JSONPath Integration

JSONPath Integration

Anonymous · 09/09/2026

JSONPath Integration

This guide explains how to use JSONPath with Derafu Selector for powerful data structure querying.

Introduction to JSONPath

JSONPath is a query language for JSON, similar to XPath for XML. It provides a powerful way to extract data from complex JSON structures. Derafu Selector integrates JSONPath to give you even more flexibility in navigating your data.

All JSONPath expressions in Derafu Selector start with `$.` to distinguish them from regular selectors.

Basic JSONPath Queries

Here are some basic JSONPath examples:

```
use Derafu\Selector\Selector;

$data = [
    'store' => [
        'books' => [
            ['title' => 'Book 1', 'price' => 10],
            ['title' => 'Book 2', 'price' => 20],
            ['title' => 'Book 3', 'price' => 30],
        ],
        'bicycles' => [
            ['model' => 'Model A', 'price' => 100],
            ['model' => 'Model B', 'price' => 200],
        ],
    ],
    'user' => [
        'name' => 'John',
        'email' => 'john@example.com',
    ],
];

// Access a simple property.
```

```
$userName = Selector::get($data, '$.user.name');
// "John"

// Access an array element by index.
$firstBook = Selector::get($data, '$.store.books[0].title');
// "Book 1"

// Get all book titles.
$allTitles = Selector::get($data, '$.store.books[*].title');
// ["Book 1", "Book 2", "Book 3"]

// Get the entire books array.
$allBooks = Selector::get($data, '$.store.books');
// The complete books array
```

Advanced JSONPath Features

JSONPath offers powerful filtering capabilities:

```
// Filter books by price (more than 15).
$expensiveBooks = Selector::get($data, '$.store.books[?(@.price > 15)].title');
// ["Book 2", "Book 3"]

// Filter books by title (exact match).
$specificBook = Selector::get($data, '$.store.books[?(@.title == "Book 2")]');
// Returns the complete book object with title "Book 2"

// Multiple conditions with AND.
$filtered = Selector::get($data,
    '$.store.books[?(@.price > 15 && @.price < 25)].title'
);
// ["Book 2"]

// Regular expression matching (if supported by the JSONPath implementation).
$booksWithPattern = Selector::get($data, '$.store.books[?(@.title =~ /Book
[1-2]/)].title');
// ["Book 1", "Book 2"]

// Array slicing.
$firstTwoBooks = Selector::get($data, '$.store.books[0:2].title');
// ["Book 1", "Book 2"]
```

Common JSONPath Syntax Elements

Symbol	Description
`$`	The root object/element

Symbol	Description
`@`	The current object/element
`.`	Child operator
`..`	Recursive descent (find all matches at any level)
`*`	Wildcard (all objects/elements)
`[n]`	Array index (0-based)
`[n:m]`	Array slice from n to m
`[?()]`	Filter expression
`==` `!=`	Equality operators
`>` `<` `>=` `<=`	Comparison operators
`&&` `||`	Logical AND and OR

Combining JSONPath with Derafu Selectors

You can mix JSONPath with regular Derafu selectors for maximum flexibility:

```
// Use JSONPath to get a value and then concatenate with text.
$message = Selector::get($data, '"User: "($.user.name)');
// "User: John"

// Use JSONPath as part of an OR selector.
$contact = Selector::get($data, '$.user.phone||$.user.email');
// Falls back to "john@example.com" if phone doesn't exist

// Use JSONPath result in a conditional expression.
$userType = Selector::get($data,
    '(($.user.email) contains "example.com" ? ("Standard") : ("Premium"))'
);
// "Standard"

// Format array results.
$bookList = Selector::get($data,
    '"Available books: "($.store.books[*].title)'
);
// "Available books: [\"Book 1\", \"Book 2\", \"Book 3\"]"

// Mix JSONPath with regular selectors.
$data = [
    'config' => ['theme' => 'dark'],
    'preferences' => [
        'admin' => ['theme' => 'light'],
        'user' => ['theme' => 'system']
```

```
    ]
];

$theme = Selector::get($data,
    '(($.user.role) = "admin" ? (preferences.admin.theme) : (config.theme))'
);
// Falls back to "dark" if user.role doesn't exist
```

Performance Considerations

While JSONPath provides powerful querying capabilities, it may have different performance characteristics compared to native Derafu selectors:

1. **Initialization Cost**: JSONPath queries require initializing the JSONPath processor, which adds some overhead.

2. **Complex Queries**: For very complex queries with multiple conditions, JSONPath may be more efficient than chaining multiple native selectors.

3. **Large Data Sets**: For very large data structures, JSONPath's optimized filtering can be more efficient than iterating through the data manually.

4. **Writing Operations**: Note that JSONPath in Derafu Selector supports read operations only. For writing operations, you need to use native selectors.

Best Practices:

- Use native Derafu selectors for simple path access (`user.profile.name`).
- Use JSONPath for complex filtering and array operations.
- Consider caching results of expensive JSONPath queries if they're used repeatedly.
- Avoid using `..` (recursive descent) on very large data structures if performance is critical.

```
// Less efficient for simple access.
$name = Selector::get($data, '$.user.name');

// More efficient native syntax for simple access.
$name = Selector::get($data, 'user.name');

// JSONPath is more efficient for complex filtering.
$expensiveItems = Selector::get($data,
    '$.items[?(@.price > 100 && @.category == "electronics")].name'
);
```

By understanding when to use JSONPath versus native selectors, you can optimize your code for both readability and performance.

Last updated on 09/09/2026