

Validation

How the rules that process the data of a form are derived from its schema, and what each keyword of the schema validates

Anonymous · 15/10/2026

Validation

`FormDataProcessor` processes the data that comes with a form: it casts, sanitizes, transforms and validates the value of each field, and gives back the processed data and the errors. The rules are the ones of [derafu/data-processor](#). Nobody writes them for the common cases: they are **derived from the schema of the form**, and the `rules` section of the definition adds the rest.

```
$result = $processor->process($form, $_POST);

if ($result->hasErrors()) {
    $errors = $result->getErrors();    // ['field' => ['message']]
    $form = $result->getForm();        // The form with its errors, to render it
    again.
} else {
    $data = $result->getProcessedData();
}
```

Where the rules come from

For each field the processor joins three sources, in this order:

1. **The property of the schema:** its type, format, pattern, limits, enum, etc. (see below).
2. **The control of the UI schema:** a control of type `file` or `image` validates a file or an image, and a control of type `editor` removes the tags of the HTML.
3. **The rules section of the definition:** explicit rules by field.

```
'rules' => [
    'name' => [
        'transform' => ['uppercase'],
        'validate' => ['min_length:3'],
    ],
],
```

The rules of a field are written in four categories, and are applied in this order: `cast`, `transform`, `sanitize`, `validate`. The explicit ones are joined to the derived ones like this:

- `cast`: the explicit one **replaces** the derived one. When the type of the cast changes, the derived sanitizers (for example `trim`, that is for strings) are dropped, and the explicit ones are kept.
- `sanitize`, `transform` and `validate`: the explicit ones are **added after** the derived ones.
- `required`: the fields that are required (see [Required fields](#)) have it as the first rule of `validate`, once.

A field whose [rule of the UI schema](#) is not active (it is hidden, for example) is not validated and is not in the processed data, even if it came with the data.

Objects

The properties of an object of the schema are fields of the form with the name of their path, `address.city`, and the data has them inside the object, as PHP reads a form (`address[city]` in the name of the control):

```
'schema' => [
    'type' => 'object',
    'required' => ['contact'],
    'properties' => [
        'contact' => [
            'type' => 'object',
            'required' => ['address', 'zone'],
            'properties' => [
                'address' => ['type' => 'string'],
                'zone' => ['type' => 'string'],
                'city' => ['type' => 'string'],
            ],
        ],
    ],
],
'uischema' => [
    'type' => 'VerticalLayout',
    'elements' => [
        ['type' => 'Control', 'scope' => '#/properties/contact/properties/address'],
        ['type' => 'Control', 'scope' => '#/properties/contact/properties/zone'],
        ['type' => 'Control', 'scope' => '#/properties/contact/properties/city'],
    ],
],
```

```
$result = $processor->process($form, ['contact' => ['address' => 'Street 1', 'zone' =>
'North', 'city' => '']]);
```

```
$result->getProcessedData(); // ['contact' => ['address' => 'Street 1', 'zone' =>
'North', 'city' => '']]
```

```
$result->getErrors(); // ['contact.zone' => [...]] when the zone is missing.
```

- The processed data and the data that comes back to the form have the same shape as the data that came.
- The errors have the name of the path of the field (`contact.zone`).
- A field is required by the `required` of **its own object**: `address` and `zone` are, `city` is not. The `required` of the root only asks for the key `contact`: it does not make any of its fields required.
- A key with the whole name (`'contact.address'`) is the value of that field too.
- The conditions of the [rules of the UI schema](#) can be on a field of an object (`#/properties/contact/properties/city`).

What the schema validates

In the schema	Rule
<code>type: string</code>	cast string; sanitize trim
<code>type: integer</code>	cast integer; validate int
<code>type: number</code>	cast float; validate numeric
<code>type: boolean</code>	cast boolean
<code>minLength, maxLength</code>	<code>min_length, max_length</code>
<code>minimum, maximum</code> (integer and number)	<code>gte, lte</code>
<code>minItems, maxItems, uniqueItems</code>	<code>min_items, max_items, unique</code>
<code>pattern</code>	<code>regex</code> (see below)
<code>enum</code> , or <code>oneOf</code> with <code>const</code>	<code>in</code> with the values
<code>contentType: application/json</code>	<code>json</code>
<code>format</code>	see the table below

A string with `format: email` is also transformed to lowercase.

Formats

format	Rule
<code>email</code>	<code>email</code>
<code>uri, url</code>	<code>url</code>
<code>date</code>	<code>date_format: Y-m-d</code>
<code>date-time</code>	<code>date_format: Y-m-d H:i:s</code>
<code>time</code>	<code>date_format: H:i:s</code>
<code>tel</code>	regex of telephone numbers
<code>hostname</code>	<code>hostname</code>

format	Rule
ipv4, ipv6	ip
uuid	uuid
base64	base64
json	json
regex	regex_pattern: the value must be a regular expression
another	string

Patterns

The `pattern` of JSON Schema is written without delimiters, like the browsers read the `pattern` attribute of HTML. PHP needs them, so the form gives the pattern the slash as its delimiter and escapes the slashes that it has, once: `^a/b$` and `^a\/b$` are the same pattern. The same conversion is used by the patterns of the types and by the conditions of the rules of the UI schema.

The regular expression is a PCRE one, the dialect of PHP. A pattern that can not be compiled is an error of the definition, not of the value: it is reported when the field is processed. A field with `format: regex` is the opposite: the **value** is the pattern, and it is invalid if it can not be compiled.

Booleans

For JSON Schema, `required` says that the key must be in the data, not that it has a given value. A boolean that is only `required` is valid when it is checked and when it is not. To ask for a value, fix it in the schema:

```
'terms' => [  
  'type' => 'boolean',  
  'title' => 'I accept the terms',  
  'const' => true,           // or 'enum' => [true]  
],
```

- `const: true` or `enum: [true]` validate with `accepted`: the value must be `true`.
- `const: false` or `enum: [false]` validate with `declined`: the value must be `false`.

A browser does not send a checkbox that is not checked, so a missing key is read as `false`, and the form that asks for `true` fails. The control has the native `required` too, see [Required fields](#).

Translated errors

The messages of the errors are translated when the processor has a translator, see [Translations](#).

Last updated on 15/10/2026