Symfony DOCS

Home / Documentation

How to Use a Form without a Data Class

Edit this page

In most cases, a form is tied to an object, and the fields of the form get and store their data on the properties of that object. This is exactly what you've seen so far in this article with the `Task` class.

But sometimes, you may want to use a form without a class, and get back an array of the submitted data. The `getData()` method allows you to do exactly that:

```
// src/Controller/ContactController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
// ...

class ContactController extends AbstractController
{
    public function contact(Request $request): Response
    {
        $defaultData = ['message' => 'Type your message here'];
        $form = $this->createFormBuilder($defaultData)
            ->add('name', TextType::class)
            ->add('email', EmailType::class)
            ->add('message', TextareaType::class)
            ->add('send', SubmitType::class)
            ->getForm();

        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // data is an array with "name", "email", and "message" keys
            $data = $form->getData();
        }

        // ... render the form
        }
    }
```

By default, a form actually assumes that you want to work with arrays of data, instead of an object. There are exactly two ways that you can change this behavior and tie the form to an object instead:

1. Pass an object when creating the form (as the first argument to `createFormBuilder()` or the second argument to `createForm()`);
2. Declare the `data_class` option on your form.

If you *don't* do either of these, then the form will return the data as an array. In this example, since `$defaultData` is not an object (and no `data_class` option is set), `$form->getData()` ultimately returns an array.

Tip

You can also access POST values (in this case "name") directly through the request object, like so:

```
$request->request->get('name');
```

Be advised, however, that in most cases using the `getData()` method is a better choice, since it returns the data (usually an object) after it's been transformed by the Form component.

Adding Validation

The only missing piece is validation. Usually, when you call `$form->handleRequest($request)`, the object is validated by reading the constraints that you applied to that class. If your form is mapped to an object (i.e. you're using the `data_class` option or passing an object to your form), this is almost always the approach you want to use. See Validation for more details.

But if the form is not mapped to an object and you instead want to retrieve an array of your submitted data, how can you add constraints to the data of your form?

The answer is to set up the constraints yourself, and attach them to the individual fields. The overall approach is covered a bit more in this validation article, but here's a short example:

```
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Validator\Constraints\Length;
use Symfony\Component\Validator\Constraints\NotBlank;

public function buildForm(FormBuilderInterface $builder, array $options): void
```

```
{
    $builder
        ->add('firstName', TextType::class, [
            'constraints' => new Length(['min' => 3]),
        ])
        ->add('lastName', TextType::class, [
            'constraints' => [
                new NotBlank(),
                new Length(['min' => 3]),
            ],
        ])
    ;
}
```

Tip

If you are using validation groups, you need to either reference the `Default` group when creating the form, or set the correct group on the constraint you are adding:

```
new NotBlank(['groups' => ['create', 'update']]);
```

Tip

If the form is not mapped to an object, every object in your array of submitted data is validated using the `Symfony\Component\Validator\Constraints\Valid` constraint, unless you disable validation.

Caution

When a form is only partially submitted (for example, in an HTTP PATCH request), only the constraints from the submitted form fields will be evaluated.