SFW Learn Symfony 5 today (https://symfony.com/book)

Connect (/connect/login?target=https://symfony.com/doc/current/form/form_customization.html#reference-form-twig-functions) (/)

Symfony 5: The Fast Track (/book) is the best book to learn modern Symfony development, from zero to production. +300 pages showcasing Symfony with Docker, APIs, queues & async tasks, Webpack, SPAs, etc.

Buy printed/PDF version Read it for free

How to Customize Form Rendering

Symfony gives you several ways to customize how a form is rendered. In this article you'll learn how to make single customizations to one or more fields of your forms. If you need to customize all your forms in the same way, create instead a form theme (form_themes.html) or use any of the built-in themes, such as the Bootstrap theme for Symfony forms (bootstrap4.html).

Form Rendering Functions ¶

A single call to the form() Twig function is enough to render an entire form, including all its fields and error messages:

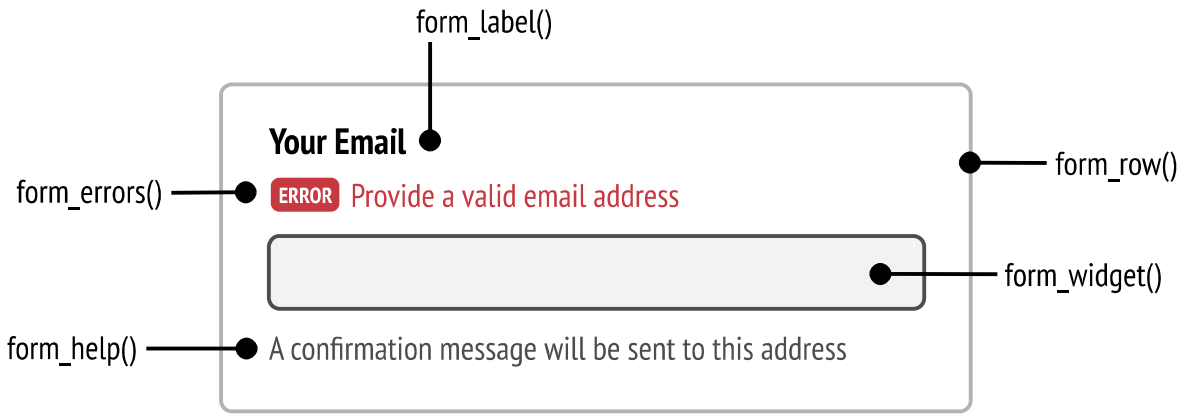
```
{# form is a variable passed from the controller and created
  by calling to the $form->createView() method #}
{{ form(form) }}
```

The next step is to use the form_start(), form_end(), form_errors() and form_row() Twig functions to render the different form parts so you can customize them adding HTML elements and attributes:

```
{{ form_start(form) }}
    <div class="my-custom-class-for-errors">
        {{ form_errors(form) }}
    </div>

    <div class="row">
        <div class="col">
            {{ form_row(form.task) }}
        </div>
        <div class="col" id="some-custom-id">
            {{ form_row(form.dueDate) }}
        </div>
    </div>
{{ form_end(form) }}
```

The `form_row()` function outputs the entire field contents, including the label, help message, HTML elements and error messages. All this can be further customized using other Twig functions, as illustrated in the following diagram:



The form_label(), form_widget(), form_help() and form_errors() Twig functions give you total control over how each form field is rendered, so you can fully customize them:

```
<div class="form-control">
    <i class="fa fa-calendar"></i> {{ form_label(form.dueDate) }}
    {{ form_widget(form.dueDate) }}

    <small>{{ form_help(form.dueDate) }}</small>

    <div class="form-error">
        {{ form_errors(form.dueDate) }}
    </div>
</div>
```

> Note
>
> Later in this article you can find the full reference of these Twig functions with more usage examples.

Form Rendering Variables ¶

Some of the Twig functions mentioned in the previous section allow to pass variables to configure their behavior. For example, the `form_label()` function lets you define a custom label to override the one defined in the form:

```
{{ form_label(form.task, 'My Custom Task Label') }}
```

Some form field types (../reference/forms/types.html) have additional rendering options that can be passed to the widget. These options are documented with each type, but one common option is `attr`, which allows you to modify HTML attributes on the form element. The following would add the `task_field` CSS class to the rendered input text field:

```
{{ form_widget(form.task, {'attr': {'class': 'task_field'}}) }}
```

> Note
>
> If you're rendering an entire form at once (or an entire embedded form), the `variables` argument will only be applied to the form itself and not its children. In other words, the following will not pass a "foo" class attribute to all of the child fields in the form:
>
> ```
> {# does **not** work - the variables are not recursive #}
> {{ form_widget(form, { 'attr': {'class': 'foo'} }) }}
> ```

If you need to render form fields "by hand" then you can access individual values for fields (such as the `id`, `name` and `label`) using its `vars` property. For example to get the `id`:

```
{{ form.task.vars.id }}
```

> Note
>
> Later in this article you can find the full reference of these Twig variables and their description.

Form Themes ¶

The Twig functions and variables shown in the previous sections can help you customize one or more fields of your forms. However, this customization can't be applied to the rest of the forms of your app.

If you want to customize all forms in the same way (for example to adapt the generated HTML code to the CSS framework used in your app) you must create a form theme (form_themes.html).

Form Functions and Variables Reference ¶

Functions ¶

form(form_view, variables) ¶

Renders the HTML of a complete form.

```
{# render the form and change the submission method #}
{{ form(form, {'method': 'GET'}) }}
```

You will mostly use this helper for prototyping or if you use custom form themes. If you need more flexibility in rendering the form, you should use the other helpers to render individual parts of the form instead:

```
{{ form_start(form) }}
    {{ form_errors(form) }}

    {{ form_row(form.name) }}
    {{ form_row(form.dueDate) }}

    {{ form_row(form.submit, { 'label': 'Submit me' }) }}
{{ form_end(form) }}
```

form_start(form_view, variables) ¶

Renders the start tag of a form. This helper takes care of printing the configured method and target action of the form. It will also include the correct `enctype` property if the form contains upload fields.

```
{# render the start tag and change the submission method #}
{{ form_start(form, {'method': 'GET'}) }}
```

form_end(form_view, variables) ¶

Renders the end tag of a form.

```
{{ form_end(form) }}
```

This helper also outputs `form_rest()` (which is explained later in this article) unless you set `render_rest` to false:

```
{# don't render unrendered fields #}
{{ form_end(form, {'render_rest': false}) }}
```

form_label(form_view, label, variables) ¶

Renders the label for the given field. You can optionally pass the specific label you want to display as the second argument.

```
{{ form_label(form.name) }}

{# The two following syntaxes are equivalent #}
{{ form_label(form.name, 'Your Name', {'label_attr': {'class': 'foo'}}) }}

{{ form_label(form.name, null, {
    'label': 'Your name',
    'label_attr': {'class': 'foo'}
}) }}
```

See “Form Variables Reference” to learn about the `variables` argument.

form_help(form_view) ¶

Renders the help text for the given field.

```
{{ form_help(form.name) }}
```

form_errors(form_view) ¶

Renders any errors for the given field.

```
{# render only the error messages related to this field #}
{{ form_errors(form.name) }}

{# render any "global" errors not associated to any form field #}
{{ form_errors(form) }}
```

> Caution
>
> In the Bootstrap 4 form theme, `form_errors()` is already included in `form_label()`, see "Error Messages (bootstrap4.html#reference-forms-bootstrap-error-messages)"

form_widget(form_view, variables) ¶

Renders the HTML widget of a given field. If you apply this to an entire form or collection of fields, each underlying form row will be rendered.

```
{# render a widget, but add a "foo" class to it #}
{{ form_widget(form.name, {'attr': {'class': 'foo'}}) }}
```

The second argument to `form_widget()` is an array of variables. The most common variable is `attr`, which is an array of HTML attributes to apply to the HTML widget. In some cases, certain types also have other template-related options that can be passed. These are discussed on a type-by-type basis. The `attributes` are not applied recursively to child fields if you're rendering many fields at once (e.g. `form_widget(form)`).

See "Form Variables Reference" to learn more about the `variables` argument.

form_row(form_view, variables) ¶

Renders the "row" of a given field, which is the combination of the field's label, errors, help and widget.

```
{# render a field row, but display a label with text "foo" #}
{{ form_row(form.name, {'label': 'foo'}) }}
```

The second argument to `form_row()` is an array of variables. The templates provided in Symfony only allow to override the label as shown in the example above.

See "Form Variables Reference" to learn about the `variables` argument.

form_rest(form_view, variables) ¶

This renders all fields that have not yet been rendered for the given form. It's a good idea to always have this somewhere inside your form as it'll render hidden fields for you and make any fields you forgot to render easier to spot (since it'll render the field for you).

```
{{ form_rest(form) }}
```

form_parent(form_view) ¶

Returns the parent form view or `null` if the form view already is the root form. Using this function should be preferred over accessing the parent form using `form.parent`. The latter way will produce different results when a child form is named `parent`.

Tests ¶

Tests can be executed by using the `is` operator in Twig to create a condition. Read the Twig documentation (https://twig.symfony.com/doc/2.x/templates.html#test-operator) for more information.

selectedchoice(selected_value) ¶

This test will check if the current choice is equal to the `selected_value` or if the current choice is in the array (when `selected_value` is an array).

```
<option {% if choice is selectedchoice(value) %}selected="selected"{% endif %}>
```

rootform ¶

This test will check if the current `form` does not have a parent form view.

```
{# DON'T DO THIS: this simple check can't differentiate between a form having
   a parent form view and a form defining a nested form field called 'parent' #}

{% if form.parent is null %}
    {{ form_errors(form) }}
{% endif %}

{# DO THIS: this check is always reliable, even if the form defines a field called 'parent' #}

{% if form is rootform %}
    {{ form_errors(form) }}
{% endif %}
```

Form Variables Reference ¶

The following variables are common to every field type. Certain field types may define even more variables and some variables here only really apply to certain types. To know the exact variables available for each type, check out the code of the templates used by your form theme (form_themes.html).

Assuming you have a `form` variable in your template and you want to reference the variables on the `name` field, accessing the variables is done by using a public `vars` property on the `Symfony\Component\Form\FormView` object:

```
<label for="{{ form.name.vars.id }}"
    class="{{ form.name.vars.required ? 'required' }}">
    {{ form.name.vars.label }}
</label>
```

Variable	Usage
`action`	The action of the current form.
`attr`	A key-value array that will be rendered as HTML attributes on the field.

Variable	Usage
`block_prefixes`	An array of all the names of the parent types.
`cache_key`	A unique key which is used for caching.
`compound`	Whether or not a field is actually a holder for a group of children fields (for example, a `choice` field, which is actually a group of checkboxes).
`data`	The normalized data of the type.
`disabled`	If `true`, `disabled="disabled"` is added to the field.
`errors`	An array of any errors attached to this specific field (e.g. `form.title.errors`). Note that you can't use `form.errors` to determine if a form is valid, since this only returns "global" errors: some individual fields may have errors. Instead, use the `valid` option.
`form`	The current `FormView` instance.
`full_name`	The `name` HTML attribute to be rendered.
`help`	The help message that will be rendered.
`id`	The `id` HTML attribute to be rendered.
`label`	The string label that will be rendered.
`label_attr`	A key-value array that will be rendered as HTML attributes on the label.
`method`	The method of the current form (POST, GET, etc.).
`multipart`	If `true`, `form_enctype` will render `enctype="multipart/form-data"`.
`name`	The name of the field (e.g. `title`) - but not the `name` HTML attribute, which is `full_name`.
`required`	If `true`, a `required` attribute is added to the field to activate HTML5 validation. Additionally, a `required` class is added to the label.
`submitted`	Returns `true` or `false` depending on whether the whole form is submitted
`translation_domain`	The domain of the translations for this form.
`valid`	Returns `true` or `false` depending on whether the whole form is valid.
`value`	The value that will be used when rendering (commonly the `value` HTML attribute). This only applies to the root form element.

Tip

Behind the scenes, these variables are made available to the `FormView` object of your form when the Form component calls `buildView()` and `finishView()` on each "node" of your form tree. To see what "view" variables a particular field has, find the source code for the form field (and its parent fields) and look at the above two functions.

This work, including the code samples, is licensed under a Creative Commons BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/) license.

Latest from the Symfony Blog

Adding 20 new checks in SymfonyInsight (/blog/adding-20-new-checks-in-symfonyinsight)
November 16, 2020

Symfony 3.4.47 released (/blog/symfony-3-4-47-released)
November 27, 2020

They Help Us Make Symfony

Thanks Ryan Linnit for being a Symfony contributor (/contributors).
1 commit · 1 lines

Get Involved in the Community

A passionate group of over 600,000 developers from more than 120 countries, all committed to helping PHP surpass the impossible.

Getting involved →

Symfony™ is a trademark of Symfony SAS. All rights reserved (/trademark).

Documentation (/blog/category/documentation)
Living on the edge (/blog/category/living-on-the-edge)
Releases (/blog/category/releases)
Security Advisories (/blog/category/security-advisories)
SymfonyInsight (/blog/category/symfony-insight)
Twig (/blog/category/twig)
SensioLabs (https://blog.sensiolabs.com/)

About (/about)
SensioLabs (https://sensiolabs.com/en/join_us/join_us.html)
Careers (https://sensiolabs.com/en/join_us/our_job_offers.html)
Support (/support)
Cloud TOS (/cloud/tos)

Deployed on

(/cloud/)

Follow Symfony

(https://github.com/symfony) (https://stackoverflow.com/questions/tagged/symfony) (/slack) (https://twitter.com/symfony) (https://www.facebook.com/SymfonyFramework) (https://www.youtube.com/user/SensioLabs) (https://symfonycasts.com/) (https://feeds.feedburner.com/symfony/blog)

Switch to dark theme