Basic Usage

Basic Usage

Anonymous · 09/09/2026

Basic Usage

This guide covers the fundamentals of working with Derafu Selector for navigating and manipulating data structures.

Reading Data

The most common operation with Selector is to read values from complex data structures. The `get()` method provides a clean, safe way to extract values without worry about undefined indices or null values.

Basic Dot Notation

```
use Derafu\Selector\Selector;

$data = [
    'user' => [
        'name' => 'John Doe',
        'email' => 'john@example.com',
        'profile' => [
            'avatar' => 'john.jpg',
        ],
    ],
];

// Simple property access.
$name = Selector::get($data, 'user.name');  // "John Doe"

// Nested properties.
$avatar = Selector::get($data, 'user.profile.avatar');  // "john.jpg"

// Non-existent path returns null.
$age = Selector::get($data, 'user.age');  // null
```

Default Values

You can provide a default value that will be returned if the selector path doesn't exist:

```
// With default value
$address = Selector::get($data, 'user.address', 'No address provided');  // "No
address provided"
```

Array Access

Access array elements by index:

```
$data = [
    'items' => ['apple', 'banana', 'cherry'],
];

$firstItem = Selector::get($data, 'items[0]');  // "apple"
$lastItem = Selector::get($data, 'items[2]');   // "cherry"
```

Combining Dot Notation and Array Access

```
$data = [
    'categories' => [
        'fruits' => ['apple', 'banana', 'cherry'],
        'vegetables' => ['carrot', 'broccoli', 'spinach'],
    ],
];

$firstFruit = Selector::get($data, 'categories.fruits[0]');  // "apple"
$secondVegetable = Selector::get($data, 'categories.vegetables[1]');  // "broccoli"
```

Writing Data

Writing data is just as easy as reading it. The `set()` method handles creating intermediate structures as needed.

Setting Simple Values

```
$data = ['user' => ['name' => 'John']];

// Update an existing value.
Selector::set($data, 'user.name', 'Jane');
// $data is now ['user' => ['name' => 'Jane']]

// Create a new property.
Selector::set($data, 'user.email', 'jane@example.com');
// $data is now ['user' => ['name' => 'Jane', 'email' => 'jane@example.com']]
```

Creating Nested Structures

Selector will automatically create intermediate arrays:

```
$data = [];

// Create a deep structure.
Selector::set($data, 'settings.theme.colors.primary', '#3366FF');
// $data is now ['settings' => ['theme' => ['colors' => ['primary' => '#3366FF']]]]
```

Working with Arrays

```
$data = ['items' => ['apple', 'banana']];

// Add or update an array element.
Selector::set($data, 'items[2]', 'cherry');
// $data is now ['items' => ['apple', 'banana', 'cherry']]

// Update a specific element.
Selector::set($data, 'items[0]', 'red apple');
// $data is now ['items' => ['red apple', 'banana', 'cherry']]
```

Merging Arrays

When setting an array value to an existing array path, the arrays are merged:

```
$data = [
    'settings' => [
        'display' => ['theme' => 'light'],
    ],
];

Selector::set($data, 'settings.display', ['fontSize' => 'large']);
// $data is now ['settings' => ['display' => ['theme' => 'light', 'fontSize' =>
'large']]]
```

Checking if Paths Exist

The `has()` method checks if a value exists at the specified path:

```
$data = [
    'user' => [
        'name' => 'John',
        'email' => null,
    ],
```

```
];

Selector::has($data, 'user.name');      // true
Selector::has($data, 'user.email');     // false (because the value is null)
Selector::has($data, 'user.address');   // false (path doesn't exist)
```

Clearing Values

Remove values from the data structure with the `clear()` method:

```
$data = [
    'user' => [
        'name' => 'John',
        'email' => 'john@example.com',
        'settings' => ['theme' => 'dark'],
    ],
];

// Remove a simple property.
Selector::clear($data, 'user.email');
// $data is now ['user' => ['name' => 'John', 'settings' => ['theme' => 'dark']]]

// Remove a nested property.
Selector::clear($data, 'user.settings.theme');
// $data is now ['user' => ['name' => 'John', 'settings' => []]]

// Parent arrays are cleaned up if they become empty.
Selector::clear($data, 'user.name');
// $data is now ['user' => ['settings' => []]]
```

Working with Different Data Types

Selector handles various data types properly:

```
$data = [
    'values' => [
        'string' => 'text',
        'number' => 42,
        'boolean' => true,
        'null_value' => null,
        'array' => [1, 2, 3],
        'object' => ['key' => 'value'],
    ],
];

Selector::get($data, 'values.string');     // "text"
```

```
Selector::get($data, 'values.number');     // 42
Selector::get($data, 'values.boolean');    // true
Selector::get($data, 'values.null_value'); // null
Selector::get($data, 'values.array');      // [1, 2, 3]
Selector::get($data, 'values.object');     // ['key' => 'value']
```

Error Handling

Selector is designed to be forgiving and avoid common PHP errors:

```
// No "undefined index" errors.
Selector::get($data, 'non.existent.path');  // returns null

// No "trying to access array offset on null" errors.
$data = ['user' => null];
Selector::get($data, 'user.name');  // returns null

// However, if you have syntax errors in your selector, you'll get a
SelectorException.
try {
    Selector::get($data, 'user..name');  // Invalid selector with double dot
} catch (\Derafu\Selector\Exception\SelectorException $e) {
    echo $e->getMessage();  // "Failed to read using selector 'user..name'..."
}
```

For more advanced usage, check out the other guides in the documentation.

Last updated on 09/09/2026