



FREE eBook

LEARNING symfony-forms

Free unaffiliated eBook created from
Stack Overflow contributors.

#symfony-
forms

Table of Contents

About.....	1
Chapter 1: Getting started with symfony-forms.....	2
Remarks.....	2
Examples.....	2
Installation or Setup.....	2
Chapter 2: Example of Symfony Form Events.....	3
Remarks.....	3
Examples.....	3
onPostSubmit Event.....	3
FormEvents::PRE_SUBMIT.....	4
FormEvents::PRE_SET_DATA.....	5
Chapter 3: Forms.....	7
Syntax.....	7
Remarks.....	7
Examples.....	7
Create a simple form in a controller.....	7
Create a custom form type.....	8
Check if all fields are rendered in the template.....	9
How to deal with form options.....	9
Deal with form events.....	10
Credits.....	11

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [symfony-forms](#)

It is an unofficial and free symfony-forms ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official symfony-forms.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with symfony-forms

Remarks

This section provides an overview of what symfony-forms is, and why a developer might want to use it.

It should also mention any large subjects within symfony-forms, and link out to the related topics. Since the Documentation for symfony-forms is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting symfony-forms set up or installed.

Read [Getting started with symfony-forms online](https://riptutorial.com/symfony-forms/topic/10894/getting-started-with-symfony-forms): <https://riptutorial.com/symfony-forms/topic/10894/getting-started-with-symfony-forms>

Chapter 2: Example of Symfony Form Events

Remarks

Hang on For the More Symfony Form Events in the above example.

Examples

onPostSubmit Event

This is an Education Form in Symfony to take user education details. We wanted to apply validation on 2 fields, education end date and is currently studying.

On Post Submit Event, We will check two things

- 1 - if the user checks the checkbox of `is_currently studying` then end date should be empty
- 2 - On the other side, we have to make sure, if end date is not empty, then is currently studying check box should be unchecked.

```
/**
 * Class QualificationFormType
 * @package UsersBundle\Form\Type
 */
class QualificationFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title')
            ->add('institution')
            ->add('startDate', 'date', [
                'label' => 'Start Date',
                'widget' => 'single_text',
                'format' => 'dd-MM-yyyy',
                'required' => true,
                'constraints' => [
                    new Assert\NotBlank(),
                    new Assert\LessThan("today"),
                ],
                'trim' => true,
                'attr' => [
                    'maxlength' => '12',
                    'minlength' => '10',
                    'placeholder' => 'when did you start this education?',
                    'class' => 'form-control input-inline datepicker datePicker',
                    'data-provide' => 'datepicker',
                    'data-date-format' => 'dd-mm-yyyy',
                ],
            ],
    }
}
```

```

        'minViewMode' => '1'
    ],
    'label_attr' => [
        'class' => 'control-label',
    ],
])
->add('endDate', 'date', [
    'label' => 'End Date',
    'widget' => 'single_text',
    'format' => 'dd-MM-yyyy',
    'required' => false,
    'attr' => [
        'placeholder' => 'when did you end this education?',
        'class' => 'form-control input-inline datepicker datePicker',
        'data-provide' => 'datepicker',
        'data-date-format' => 'dd-mm-yyyy',
        'minViewMode' => '1'
    ],
    'label_attr' => [
        'class' => 'control-label',
    ],
])
->add('current', null, [
    'label' => ucfirst('I am currently studying'),
    'label_attr' => [
        'class' => 'control-label',
    ],
])
->add('save', 'submit')

;

$builder->addEventListener(FormEvents::POST_SUBMIT, [$this, 'onPostSubmit']);
}

function onPostSubmit(FormEvent $event) {
    $form = $event->getForm();
    $endDate = $form->get('endDate')->getData();
    $current = $form->get('current')->getData();
    If(!$current){
        if ($startDate>$endDate ) {
            $form['startDate']->addError(new FormError("Start Date cannot be greater
than end date..."));
        }
    }
}
}
}

```

FormEvents::PRE_SUBMIT

This example is about changing the form depending on decisions the user did with the form previously.

In my special case, I needed to disable a selectbox, if a certain checkbox wasn't set.

So we have the FormBuilder, where we'll set the `EventListener` on the `FormEvents::PRE_SUBMIT`

event. We're using this event, because the form is already set with the submitted data of the form, but we're still able to manipulate the form.

```
class ExampleFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $data = $builder->getData();
        $builder
            ->add('choiceField', ChoiceType::class, array(
                'choices' => array(
                    'A' => '1',
                    'B' => '2'
                ),
                'choices_as_values' => true,
            ))
            ->add('hiddenField', HiddenType::class, array(
                'required' => false,
                'label' => ''
            ))
            ->addEventListener(FormEvents::PRE_SUBMIT, function(FormEvent $event) {

                // get the form from the event
                $form = $event->getForm();

                // get the form element and its options
                $config = $form->get('choiceField')->getConfig();
                $options = $config->getOptions();

                // get the form data, that got submitted by the user with this request / event
                $data = $event->getData();

                // overwrite the choice field with the options, you want to set
                // in this case, we'll disable the field, if the hidden field isn't set
                $form->add(
                    'choiceField',
                    $config->getType()->getName(),
                    array_replace(
                        $options, array(
                            'disabled' => ($data['hiddenField'] == 0 ? true : false)
                        )
                    )
                );
            })
    }
}
```

FormEvents::PRE_SET_DATA

Requirement is to check if in a form, 'Online_date' field is blank or filled. If it is blank, then fill it with current date, on form load.

Controller calls '\$form->createForm()' with type "folder". In "FolderType", event subscriber "FolderSubscriber" is added.

Controller:

```

        $form = $this->createForm('folder', $folder, array(
            'action' => $this->generateUrl('folders_edit', array('id' => $folder-
>getId()))),
            'method' => 'post'
        ));

```

FolderType:

```

class FolderType extends AbstractType
{
    public function __construct( FolderSubscriber $folderSubscriber)
    {
        $this->folderSubscriber = $folderSubscriber;
    }

    public function buildForm(FormBuilderInterface $builder, array $options = array())
    {
        $builder ->add("onlineDate", "datetime", array( 'widget' => 'single_text'));
        $builder->addEventSubscriber($this->folderSubscriber);
    }
    public function getName()
    {
        return 'folder';
    }
}

```

FolderSubscriber: Gets called from FolderType; where it is registered as Event Subscriber

```

class FolderSubscriber implements EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        return array(
            FormEvents::PRE_SET_DATA => 'onPreSetData',
        );
    }

    public function onPreSetData(FormEvent $event)
    {
        $this->setDefaultOnlineDate($event);
    }

    protected function setDefaultOnlineDate(FormEvent $event)
    {
        $content = $event->getData();
        if (!$content->getOnlineDate() instanceof \DateTime){
            $content->setOnlineDate(new \DateTime());
        }
    }
}

```

Read Example of Symfony Form Events online: <https://riptutorial.com/symfony-forms/topic/5039/example-of-symfony-form-events>

Chapter 3: Forms

Syntax

- Form `createForm(string|FormTypeInterface $type, mixed $data = null, array $options = array())`
- FormBuilder `createFormBuilder(mixed $data = null, array $options = array())`

Remarks

You can "customize" Event of the process of **Form** component with a Form Event compatible with **Event Dispatcher** Component.

[Symfony Docs](#) :

The Form component provides a structured process to let you customize your forms, by making use of the EventDispatcher component. Using form events, you may modify information or fields at different steps of the workflow: from the population of the form to the submission of the data from the request.

Examples

Create a simple form in a controller

A form gives the user a way to change data in your application, in a structured way. To mutate a simple `array` of data, we create a form using a form builder:

```
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\NumberType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

// ...

function myAction (Request $request) {
    $data = array(
        'value' => null,
        'number' => 10,
        'string' => 'No value',
    );
    $form = $this->createFormBuilder($data)
        ->add('value', TextType::class, array('required' => false))
        ->add('number', NumberType::class)
        ->add('string', TextType::class)
        ->add('save', SubmitType::class)
        ->getForm();

    $form->handleRequest($request);
    if ($form->isValid()) {

        // $data is now changed with the user input
    }
}
```

```

        // Do something with the data
    }

    return $this->render(..., array(
        'form' => $form->createView(),
        // ...
    ));
}

```

In your template, render your form using the `form(...)` Twig function:

```

{# Render the form #}
{{ form(form) }}

```

It will, without styling, look something like below:

The image shows a rendered form with the following elements:

- A label "Value" followed by an empty text input field.
- A label "Number" followed by a text input field containing the value "10".
- A label "String" followed by a text input field containing the value "No value".
- A "Save" button below the other fields.

The labels, IDs, names and form tags are generated automatically. By default, the form submits to the current page with a POST request.

Create a custom form type

A custom form type is a class which defines a reusable form component. Custom form components can be nested to create complicated forms.

Instead of creating a form in the controller using a form builder, you can use your own type to make the code more readable, reusable and maintainable.

Create a class which represents your form type

```

// src/AppBundle/Form/ExampleType.php
namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\NumberType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class ExampleType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options) {
        $builder
            ->add('value', TextType::class, array('required' => false))
            ->add('number', NumberType::class)
            ->add('string', TextType::class)
            ->add('save', SubmitType::class)
        ;
    }
}

```

You can now use your form in controller:

```
use AppBundle\Form\ExampleType;

// ...

$form = $this->createForm(ExampleType::class, $data)
```

Check if all fields are rendered in the template

When rendering a form 'by hand', it can be useful to know if there are fields left to render or not. The function `isRendered()` from the *FormView* class returns `true` if there are still fields left to be rendered to the template.

This snippet prints `<h3>Extra fields</h3>` if there are fields left to be added to the template, followed by the fields themselves.

```
{% if not form.isRendered() %}
    <h3>Extra fields</h3>
    {{ form_rest(form) }}
{% endif %}
```

How to deal with form options

In this example, I created a form which is used to register a new user. In the options passed to the form, I give the different roles a user can have.

Creating a reusable class for my form with configured data class and an extra option that fills the choice field to pick a userrole:

```
class UserType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('firstName', TextType::class, array(
                'label' => 'First name'
            ))
            ->add('lastName', TextType::class, array(
                'label' => 'Last name'
            ))
            ->add('email', EmailType::class, array(
                'label' => 'Email'
            ))
            ->add('role', ChoiceType::class, array(
                'label' => 'Userrole',
                'choices' => $options['rolechoices']
            ))
            ->add('plain_password', RepeatedType::class, array(
                'type' => PasswordType::class,
                'first_options' => array('label' => 'Password'),
                'second_options' => array('label' => 'Repeat password')
            ))
            ->add('submit', SubmitType::class, array(
```

```

        'label' => 'Register user'
    ));
}

public function configureOptions(OptionsResolver $optionsResolver)
{
    $optionsResolver->setDefaults(array(
        'data_class' => 'WebsiteBundle\Entity\User',
        'rolechoices' => array()
    ));
}
}

```

As you can see, there's a default option added to the form named 'roleChoices'. This option is created and passed in the method to create a form object. See next code.

Creating a form-object in my controller:

```

$user = new User();
$roles = array(
    'Admin' => User::ADMIN_ROLE,
    'User' => User::USER_ROLE
);
$form = $this->createForm(UserType::class, $user, array(
    'rolechoices' => $roles
));

```

Deal with form events

To be able to deal with form events, it's important to attach the request, that is sent to a controller action after submitting a form, to the form created in that action.

```

public function registerAction(Request $request)
{
    $data = new ExampleObject();
    $form = $this->createForm(ExampleObjectType::class, $data);

    $form->handleRequest($request);
    if($form->isSubmitted() && $form->isValid()){
        // do something with form data
        return $this->redirectToRoute('route_name');
    }
    return $this->render('view.html.twig', array(
        'form' => $form->createView()
    ));
}

```

The request variable passed to the action is of type `Symfony\Component\HttpFoundation\Request`

Read Forms online: <https://riptutorial.com/symfony-forms/topic/4440/forms>

Credits

S. No	Chapters	Contributors
1	Getting started with symfony-forms	Community
2	Example of Symfony Form Events	Ajay Bisht , jkucharovic , KhorneHoly , Muhammad Taqi
3	Forms	Alfro , Hidde , jkucharovic , Mathieu Dorneval , rubenj