

Required Fields

How a field is required, how its label and its control say it, and how the browser asks for it

Anonymous · 15/10/2026

Required Fields

A field is **required** when its property is in the `required` list of the object that has it, as in JSON Schema:

```
'schema' => [  
  'type' => 'object',  
  'required' => ['name', 'email'],  
  'properties' => [  
    'name' => ['type' => 'string'],  
    'email' => ['type' => 'string', 'format' => 'email'],  
    'nick' => ['type' => 'string'],  
  ],  
],
```

The form marks each of its fields (`$field->isRequired()`), and three things use it:

- The **processor** validates the value with the rule `required` before any other (see [Validation](#)).
- The **label** shows the marker of a required field.
- The **control** says that it is required to the browser and to the assistive technologies.

The marker of the label

The label of a required field has the class `required` and an asterisk (`<span class="text-danger">*</span>`). It is shown unless the option `hide_required_asterisk` is `true`:

```
// In the options of the form: all of its fields.  
'options' => ['hide_required_asterisk' => true],  
  
// In a control: only that field.  
['type' => 'Control', 'scope' => '#/properties/name', 'options' =>  
  ['hide_required_asterisk' => true]],
```

It can also be given when rendering (`{{ form_row(field, {hide_required_asterisk: true}) }}`). The option of the control decides first, then the ones that are given when rendering and last the ones of the form (see [Rendering options](#)). A value that is not a boolean is an error.

What the browser asks for

By default the control of a required field has the **native required**, so the browser does not send the form until it has a value:

Control	Attribute
Input (except hidden), textarea, select	<code>required</code>
Radio buttons	<code>required</code> in the first one (it is enough for the group), and <code>aria-required="true"</code> in the group (<code>role="radiogroup"</code>)
Group of checkboxes	<code>aria-required="true"</code> in the group (<code>role="group"</code>): a browser can not ask for one of several
Slider	none: it always has a value

The form can say that the browser does not ask for it, with the option `native_required`:

```
'options' => ['native_required' => false],
```

Then the controls only have `aria-required="true"` (the assistive technologies say that the field is required, and the browser lets the form go). `native_required` is read like `hide_required_asterisk`: from the control, from what is given when rendering, from the form.

When the form does not say it, `native_required` is the opposite of `novalidate`: a form with the attribute `novalidate` (`'options' => ['attributes' => ['novalidate' => true]]`) is not validated by the browser, so it does not ask for the required fields either. An explicit `native_required: true` wins over `novalidate`.

The widgets and the labels that are rendered alone (`form_widget(field)`, `form_label(field)`) have the same behavior as the ones inside a form: it is the field that carries what its form decided.

```
$field->isRequired();           // It is in the `required` list.
$field->isNativeRequired();     // Its control has the native `required`.
$field->isRequiredAsteriskHidden(); // Its label does not show the asterisk.
```

Booleans

`required` says that the key must be in the data, not that it has a value, so a boolean that is only required has no marker and no `required` in its control. To ask for `true` (to accept the terms of a service) the schema says it:

```
'terms' => [
  'type' => 'boolean',
  'title' => 'I accept the terms',
```

```
'const' => true,          // or 'enum' => [true]
],
```

Its checkbox always has the native `required` and its label the asterisk, and the processor validates it with `accepted` (see [Validation](#)).

Fields that are not always there

A field with a [rule of the UI schema](#) that is not active when the form is rendered (it is hidden, or it is not enabled) has its controls disabled: the browser does not validate them nor sends them. When the data of the form changes, `ui-schema-rules.js` disables and enables them again as the rule says.

The processor does not validate the field that is not active either, even if its value came.

Tabs and sections that are closed

A browser validates the controls that are in a tab that is not active or in a section that is collapsed, but it can not focus them: it blocks the form and shows nothing. `invalid-reveal.js` solves it. When the form is invalid, it opens the tabs and the sections that have the first invalid control, focuses it and shows the message of the browser. It uses Bootstrap 5, the same that the templates of the categorizations and of the groups use.

```
import { InvalidReveal } from 'derafu-form/invalid-reveal.js';

document.addEventListener('DOMContentLoaded', () => InvalidReveal.init());
```

(With the Symfony bundle the module is in the import map under `derafu-form/`; without it, import it from `vendor/derafu/form/resources/js/invalid-reveal.js`.) If the control is already visible, the script does nothing.

Editors

The JSON editor (`json-editor.js`) and the HTML editor (`html-editor.js`) hide the control that they replace. A hidden control can not be focused, so both remove its `required`; the processor still asks for the value.

Last updated on 15/10/2026