

Basic Usage

Create an in-memory repository and request repositories through RepositoryManager

Anonymous · 20/10/2026

Basic Usage

Create an In-Memory Repository

Pass the rows directly, keyed by their repository identifiers. String keys, numeric lists and sparse numeric keys are supported.

```
use Derafu\Repository\Entity;
use Derafu\Repository\Repository;

$rows = [
    'prod-001' => [
        'name' => 'Laptop XPS',
        'category' => 'computers',
        'price' => 1299.99,
    ],
    'prod-002' => [
        'name' => 'Keyboard',
        'category' => 'accessories',
        'price' => 59.90,
    ],
];

$repository = new Repository($rows, Entity::class, idAttribute: 'id');
$product = $repository->find('prod-001');
$name = $product?->getAttribute('name'); // 'Laptop XPS'.
$all = $repository->findAll();
$computers = $repository->findBy(['category' => 'computers']);
$laptop = $repository->findOneBy(['name' => 'Laptop XPS']);
$total = $repository->count(); // 2.
```

`idAttribute` fills a missing field from the row key. An existing field value is preserved. `find()` uses the row key, even if that field contains a different value.

The direct constructor accepts an array, supported array-access objects or a PHP file path returning rows. Each row must be an array. When no entity class is supplied, the direct `Repository` constructor returns `stdClass` objects.

Use the Manager

The manager accepts a data provider and creates repositories when requested. A plain identifier such as `products` uses the manager's default entity class, `Derafu\Repository\Entity`.

```
use Derafu\Repository\Service\DataProvider;
use Derafu\Repository\Service\RepositoryManager;

$provider = new DataProvider(
    sources: ['products' => '/path/to/products.json'],
);
$manager = new RepositoryManager($provider);

$available = $manager->getAvailableRepositories(); // ['products'].
$product = $manager->getRepository('products');
$product = $product->find('prod-001');
```

The file contains the rows themselves, without a surrounding `products` property. Its extension selects the reader. See [data sources](#) for supported formats.

Listing available repositories does not read their data or construct them. A successful `getRepository()` call is cached by identifier: subsequent calls return the same repository instance.

Query Contract and Container Operations

`RepositoryInterface` exposes entity queries. It does not require container operations, database persistence or a particular constructor.

The concrete `Repository` class also provides container operations such as `all()`, `get()`, `set()`, `collection()` and array access through its container base class. These operate on stored row data. For example, `all()` returns raw rows whereas `findAll()` returns entity objects.

Depend explicitly on the concrete class or container capability when you need those operations. A repository obtained from a custom factory is only required to implement the query contract.

Last updated on 20/10/2026