

Paths

Read and update nested arrays, transform dot notation and wrap values in lists

Anonymous · 26/10/2026

ArrayPath

`ArrayPath` provides static operations over nested arrays. The transformations return arrays without modifying their input.

```
use Derafu\Array\ArrayPath;

$data = ['customer' => ['name' => 'Alice', 'note' => null]];
ArrayPath::get($data, 'customer.name'); // 'Alice'
ArrayPath::get($data, ['customer', 'note'], 'default'); // null
ArrayPath::get($data, 'customer.missing', 'default'); // 'default'

$updated = ArrayPath::set($data, 'customer.address.city', 'Santiago');
// The original $data is unchanged.
```

Reading and Setting Paths

`get($array, $path, $default = null)`, `set($array, $path, $value)` and `wrap($array, $path)` accept a dot-separated string or a non-empty list of string segments.

`get()` returns a stored `null` unchanged. Its default is used when a key is absent or an intermediate level is not an array. `set()` creates missing levels and replaces non-array intermediate values with arrays.

For a string path, all three methods first use an existing literal key matching the complete string. Only when that key is absent do dots separate nested levels. A literal key with a null value still takes precedence.

```
$data = [
    'profile.name' => 'Alice',
    'profile' => ['name' => 'Nested'],
];

ArrayPath::get($data, 'profile.name'); // 'Alice'

$updated = ArrayPath::set($data, 'profile.name', 'Bob');
// ['profile.name' => 'Bob', 'profile' => ['name' => 'Nested']]
```

```
ArrayPath::get($updated, 'profile.name'); // 'Bob'
```

When the literal key is absent, `set()` creates or updates the nested path. Its interpretation therefore depends on the existing keys, including when starting from an empty array:

```
ArrayPath::set([], 'profile.name', 'Bob');
// ['profile' => ['name' => 'Bob']]
```

Literal matching applies to the complete string at the supplied array's root. After falling back to navigation, dots separate individual segments. Segment lists describe an explicit sequence of keys, including intermediate keys containing dots:

```
ArrayPath::get(['a.b' => ['c' => 1]], ['a.b', 'c']); // 1
ArrayPath::get(['items' => [['id' => 7]]], 'items.0.id'); // 7
```

`get()` returns `mixed`; use `ArrayReader` or `ValueType` when values must be validated.

Flat and Nested Forms

```
ArrayPath::flatten(['a' => ['b' => 1, 'c' => [2, 3]]]);
// ['a.b' => 1, 'a.c.0' => 2, 'a.c.1' => 3]

ArrayPath::flatten(['b' => 1], 'a');
// ['a.b' => 1]

ArrayPath::expand(['a.b' => 1, 'a.c' => 2, 'd' => 3]);
// ['a' => ['b' => 1, 'c' => 2], 'd' => 3]
```

`expand()` keeps integer keys at the current level and replaces conflicting scalar levels when a later path needs an array. `flatten()` emits leaf values; empty nested arrays produce no entry. Dots in literal keys are not escaped. These transformations are not a lossless round trip for every possible array.

Wrapping a Value

`wrap()` places a value in a one-element list unless it is already a list. It uses `array_is_list()`: having a key `0` alone does not make an array a list.

```
ArrayPath::wrap(['items' => ['name' => 'Alice']], 'items');
// ['items' => [['name' => 'Alice']]]

ArrayPath::wrap(['items' => [0 => 'a', 2 => 'b']], 'items');
// ['items' => [[0 => 'a', 2 => 'b']]]
```

```
ArrayPath::wrap(['document.reference' => ['id' => 123]], 'document.reference');  
// ['document.reference' => [['id' => 123]]]
```

As with `get()` and `set()`, an existing complete literal key takes precedence over the nested path. An empty array and an existing list are unchanged. A missing path, a null value at the target, or a non-array intermediate level is left unchanged. Assign the returned array when applying a transformation.

Last updated on 26/10/2026