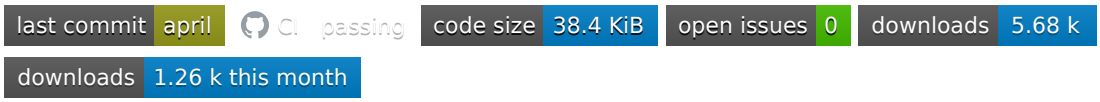
Container Project

Flexible Data Containers for PHP

Anonymous · 24/08/2026 · #php

Flexible Data Containers for PHP



A collection of specialized PHP data containers that provide different levels of structure and validation, from simple bags to schema-validated stores.

Features

- □ Purpose-built containers for different needs.
- □ JSON Schema validation support.
- □ Simple to complex data structures.
- □ Sequential data handling.
- □ Type-safe operations.
- □ Common interface across containers.
- □ Minimal dependencies.
- □ Comprehensive test coverage.

Why Derafu\Container?

Unlike generic data structures, Derafu\Container provides specialized containers each designed for specific use cases:

- **Bag**: Simple, flexible data storage without restrictions.
- **Vault**: Basic structured data with validation.
- **Store**: Advanced data validation using JSON Schema.
- **Journal**: Sequential data storage with LIFO access.

Installation

Install via Composer:

```
composer require derafu/container
```

Basic Usage

Bag - Flexible Container

```
use Derafu\Container\Bag;

$bag = new Bag();
$bag->set('user.name', 'John');
$bag->set('user.email', 'john@example.com');

echo $bag->get('user.name'); // "John"
```

Vault - Structured Container

```
use Derafu\Container\Vault;

$vault = new Vault([], [
    'user' => [
        'type' => 'array',
        'required' => true,
        'schema' => [
            'name' => ['type' => 'string'],
            'age' => ['type' => 'integer'],
        ],
    ],
]);

$vault->set('user', [
    'name' => 'John',
    'age' => 30,
]);
```

Store - JSON Schema Container

```
use Derafu\Container\Store;

$schema = [
    'type' => 'object',
    'properties' => [
        'user' => [
            'type' => 'object',
            'required' => ['name', 'email'],
            'properties' => [
                'name' => ['type' => 'string'],
                'email' => ['type' => 'string', 'format' => 'email'],
```

```
            ],
        ],
    ],
];

$store = new Store([], $schema);
$store->set('user', [
    'name' => 'John',
    'email' => 'john@example.com',
]);
```

Journal - Sequential Container

```
use Derafu\Container\Journal;

$journal = new Journal();
$journal->add('First Entry');
$journal->add('Second Entry');

print_r($journal->reverse()); // Shows entries newest first.
```

Container Comparison

Feature	Bag	Vault	Store	Journal
Schema	No	Simple	JSON	No
Validation	No	Basic	Full	No
Nested Data	Yes	Yes	Yes	No
Sequential	No	No	No	Yes
Dot Notation	Yes	Yes	Yes	No

Common Interface

All containers implement `ContainerInterface`:

```
interface ContainerInterface extends ArrayAccess
{
    public function set(string $key, mixed $value): static;
    public function get(string $key, mixed $default = null): mixed;
    public function has(string $key): bool;
    public function clear(?string $key = null): void;
}
```

Advanced Usage

Custom Validation in Vault

```
use Derafu\Container\Vault;

$vault = new Vault([], [
    'age' => [
        'type' => 'integer',
        'validator' => fn ($value) => $value >= 18
    ]
]);
```

Complex JSON Schema in Store

```
use Derafu\Container\Store;

$store = new Store([], [
    'type' => 'object',
    'properties' => [
        'users' => [
            'type' => 'array',
            'items' => [
                'type' => 'object',
                'required' => ['id', 'name'],
                'properties' => [
                    'id' => ['type' => 'integer'],
                    'name' => ['type' => 'string'],
                    'email' => ['type' => 'string', 'format' => 'email'],
                ],
            ],
        ],
    ],
]);
```

Journal with Filtering

```
use Derafu\Container\Journal;

$journal = new Journal();
$journal->add(['level' => 'info', 'message' => 'System started']);
$journal->add(['level' => 'error', 'message' => 'Connection failed']);

$errors = array_filter($journal->reverse(), fn($entry) =>
    $entry['level'] === 'error'
);
```

Last updated on 24/08/2026