Form Bundle

Form Bundle

Anonymous · 24/08/2026 · #php, #symfony

Form Bundle

This bundle integrates the derafu/form declarative forms library with Symfony. Forms are defined as plain PHP files returning a JSON Schema + UI Schema structure rather than PHP builder classes, and the bundle wires the loader, renderer, and Twig extension into the Symfony container automatically.

Requirements

- PHP 8.5 or higher
- Symfony 8.0 or higher (framework-bundle + twig-bundle)
- Symfony AssetMapper (required for the JavaScript widgets)

Installation

Install the package via Composer:

```
composer require derafu/symfony-form-bundle
```

Because the bundle does not ship a Symfony Flex recipe, you must register it manually in `config/bundles.php`:

```
return [
    // ...
    Derafu\FormBundle\FormBundle::class => ['all' => true],
];
```

That is the only mandatory step. Everything else — asset mapper path registration, `importmap.php` entries, and form-path discovery across bundles — is handled automatically during container compilation.

What the bundle configures automatically

When the container is compiled (either on first boot or after `cache:clear`), the bundle performs three automatic setup tasks.

Asset mapper path. The extension registers `vendor/derafu/form/resources/js` under the `derafu-form` namespace in Symfony's asset mapper, so the JavaScript widgets become available as logical paths like `derafu-form/collection-widget.js`.

ImportMap entries. A compiler pass scans that same JS directory and adds any missing entries to `importmap.php`. The operation is idempotent: entries already present are never duplicated. If your project does not use AssetMapper and therefore has no `importmap.php`, this step is silently skipped.

Form-path discovery. Another compiler pass iterates over every registered bundle and checks whether it contains a `resources/forms/` directory. Any directory found is registered with the `PhpFormLoader` in bundle registration order. The application's own forms path (see the configuration section below) is added last, giving it the highest priority so the application can override any form defined by a bundle.

Optional configuration

The bundle works out of the box with its defaults. If you need to change either option, create `config/packages/derafu_form.yaml`:

```
derafu_form:
    twig_prefix: 'derafu_'
    forms_path: '%kernel.project_dir%/resources/forms'
```

`twig_prefix` controls the prefix of every Twig function registered by the bundle. The default `derafu_` avoids collisions with Symfony's own `symfony/form` functions (`form()`, `form_start()`, etc.). With the default prefix the functions are called `derafu_form()`, `derafu_form_start()`, `derafu_form_end()`, and so on. The value must not be empty.

`forms_path` is the directory where the application stores its own form definition files. It defaults to `resources/forms/` at the project root. This directory does not need to exist at installation time; the compiler pass only registers it if it is present when the container is compiled.

Writing form definition files

Form definitions are plain PHP files that return either an array or a callable. Place them inside `resources/forms/` at the project root (or inside `resources/forms/` within any registered bundle).

A static definition returns an associative array with `schema`, `uischema`, and optionally `data` keys:

```
// resources/forms/auth/login.form.php
return [
    'schema' => [
        'type' => 'object',
        'properties' => [
            'email'    => ['type' => 'string', 'format' => 'email'],
            'password' => ['type' => 'string'],
        ],
        'required' => ['email', 'password'],
    ],
    'uischema' => [
        'type'     => 'VerticalLayout',
        'elements' => [
            ['type' => 'Control', 'label' => 'Email',    'scope' =>
'#/properties/email'],
            ['type' => 'Control', 'label' => 'Password', 'scope' =>
'#/properties/password'],
        ],
    ],
    'data' => ['email' => '', 'password' => ''],
];
```

A dynamic definition returns a callable that receives a `$context` array and returns the same structure. This is useful when the form data or schema depends on values only known at runtime:

```
// resources/forms/user/edit.form.php
return function (array $context = []): array {
    return [
        'schema' => [...],
        'data'   => ['name' => $context['name'] ?? ''],
    ];
};
```

Files are addressed by their path relative to the forms directory, without the `.form.php` suffix. A file at `resources/forms/auth/login.form.php` is loaded with the key `auth/login`.

Loading and rendering forms in a controller

Inject `FormLoaderInterface` to load a form definition and `FormDataProcessorInterface` to validate a submission:

```
use Derafu\Form\Contract\FormDataProcessorInterface;
use Derafu\Form\Contract\FormLoaderInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
```

```
final class LoginController extends AbstractController
{
    public function __invoke(
        Request $request,
        FormLoaderInterface $formLoader,
        FormDataProcessorInterface $processor,
    ): Response {
        $form = $formLoader->load('auth/login');

        if ($request->isMethod('POST')) {
            $result = $processor->process($form, $request->request->all());

            if ($result->isValid()) {
                $data = $result->getProcessedData();
                // handle valid submission...
            }

            $form = $form->withData($result->getData(), $result->getErrors());
        }

        return $this->render('auth/login.html.twig', ['form' => $form]);
    }
}
```

In the Twig template, render the form using the prefixed helper functions:

```
{{ derafu_form(form) }}
```

Or, if you need to wrap the form in custom markup, use the start/end pair:

```
{{ derafu_form_start(form) }}
    {# custom markup #}
{{ derafu_form_end(form) }}
```

JavaScript widgets

The bundle ships three JavaScript modules, all registered automatically in the import map under the `derafu-form/` namespace.

`derafu-form/collection-widget.js` handles dynamic array/collection fields: adding new items, removing existing ones, and keeping the add/remove buttons enabled or disabled according to configured min/max constraints. It has no external dependencies and initialises itself on `DOMContentLoaded`. Import it on any page that contains a collection field:

```
import 'derafu-form/collection-widget.js';
```

`derafu-form/html-editor.js` integrates the Summernote WYSIWYG editor into `textarea` fields marked for rich-text editing. It requires jQuery and Summernote to be loaded beforehand — add them to your import map (or load them from a CDN) before importing this module.

`derafu-form/json-editor.js` integrates the JSONEditor library for fields that hold structured JSON. It requires the JSONEditor library to be available globally before this module runs.

Unless your forms include collection, rich-text, or JSON fields you do not need to import any of these modules manually; the base form rendering has no JavaScript dependency.

Last updated on 24/08/2026