

Readers

Optional and required typed reads, null handling, integer keys, nested readers and sparse collections

Anonymous · 26/10/2026

ArrayReader

```
use Derafu\Array\ArrayReader;  
use Derafu\Value\ValueType;  
  
$reader = ArrayReader::fromArray(['name' => 'Alice', 'enabled' => false]);  
$reader->requireString('name'); // 'Alice'  
$reader->getBool('enabled', true); // false
```

Creating a Reader

Factory	Input	Behavior
<code>fromArray(array \$data, ?string \$source = null)</code>	An array.	Creates a reader at the root.
<code>fromValue(mixed \$data, ?string \$source = null, string \$path = '')</code>	Untrusted input.	Validates that it is an array, without converting it.
<code>new ArrayReader(\$data, \$path = '', \$source = null)</code>	An array.	Creates a reader with an explicit path and source.

`toArray()` returns the stored array. Changing the original array after creating a reader does not change the reader's array values.

Optional and Required Values

A missing key or a value of `null` uses the optional getter's default. A present value of another type throws. Falsy values such as `false`, `0`, `''` and `[]` are preserved when they match the requested type.

Optional method	Required method	Value
<code>getString(\$key, \$default = null)</code>	<code>requireString(\$key)</code>	String.
<code>getNonEmptyString(\$key)</code>	<code>requireNonEmptyString(\$key)</code>	String containing at least one non-whitespace character; returned without trimming.

Optional method	Required method	Value
<code>getInt(\$key, \$default = null)</code>	<code>requireInt(\$key)</code>	Integer.
<code>getFloat(\$key, \$default = null)</code>	<code>requireFloat(\$key)</code>	Float; stored integers are rejected.
<code>getNumber(\$key, \$default = null)</code>	<code>requireNumber(\$key)</code>	Integer or float.
<code>getBool(\$key, \$default = null)</code>	<code>requireBool(\$key)</code>	Boolean.
<code>getDateTime(\$key)</code>	<code>requireDateTime(\$key)</code>	<code>DateTimeInterface</code> ; see Value Types .
<code>getObject(\$key, \$class)</code>	<code>requireObject(\$key, \$class)</code>	Instance of the class or interface; the return type is that class.
<code>getArray(\$key)</code>	<code>requireArray(\$key)</code>	Any array.
<code>getMap(\$key)</code>	<code>requireMap(\$key)</code>	Array with string keys.
<code>getList(\$key)</code>	<code>requireList(\$key)</code>	Zero-based consecutive list.
<code>getStringList(\$key)</code>	<code>requireStringList(\$key)</code>	List of strings.
<code>getIntList(\$key)</code>	<code>requireIntList(\$key)</code>	List of integers.
<code>getScalarList(\$key)</code>	<code>requireScalarList(\$key)</code>	List of booleans, numbers or strings.
<code>getArrayOf(\$key, \$type)</code>	<code>requireArrayOf(\$key, \$type)</code>	Array whose values match a type; keys are preserved.
<code>getMapOf(\$key, \$type)</code>	<code>requireMapOf(\$key, \$type)</code>	String-keyed array whose values match a type.
<code>getListOf(\$key, \$type)</code>	<code>requireListOf(\$key, \$type)</code>	List whose values match a type.

Optional methods return `null` when there is no value or default. The primitive getters with a non-null default have a non-null return type for PHPStan. Required methods reject missing keys and `null`, including when the requested `ValueType` is nullable.

The generic methods accept any [composable type](#):

```
$reader = ArrayReader::fromArray(['mode' => 'fast']);
$reader->get('mode', ValueType::literal('fast', 'safe'), 'safe');
$reader->require('mode', ValueType::string());
$reader->get('tags', ValueType::listOf(ValueType::string()), []);
```

Defaults are returned as supplied, not validated or converted by the requested type.

Explicit Conversions

Strict getters validate existing types. The `getAsXXX()`/`requireAsXXX()` methods explicitly convert through `ValueType` and `ValueCaster` from [Derafu Value](#).

Optional method	Required method	Converted type
<code>getAsInt(\$key, \$default = null)</code>	<code>requireAsInt(\$key)</code>	Integer.
<code>getAsFloat(\$key, \$default = null)</code>	<code>requireAsFloat(\$key)</code>	Float.
<code>getAsNumber(\$key, \$default = null)</code>	<code>requireAsNumber(\$key)</code>	Integer when possible, otherwise float.
<code>getAsBool(\$key, \$default = null)</code>	<code>requireAsBool(\$key)</code>	Boolean.
<code>getAsString(\$key, \$default = null)</code>	<code>requireAsString(\$key)</code>	String.

```
use Derafu\Array\ArrayReader;

$reader = ArrayReader::fromArray(['port' => '5432', 'enabled' => 'off', 'label' => false]);
$reader->requireAsInt('port');           // 5432
$reader->requireAsBool('enabled');       // false
$reader->requireAsString('label');       // '0'
$reader->getAsNumber('missing', 0);      // 0
```

Missing keys and null retain the reader's existing default/required rules. Present values that cannot be converted throw; they do not use the default. See [ValueCaster's conversion contracts](#).

Presence and Allowed Keys

Every read accepts an integer or string key. An existing literal key takes precedence; otherwise, dot notation such as `'connection.port'` navigates nested arrays.

Method	Meaning
<code>hasKey(\$key)</code>	The key exists, even when its value is <code>null</code> .
<code>hasValue(\$key)</code>	The key exists and its value is not <code>null</code> .
<code>assertOnlyKeys(...\$keys)</code>	Throws for any key not allowed; returns the same reader. It does not require the allowed keys to exist.

```
$reader = ArrayReader::fromArray([0 => null, 'enabled' => false]);
$reader->hasKey(0); // true
$reader->hasValue(0); // false
$reader->hasValue('enabled'); // true
$reader->assertOnlyKeys(0, 'enabled');
```

Integer allowed keys and their equivalent string forms are accepted by `assertOnlyKeys()`. This method checks actual keys at the current level; allowed names are literal and are not expanded into paths. To check nested keys, obtain a nested reader and call `assertOnlyKeys()` on it.

Nested Readers and Collections

`getReader()` and `requireReader()` read any nested array as another reader: maps, lists and sparse arrays all work. An optional second argument validates the selected array with a `ValueType` before creating its reader. Omitting it, or passing `null`, accepts any array as before.

```
$reader = ArrayReader::fromArray(['authors' => [['Alice', 'alice']]]);
$author = $reader->requireReader('authors')->requireReader(0);
$author->requireString(0); // 'Alice'
$author->getString(1); // 'alice'
```

Optional	Required	Outer collection
<code>getReaders(\$key)</code>	<code>requireReaders(\$key)</code>	A list of arrays, returned as a list of readers.
<code>getArrayReaders(\$key)</code>	<code>requireArrayReaders(\$key)</code>	An array of arrays, preserving arbitrary keys.

Each record can itself be any array. The collection methods also accept an optional second `ValueType` argument: it validates each record without changing the outer collection's requirements.

`getReaders()` and `requireReaders()` still require a consecutive list; `getArrayReaders()` and `requireArrayReaders()` preserve arbitrary keys.

Optional methods return `null` for absent or null values without invoking the type. Required methods throw `MissingValueException` for those values before invoking it. Empty collections return `[]`.

```
use Derafu\Array\ArrayReader;
use Derafu\Value\ValueType;

$reader = ArrayReader::fromArray([
    'innerJoin' => ['table' => 'stock', 'condition' => 'products.id =
stock.product_id'],
    'questions' => [['text' => 'What is PHP?']],
]);

$join = $reader->requireReader('innerJoin', ValueType::map());
$table = $join->requireString('table'); // 'stock'.
$questions = $reader->requireReaders('questions', ValueType::map());
$text = $questions[0]->requireString('text'); // 'What is PHP?'.
```

The supplied type must produce an array. Examples include `ValueType::map()`, `ValueType::list()`, `ValueType::arrayOf()` and `ValueType::mapOf()`. Scalar-producing types are not valid arguments. A map requires string keys; a positional list is rejected when a map is requested.

Each selected array is parsed once during reader creation, and the parsed result becomes the child reader's data. Explicit conversion types can therefore prepare a reader's values without modifying the parent reader. Both validation errors and subsequent child reads retain the full path and source.

```
use Derafu\Array\ArrayReader;
use Derafu\Value\ValueType;

$reader = ArrayReader::fromArray([
    'translations' => [
        'es' => ['save' => 'Guardar'],
        'en' => ['save' => 'Save'],
    ],
]);

$languages = $reader->requireArrayReaders(
    'translations',
    ValueType::mapOf(ValueType::string()),
);
$label = $languages['es']->requireString('save'); // 'Guardar'.
```

```
$reader = ArrayReader::fromValue(
    ['rows' => [3 => ['name' => 'Alice'], 8 => ['name' => 123]]],
    source: 'account API',
    path: 'response',
);

$rows = $reader->requireArrayReaders('rows');
$rows[3]->requireString('name'); // 'Alice'
$rows[8]->requireString('name');
// Throws: The key "response.rows.8.name" of account API must be of type string, int
given.
```

`requireReaders('rows')` would reject this outer collection because its keys are not a consecutive list. No reader reindexes an arbitrary collection.

Last updated on 26/10/2026