JMESPath Integration

JMESPath Integration

Anonymous · 09/09/2026

JMESPath Integration

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

Introduction to JMESPath

JMESPath is a query language for JSON that allows you to extract and transform elements from complex data structures. It provides a more structured and feature-rich alternative to JSONPath with a clear specification.

All JMESPath expressions in Derafu Selector start with `jmespath:` to distinguish them from regular selectors.

Basic JMESPath Queries

Here are some basic JMESPath examples:

```
use Derafu\Selector\Selector;

$data = [
    'people' => [
        [
            'name' => 'John',
            'age' => 30,
            'phones' => ['home' => '555-1234', 'mobile' => '555-5678'],
        ],
        [
            'name' => 'Jane',
            'age' => 25,
            'phones' => ['home' => '555-4321', 'mobile' => '555-8765'],
        ],
    ],
    'locations' => [
        'Seattle' => ['state' => 'WA', 'population' => 724305],
        'New York' => ['state' => 'NY', 'population' => 8804190],
    ],
];
```

```
// Access a simple path.
$firstPerson = Selector::get($data, 'jmespath:people[0]');
// Complete first person object.

// Access a specific property.
$firstName = Selector::get($data, 'jmespath:people[0].name');
// "John"

// Access all names.
$allNames = Selector::get($data, 'jmespath:people[*].name');
// ["John", "Jane"]

// Access a nested property.
$firstMobile = Selector::get($data, 'jmespath:people[0].phones.mobile');
// "555-5678"

// Use bracket notation for keys with special characters.
$nyState = Selector::get($data, 'jmespath:locations."New York".state');
// "NY"
```

Advanced JMESPath Features

JMESPath offers powerful filtering, projection, and transformation capabilities:

Filtering

```
// Filter people over age 25.
$over25 = Selector::get($data, 'jmespath:people[?age > `25`]');
// Returns the complete person object for John.

// Filter with multiple conditions.
$filtered = Selector::get($data,
    'jmespath:people[?age > `25` && contains(name, `Jo`)]'
);
// Returns John's complete object.

// Get just the names of filtered results.
$names = Selector::get($data, 'jmespath:people[?age > `25`].name');
// ["John"]

// Filter on nested properties.
$withMobile = Selector::get($data,
    'jmespath:people[?phones.mobile != null].name'
);
// ["John", "Jane"]
```

Multi-select and Projections

```
// Select specific fields (projection).
$simplified = Selector::get($data, 'jmespath:people[*].{n: name, a: age}');
// [{"n":"John","a":30}, {"n":"Jane","a":25}]

// Multi-select on a single object.
$details = Selector::get($data, 'jmespath:people[0].{name: name, mobile:
phones.mobile}');
// {"name":"John","mobile":"555-5678"}

// Flatten nested structures.
$phones = Selector::get($data, 'jmespath:people[*].phones.*');
// ["555-1234", "555-5678", "555-4321", "555-8765"]
```

Functions

JMESPath supports a variety of built-in functions:

```
// String functions.
$upperNames = Selector::get($data, 'jmespath:people[*].name | [*].to_upper(@)');
// ["JOHN", "JANE"]

// Length/count.
$peopleCount = Selector::get($data, 'jmespath:length(people)');
// 2

// Min/max of values.
$data = ['values' => [5, 3, 8, 1, 7]];
$maxValue = Selector::get($data, 'jmespath:max(values)');
// 8

// Sort function.
$sortedNames = Selector::get($data, 'jmespath:sort(people[*].name)');
// ["Jane", "John"]

// Map function for transformations.
$ages = Selector::get($data, 'jmespath:people[*].age | map(&to_string(@), @)');
// ["30", "25"]
```

Slicing and Indexing

```
// Array slicing.
$firstPerson = Selector::get($data, 'jmespath:people[:1]');
// [Complete first person object]

// Negative indices.
$lastPerson = Selector::get($data, 'jmespath:people[-1]');
```

```
// Complete last person object (Jane).

// Step values.
$everyOtherPerson = Selector::get($data, 'jmespath:people[::2]');
// Every other person in the array.
```

Combining JMESPath with Derafu Selectors

You can mix JMESPath with regular Derafu selectors:

```
// Use JMESPath to get a value and then concatenate with text.
$greeting = Selector::get($data, '"Hello, "(jmespath:people[0].name)"!"');
// "Hello, John!"

// Use JMESPath as part of an OR selector.
$location = Selector::get($data,
    'jmespath:current_location||jmespath:locations."Seattle".state'
);
// Falls back to "WA" if current_location doesn't exist.

// Use JMESPath within a conditional.
$message = Selector::get($data,
    '((jmespath:people | length(@)) > "1" ? ("Multiple people") : ("One person"))'
);
// "Multiple people"

// Format array results.
$nameList = Selector::get($data,
    '"People: "(jmespath:people[*].name)'
);
// "People: [\"John\", \"Jane\"]"
```

Choosing Between JMESPath and JSONPath

Derafu Selector supports both JMESPath and JSONPath. Here are some considerations for choosing between them:

JMESPath Advantages

- **Formal Specification**: JMESPath has a formal, well-defined specification.
- **Functions**: Built-in functions for transformation and manipulation.
- **Multi-select Projection**: Create new structures with the data you extract.
- **Expressions**: More powerful expression language.
- **Pipe Operator**: Chain operations together.

JSONPath Advantages

- **Simplicity**: More straightforward syntax for basic queries.
- **Familiarity**: If you're already using JSONPath elsewhere.
- **Recursive Descent**: The `..` operator for finding elements at any level.

Examples of the Same Query in Both

```
$data = [
    'products' => [
        ['name' => 'Laptop', 'price' => 999, 'category' => 'electronics'],
        ['name' => 'Phone', 'price' => 699, 'category' => 'electronics'],
        ['name' => 'Book', 'price' => 15, 'category' => 'media'],
    ],
];

// Get electronics products with JSONPath.
$electronics = Selector::get($data,
    '$.products[?(@.category == "electronics")].name'
);
// ["Laptop", "Phone"]

// Same query with JMESPath.
$electronics = Selector::get($data,
    'jmespath:products[?category == `electronics`].name'
);
// ["Laptop", "Phone"]

// Filter by price and sort with JSONPath (sorting may not be available in all
implementations).
// May require multiple selector calls.

// With JMESPath, this is built-in.
$sortedProducts = Selector::get($data,
    'jmespath:products[?price > `500`].name | sort(@)'
);
// ["Laptop", "Phone"] (sorted)
```

By understanding the strengths of each query language, you can choose the right tool for your specific data navigation needs.

Last updated on 09/09/2026