayWriter::fromReference($data, 'document')->set('order.id.x', 1);
// TypeMismatchException:
```

```
// The key "order.id" of document must be of type array, int given.
```

`append()` also requires the value at the key to be a list. `remove()` never fails: a path that does not exist, or that goes through a value that is not an array, has nothing to remove.

Merging

`merge()` and `mergeDefaults()` use the semantics of `array_replace_recursive()`: nested arrays are merged by key, lists are merged by index and not concatenated, and `null` counts as a value.

```
$data = ['a' => ['x' => 1, 'y' => 2]];
$writer = ArrayWriter::fromReference($data);

$writer->merge(['a' => ['y' => 20, 'z' => 30]]);
// ['a' => ['x' => 1, 'y' => 20, 'z' => 30]]

$writer->mergeDefaults(['a' => ['x' => 0, 'w' => 4]]);
// ['a' => ['x' => 1, 'w' => 4, 'y' => 20, 'z' => 30]]
```

Unlike `set()`, they do not fail when a scalar meets an array: the value that is merged in replaces it in `merge()`, and the existing value is kept in `mergeDefaults()`.

Writing in a Scope

`getWriter($key)` returns a writer over the array at a key. What it writes is written in the array it comes from. The levels that are missing are created as arrays, so, unlike `ArrayReader::getReader()`, it never returns `null`.

```
$data = [];
$writer = ArrayWriter::fromReference($data);

$totals = $writer->getWriter('order.totals');
$totals->set('net', 1000);
$totals->mergeDefaults(['tax' => 0, 'exempt' => 0]);

// ['order' => ['totals' => ['net' => 1000, 'tax' => 0, 'exempt' => 0]]]
```

The writer of a scope knows its path from the root, so its errors and the reader it gives say the full path (`order.totals.net`).

The writer of a scope keeps a PHP reference to its level. While it is alive, copies of the array it writes in share that level.

Reading What Was Written

The writer does not read. `getReader()` returns an `ArrayReader` over the current data of the writer, with its path and source. It is a snapshot: ask for a new one after writing to read what was written.

```
$data = ['order' => ['total' => '1190']];
$writer = ArrayWriter::fromReference($data);

$writer->set('order.paid', true);
$writer->getReader()->requireAsInt('order.total'); // 1190
$writer->getReader()->requireBool('order.paid');   // true
```

Writer or ArrayPath

`ArrayPath::set()` is for a single write: it returns a new array and leaves its input unchanged. Use `ArrayWriter` when an array is written several times, or when the writes happen at different levels of the same array.

Last updated on 05/11/2026