Working with Arrays and Filters

Working with Arrays and Filters

Anonymous · 09/09/2026

Working with Arrays and Filters

This guide focuses on techniques for working with arrays and using filters in Derafu Selector.

Basic Array Access

Arrays can be accessed using both dot notation and square bracket syntax:

```
use Derafu\Selector\Selector;

$data = [
    'items' => ['apple', 'banana', 'cherry'],
    'counts' => [5, 10, 15],
];

// Access entire array.
$allItems = Selector::get($data, 'items');
// ['apple', 'banana', 'cherry']

// First element.
$firstItem = Selector::get($data, 'items[0]');
// 'apple'

// Last element.
$lastItem = Selector::get($data, 'items[2]');
// 'cherry'

// Non-existent index.
$nonExistent = Selector::get($data, 'items[5]');
// null
```

Numeric Indexing

```
$data = [
    'matrix' => [
        [1, 2, 3],
        [4, 5, 6],
```

```
        [7, 8, 9],
    ],
];

// Accessing multi-dimensional arrays.
$center = Selector::get($data, 'matrix[1][1]');
// 5

// Alternative dot notation.
$center = Selector::get($data, 'matrix.1.1');
// 5

// Mixed notation.
$firstRow = Selector::get($data, 'matrix[0]');
// [1, 2, 3]
$firstRowThirdElement = Selector::get($data, 'matrix[0][2]');
// 3
```

Filtering Arrays by Condition

The most powerful feature for array handling is filtering by key/value conditions:

```
$data = [
    'users' => [
        ['id' => 1, 'name' => 'John', 'role' => 'admin'],
        ['id' => 2, 'name' => 'Jane', 'role' => 'user'],
        ['id' => 3, 'name' => 'Bob', 'role' => 'user'],
    ]
];

// Find user by ID.
$userName = Selector::get($data, 'users[id=2:name]');
// 'Jane'

// Find user by role.
$adminName = Selector::get($data, 'users[role=admin:name]');
// 'John'

// Non-matching filter.
$nonExistent = Selector::get($data, 'users[id=99:name]');
// null
```

Syntax for Dependent Selectors

The syntax for array filtering with dependent selectors is:

```
arrayKey[requiredKey=requiredValue:dependentKey]
```

Where:

- `arrayKey` is the key containing the array to search in.
- `requiredKey` is the key to match in each array element.
- `requiredValue` is the value that `requiredKey` should equal.
- `dependentKey` is the key to extract from the matching element.

Complex Filtering

You can combine filters with further navigation:

```
$data = [
    'orders' => [
        [
            'id' => 1001,
            'customer' => 'John',
            'items' => [
                ['product' => 'Laptop', 'price' => 999],
                ['product' => 'Mouse', 'price' => 25],
            ],
        ],
        [
            'id' => 1002,
            'customer' => 'Jane',
            'items' => [
                ['product' => 'Phone', 'price' => 699],
                ['product' => 'Headphones', 'price' => 149],
            ],
        ],
    ],
];

// Get the first item from order 1002.
$product = Selector::get($data, 'orders[id=1002:items][0].product');
// 'Phone'

// Get the price of the second item from order 1001.
$price = Selector::get($data, 'orders[id=1001:items][1].price');
// 25

// Find an item by product name within an order.
$data = [
    'orders' => [
        [
            'id' => 1001,
            'items' => [
                ['product' => 'Laptop', 'price' => 999],
```

```
                ['product' => 'Mouse', 'price' => 25],
            ],
        ],
    ],
];
$mousePrice = Selector::get($data, 'orders[id=1001:items][product=Mouse:price]');
// 25
```

Nested Arrays

For deeply nested structures:

```
$data = [
    'departments' => [
        [
            'name' => 'Engineering',
            'teams' => [
                [
                    'name' => 'Frontend',
                    'members' => [
                        ['id' => 101, 'name' => 'Alice'],
                        ['id' => 102, 'name' => 'Bob'],
                    ]
                ],
                [
                    'name' => 'Backend',
                    'members' => [
                        ['id' => 201, 'name' => 'Charlie'],
                        ['id' => 202, 'name' => 'Diana'],
                    ],
                ],
            ],
        ],
    ],
];

// Find Charlie's ID through the department and team.
$charlieId = Selector::get($data,
    'departments[name=Engineering:teams][name=Backend:members][name=Charlie:id]'
); // 201
```

Array Operations

Writing to Arrays

```
$data = [
```

```
    'users' => [
        ['id' => 1, 'name' => 'John'],
        ['id' => 2, 'name' => 'Jane'],
    ],
];

// Update a value.
Selector::set($data, 'users[id=1:name]', 'Johnny');
// $data['users'][0]['name'] is now 'Johnny'

// Add a new property.
Selector::set($data, 'users[id=2:email]', 'jane@example.com');
// $data['users'][1]['email'] is now 'jane@example.com'

// Add a new element.
Selector::set($data, 'users[3]', ['id' => 3, 'name' => 'Bob']);
// $data['users'][3] is now ['id' => 3, 'name' => 'Bob']

// If no matching element exists, a new one is created.
Selector::set($data, 'users[id=4:name]', 'Alice');
// Adds a new element to $data['users'] with id=4 and name='Alice'
```

Clearing Array Elements

```
$data = [
    'users' => [
        ['id' => 1, 'name' => 'John', 'email' => 'john@example.com'],
        ['id' => 2, 'name' => 'Jane', 'email' => 'jane@example.com'],
    ],
];

// Remove a property.
Selector::clear($data, 'users[id=1:email]');
// John's email is now removed.

// Remove an entire array element.
Selector::clear($data, 'users[id=2]');
// Jane's record is now completely removed.

// Clean up empty array elements.
$data = [
    'teams' => [
        ['id' => 1, 'members' => ['Alice', 'Bob']],
        ['id' => 2, 'members' => ['Charlie']],
    ],
];

// Remove the last member from team 2.
Selector::clear($data, 'teams[id=2:members][0]');
// Now teams[id=2:members] is an empty array.
```

```
// Derafu Selector automatically cleans up empty arrays:
// $data['teams'][1]['members'] is now an empty array.
// If we clear team 1's members too, it would clean up further.
Selector::clear($data, 'teams[id=1:members]');
// Now both teams have empty members arrays.
```

Working with Mixed Array Types

Selector handles both associative and indexed arrays seamlessly:

```
$data = [
    'products' => [
        'electronics' => [
            ['id' => 'e1', 'name' => 'Laptop', 'price' => 999],
            ['id' => 'e2', 'name' => 'Phone', 'price' => 699],
        ],
        'books' => [
            ['id' => 'b1', 'name' => 'Novel', 'price' => 15],
            ['id' => 'b2', 'name' => 'Textbook', 'price' => 50],
        ],
    ],
];

// Access by category and index.
$firstElectronic = Selector::get($data, 'products.electronics[0].name');
// 'Laptop'

// Access by category and ID.
$textbookPrice = Selector::get($data, 'products.books[id=b2:price]');
// 50

// Numeric indices still work with filtering.
$secondBookName = Selector::get($data, 'products.books[1].name');
// 'Textbook'
```

Handling Empty or Non-existent Arrays

Selector provides safe handling for empty or non-existent arrays:

```
$data = [
    'categories' => [
        'active' => [
            ['id' => 1, 'name' => 'Electronics'],
            ['id' => 2, 'name' => 'Books'],
        ],
        'inactive' => [],
    ],
];
```

```
// Access on empty array.
$inactiveCategory = Selector::get($data, 'categories.inactive[0]');
// null

// Filter on empty array.
$result = Selector::get($data, 'categories.inactive[id=1:name]');
// null

// Non-existent array path.
$result = Selector::get($data, 'categories.archived[0].name');
// null

// Providing defaults.
$result = Selector::get($data, 'categories.archived[0].name', 'Not found');
// 'Not found'
```

Modifying Array Elements in Place

You can modify array elements directly:

```
$data = [
    'cart' => [
        ['id' => 'p1', 'name' => 'Laptop', 'quantity' => 1],
        ['id' => 'p2', 'name' => 'Mouse', 'quantity' => 2],
    ],
];

// Increase quantity.
$currentQuantity = Selector::get($data, 'cart[id=p1:quantity]');
Selector::set($data, 'cart[id=p1:quantity]', $currentQuantity + 1);
// Now p1's quantity is 2

// Add a new property to an array element.
Selector::set($data, 'cart[id=p2:price]', 25);
// Mouse now has a price property

// Modify a nested property.
$data = [
    'orders' => [
        [
            'id' => 1,
            'items' => [
                ['product' => 'Laptop', 'price' => 999],
            ],
        ],
    ],
];

// Apply discount.
```

```
Selector::set($data, 'orders[id=1:items][product=Laptop:price]', 899);
// Price is now 899
```

By leveraging these array handling capabilities, you can work with complex nested data structures in a clean, readable way without worrying about undefined indices or complex nesting logic.

Last updated on 09/09/2026