Advanced Syntax

Advanced Selector Syntax

Anonymous · 09/09/2026

Advanced Selector Syntax

This guide covers the advanced selector syntax features that make Derafu Selector powerful for complex data manipulation.

String Literals and Concatenation

You can include literal strings in your selectors and concatenate them with values from your data:

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

// Simple string concatenation.
$greeting = Selector::get($data, '"Hello, "(user.name)"!"');
// "Hello, John!"

// Multiple concatenations.
$message = Selector::get($data, '"User "(user.name)" logged in at "(timestamp)');
// "User John logged in at 2023-04-15 10:30:45"
```

OR Operator for Fallbacks

The `||` operator provides fallback values when a selector path doesn't exist or returns null:

```
$data = [
    'user' => [
        'name' => 'John',
        'email' => 'john@example.com',
        // phone is not set
    ],
];

// Use email if phone doesn't exist.
$contact = Selector::get($data, 'user.phone||user.email');
// "john@example.com"

// Chains of fallbacks.
```

```
$identifier = Selector::get($data, 'user.id||user.username||user.email');
// "john@example.com"

// Fallback to a literal string.
$phone = Selector::get($data, 'user.phone||"Not provided"');
// "Not provided"

// Combine with string concatenation.
$phoneDisplay = Selector::get($data, '"Phone: "(user.phone||"N/A")');
// "Phone: N/A"
```

Conditional (Ternary) Selectors

Conditionals let you choose between different selector paths based on conditions:

```
$data = [
    'user' => [
        'type' => 'admin',
        'admin_role' => 'super_admin',
        'user_role' => 'regular',
    ],
];

// Simple condition using equality.
$role = Selector::get($data,
    '((user.type) = "admin" ? (user.admin_role) : (user.user_role))'
); // "super_admin"

// Using not equal.
$accessLevel = Selector::get($data,
    '((user.type) != "guest" ? ("authenticated") : ("anonymous"))'
); // "authenticated"

// Numeric comparisons.
$data = ['score' => 85];
$grade = Selector::get($data,
    '((score) >= "90" ? ("A") : ((score) >= "80" ? ("B") : ("C")))'
); // "B"
```

Dependent Selectors

Find and access specific elements in arrays based on key/value matching:

```
$data = [
    'users' => [
        ['id' => 101, 'name' => 'John', 'email' => 'john@example.com'],
```

```
        ['id' => 102, 'name' => 'Jane', 'email' => 'jane@example.com'],
        ['id' => 103, 'name' => 'Bob', 'email' => 'bob@example.com'],
    ],
];

// Get the name of user with id 102.
$name = Selector::get($data, 'users[id=102:name]');
// "Jane"

// Get the email of user with id 101.
$email = Selector::get($data, 'users[id=101:email]');
// "john@example.com"

// Nested properties.
$data = [
    'orders' => [
        [
            'id' => 1001,
            'customer' => ['id' => 101, 'name' => 'John'],
            'items' => [['product' => 'Laptop', 'price' => 999]],
        ],
        [
            'id' => 1002,
            'customer' => ['id' => 102, 'name' => 'Jane'],
            'items' => [['product' => 'Phone', 'price' => 699]],
        ],
    ],
];

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

// Access by string values with special characters.
$data = [
    'categories' => [
        ['id' => 'cat-1', 'name' => 'Electronics'],
        ['id' => 'cat-2', 'name' => 'Books']
    ]
];

$catName = Selector::get($data, 'categories[id=cat-1:name]');
// "Electronics"
```

Comparison Operators

Conditional selectors support several comparison operators:

```
$data = [
    'product' => [
        'price' => 50,
        'stock' => 10,
        'name' => 'Gadget',
        'tags' => ['electronics', 'new'],
    ],
];

// Equality.
$isGadget = Selector::get($data,
    '((product.name) = "Gadget" ? ("Yes") : ("No"))'
); // "Yes"

// Not equal.
$isExpensive = Selector::get($data,
    '((product.price) != "100" ? ("Affordable") : ("Expensive"))'
); // "Affordable"

// Greater than.
$priceTier = Selector::get($data,
    '((product.price) > "75" ? ("Premium") : ("Standard"))'
); // "Standard"

// Less than or equal.
$stockStatus = Selector::get($data,
    '((product.stock) <= "5" ? ("Low Stock") : ("In Stock"))'
); // "In Stock"

// Contains (for arrays and strings).
$isNew = Selector::get($data,
    '((product.tags) contains "new" ? ("New Arrival") : ("Regular Item"))'
); // "New Arrival"

// Length check.
$nameLength = Selector::get($data,
    '((product.name) length "6" ? ("Six Letters") : ("Other Length"))'
); // "Six Letters"

// Null check.
$data['product']['description'] = null;
$hasDescription = Selector::get($data,
    '((product.description) is "null" ? ("No Description") : ("Has Description"))'
); // "No Description"
```

Special Functions

The selector syntax includes several special functions:

```
$data = [
    'product' => [
        'name' => 'Gadget',
        'price' => 49.99,
        'tags' => ['electronics', 'gadget', 'new'],
    ],
];

// contains - check if an array or string contains a value.
$hasTag = Selector::get($data,
    '((product.tags) contains "gadget" ? ("Tagged as gadget") : ("Not tagged"))'
); // "Tagged as gadget"

// length - check the length of a string or array.
$tagCount = Selector::get($data,
    '((product.tags) length "3" ? ("Has 3 tags") : ("Has another tag count"))'
); // "Has 3 tags"

$nameLength = Selector::get($data,
    '((product.name) length "6" ? ("Name has 6 chars") : ("Name has different
length"))'
); // "Name has 6 chars"

// is - check the type of a value (currently supports null checks).
$hasDescription = Selector::get($data,
    '((product.description) is "null" ? ("No description") : ("Has description"))'
); // "No description"
```

Parentheses and Escaping

The selector syntax uses parentheses and quotes extensively:

```
// (selector) extracts a value from the data
// "string" is a literal string

$data = ['message' => 'Hello World'];

// Simple extraction.
$msg = Selector::get($data, '(message)'); // "Hello World"

// Literal string.
$literal = Selector::get($data, '"Static text"'); // "Static text"

// Concatenating extractions and literals.
$full = Selector::get($data, '(message)" - welcome!"'); // "Hello World - welcome!"

// Escaping quotes and parentheses.
$escaped = Selector::get($data, '"This is a \\"quoted\\" string"'); // 'This is a
```

```
"quoted" string'
$parentheses = Selector::get($data, '"Formula: (x + y)"'); // "Formula: (x + y)"

// Nested parentheses for complex expressions.
$nested = Selector::get($data, '((message) = "Hello World" ? ("Greeting") : ("Other
message"))');
// "Greeting"
```

For even more advanced capabilities, explore the integration with JSONPath and JMESPath in their respective guides.

Last updated on 09/09/2026