élao

Agence Web - iPhone & iPad Apps - Experts Symfony

Form Sheets

Simple Form

```php
<?php
/** Build a form */
$form = $this->createFormBuilder()->add('name')->getForm();

/** Bild a form linked to an entity */
$form = $this->createFormBuilder($people)->add('name')->getForm();
```

Form Class

```php
<?php
class ProductType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('name', 'text');
    }

    public function getName()
    {
        return 'product';
    }
}
```

One to Many Relation

```php
<?php
$builder->add('category', 'entity', array(
    'class' => 'Demo\\MyBundle\\Entity\\Category'
));
```

Many to Many Relation

```php
<?php
$builder->add('tags', 'entity', array(
    'class' => 'Demo\\MyBundle\\Entity\\Tag',
    'multiple' => true,
    'expanded' => true
));
```

Embed Form

```php
<?php
$builder->add('category', new CategoryType());
```

Embed Collection

```php
<?php
// Elao\DemoBundle\Entity\Contact
class Contact
{
    public $phones = array();

    public function __construct()
    {
        $this->phones = array(new Phone(), new Phone(), new Phone());
    }
}
```

```php
<?php
// Elao\DemoBundle\Entity\Phone
class Phone
{
    public $number;
}
```

```php
<?php
// Elao\DemoBundle\Form\PhoneType
$builder->add('number', 'text');
```

```php
<?php
// Elao\DemoBundle\Form\ContactType
$builder->add('phones', 'collection', array(
    'type' => new PhoneType()
));
```

Binding

```php
<?php
public function newAction()
{
    $request = $this->getRequest();
    if ('POST' === $request->getMethod()) {
        $form->bind($request);
        if ($form->isValid()) {
            // treatment
        }
    }
}
```

Widgets

Widget	Parent	Common Options
*		data_class, required, read_only, disabled, max_length, pattern, property_path, mapped, by_reference, label, attr, label_attr, translation_domain
text	field	
textarea	text	
password	text	always_empty
checkbox	field	value
radio	checkbox	
file	field	
choice	field	multiple, expanded, choices, preferred_choices, empty_value
hidden	field	
number	field	precision, grouping, rounding_mode
money	field	precision, grouping, divisor, currency
percent	field	precision, type
datetime	field	input, model_timezone, view_timezone, format, date_format, widget, date_widget, time_widget, with_seconds, by_reference
birthday	date	years
date	field	years, months, days, widget, input, format, model_timezone, view_timezone, by_reference
time	field	hours, minutes, seconds, widget, input, with_seconds, model_timezone, view_timezone, by_reference
locale	choice	
language	choice	
country	choice	
timezone	choice	
entity		
url	text	default_protocol
repeated		type, options, first_options, second_options, first_name, second_name
collection		allow_add, allow_delete, prototype, prototype_name, type, options
email	text	
search	text	
integer	field	precision, grouping, rounding_mode

TWIG

Form Rendering

```twig
<form action="{{ path('store') }}" method="post" {{ form_enctype(form) }}>
    {{ form_widget(form) }}
    <input type="submit" />
</form>
```

Form Template

```twig
{# src/Acme/StoreBundle/Resources/views/Default/index.html.twig #}
<form action="{{ path('store') }}" method="post" {{ form_enctype(form) }}>
    {{ form_errors(form) }}

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

    {{ form_rest(form) }}

    <input type="submit" />
</form>
```

Embed Form

```twig
{{ form_row(form.price) }}

<h3>Category</h3>
<div class="category">
    {{ form_row(form.category.name) }}
</div>

{{ form_rest(form) }}
```

Form Theming

```twig
{# src/Elao/BlogBundle/Resources/public/views/Form/fields.html.twig #}
{% block form_row %}
{% spaceless %}
    <div class="form_row">
        {{ form_label(form) }}
        {{ form_errors(form) }}
        {{ form_widget(form) }}
    </div>
{% endspaceless %}
{% endblock form_row %}
```

```twig
{# src/Elao/BlogBundle/Resources/public/views/show.html.twig #}
{% form_theme form 'ElaoBlogBundle:Form:fields.html.twig'}
```

OR

```yaml
# app/config/config.yml
twig:
    form:
        resources: ['ElaoBlogBundle:Form:fields.html.twig']
```

Validation

YML File

```yaml
# src/Elao/DemoBundle/Resources/config/validation.yml
Elao\DemoBundle\Entity\Thing:
    properties:
        foobar:
            - NotBlank
    getters:
        tokenValid:
            - True: { message: "The token is invalid" }
```

Annotations

```php
<?php
class Thing
{
    /**
     * @Assert\NotBlank
     */
    protected $foobar;

    /**
     * @Assert\True(message={"The token is invalid"})
     */
    public function isTokenValid()
    {
        return $this->token == $this->generateToken();
    }
}
```

PHP

```php
<?php
use Symfony\Component\Validator\Mapping\ClassMetadata;
use Symfony\Component\Validator\Constraints\NotBlank;

class Thing
{
    public static function loadValidatorMetadata(ClassMetadata $metadata)
    {
        $metadata->addPropertyConstraint('foobar', new NotBlank());
        $metadata->addGetterConstraint('tokenValid', new True(array(
            'message' => 'The token is invalid',
        )));
    }
}
```

Constraints

All	
Blank	message
Callback	methods
Choice	choices, callback, multiple, strict, min, max, message, multipleMessage, minMessage, maxMessage
Collection	fields, allowExtraFields, allowMissingFields, extraFieldsMessage, missingFieldsMessage
Count	minMessage, maxMessage, exactMessage, min, max
Country	message
Date	message
DateTime	message
Email	message, checkMX, checkHost
False	message
File	maxSize, mimeTypes, ...
Image	mimeTypes, minWidth, maxWidth, maxHeight, minHeight, ...
Ip	version, message
Language	message
Length	maxMessage, minMessage, exactMessage, max, min, charset
Locale	message
NotBlank	message
NotNull	message
Null	message
Range	minMessage, maxMessage, invalidMessage, min, max
Regex	message, pattern, htmlPattern, match
Time	message
True	message
Type	message, type
Url	message, protocols

Callback Constraints

```php
<?php
// src/Acme/BlogBundle/Entity/Author.php
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Component\Validator\ExecutionContext;

/**
 * @Assert\Callback(methods={"isAuthorValid"})
 */
class Author
{
  private $firstName;

    public function isAuthorValid(ExecutionContext $context)
    {
        // somehow you have an array of "fake names"
        $fakeNames = array();

        // check if the name is actually a fake name
        if (in_array($this->getFirstName(), $fakeNames)) {
            $context->addViolationAtSubPath('firstname', 'Error', array(), null);
        }
    }
}
```

Validation Groups

```php
<?php
// src/Acme/BlogBundle/Entity/User.php
namespace Acme\BlogBundle\Entity;

use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Validator\Constraints as Assert;

class User implements UserInterface
{
    /**
    * @Assert\Email(groups={"registration"})
    */
    private $email;

    /**
    * @Assert\NotBlank(groups={"registration"})
    * @Assert\MinLength(limit=7, groups={"registration"})
    */
    private $password;

    /**
    * @Assert\MinLength(2)
    */
    private $city;
}
```

```php
// with the validator component
$errors = $validator->validate($author, array('registration'));
```

```php
// inside a Form class
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

public function setDefaultOptions(OptionsResolverInterface $resolver)
{
    $resolver->setDefaults(array(
        'validation_groups' => array('registration')
    ));
}
```

Create custom constraints

Constraint Class

```php
<?php
use Symfony\Component\Validator\Constraint;

/**
 * @Annotation
 */
class Foobar extends Constraint
{
    public $message = 'This value should be "foobar"';
}
```

Constraint Validator

```php
<?php
use Symfony\Component\Validator\Constraint;
use Symfony\Component\Validator\ConstraintValidator;

class FoobarValidator extends ConstraintValidator
{
    public function validate($value, Constraint $constraint)
    {
        if ($value !== 'foobar') {
            $this->context->addViolation($constraint->message);
            return false;
        }

        return true;
    }
}
```