Symfony

Working with Symfony2 Forms

There are 3 ways!

1 DIRECTLY IN THE CONTROLLER

Simple and Easy

Translate user-submitted data back to the properties of an object

Data Class

Entity

src/Acme/TaskBundle/Entity/Task.php

```php
namespace Acme\TaskBundle\Entity;

class Task
{
   protected $task;
   protected $dueDate;

   public function getTask()
   {
      return $this->task;
   }
   public function setTask($task)
   {
      $this->task = $task;
   }

   public function getDueDate()
   {
      return $this->dueDate;
   }
   public function setDueDate(\DateTime $dueDate = null)
   {
      $this->dueDate = $dueDate;
   }
}
```

Create / Handle Submission

Controller

src/Acme/TaskBundle/Controller/DefaultController.php

```php
namespace Acme\TaskBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Acme\TaskBundle\Entity\Task;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller
{
   public function newAction(Request $request)
   {
     // just setup a fresh $task object
     $task = new Task();

     $form = $this->createFormBuilder($task)
     ->add('task', 'text')
     ->add('dueDate', 'date')
     ->getForm();

     $request = $this->getRequest();

     if ($request->getMethod() == 'POST') {
        $form->bindRequest($request);

        if ($form->isValid()) {
           // ...
           return $this->redirect($this->generateUrl('task_success'));
        }
     }
     return $this->render('AcmeTaskBundle:Default:new.html.twig',
        array(
           'form' => $form->createView(),
        ));
   }
}
```

create the form

field type

determine which HTML form tag will be rendered for the field

call validation

prevent re-post data

render the form (view)

Render

View

src/Acme/TaskBundle/Resources/views/Default/new.html.twig

TWIG

```twig
<form action="{{ path('task_new') }}" method="post"
                    {{ form_enctype(form) }}>
  {{ form_widget(form) }}

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

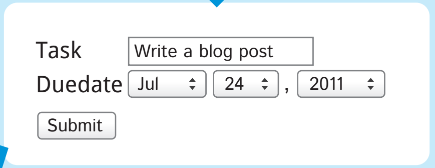
OR

src/Acme/TaskBundle/Resources/views/Default/new.html.php

PHP

```php
<form action="<?php echo $view['router']->generate('task_new') ?>"
    method="post" <?php echo $view['form']->enctype($form) ?> >
  <?php echo $view['form']->widget($form) ?>

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



Task Write a blog post

Duedate Jul 24 , 2011

Submit

You can render the entire form with just one line (like above) or customize every part of your form.

Everything can be customized (Using fragments and themes)

To customize any part of how a form renders:

Twig: override the appropriate block

PHP: override the existing template by creating a new one

Validate

Validation is done by adding a set of rules (called constraints) to a class.

Calling $form->isValid() is a shortcut that asks the $task object whether or not it has valid data.

src/Acme/TaskBundle/Entity/Task.php

Annotations

```php
use Symfony\Component\Validator\Constraints as Assert;

class Task
{
   /**
    * @Assert\NotBlank()
    */
   public $task;

   /**
    * @Assert\NotBlank()
    * @Assert\Type("\DateTime")
    */
   protected $dueDate;
}
```

OR

src/Acme/TaskBundle/Resources/config/validation.yml

YAML

```yaml
Acme\TaskBundle\Entity\Task:
  properties:
    task:
      - NotBlank: ~
    dueDate:
      - NotBlank: ~
      - Type: \DateTime
```

You can use annotations, YAML, XML or PHP.

See http://symfony.com/doc/current/book/forms.html

Working with Symfony2 Forms

There are 3 ways!

Symfony

2 CREATING FORM CLASSES (standalone PHP class)

Best way (better practice, reusable)

Create

Form Class

src/Acme/TaskBundle/Form/Type/TaskType.php

```php
namespace Acme\TaskBundle\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilder;

class TaskType extends AbstractType
{
  public function buildForm(FormBuilder $builder, array $options)
  {
    $builder->add('task');
    $builder->add('dueDate', null, array('widget' => 'single_text'));
  }

  public function getDefaultOptions(array $options)
  {
     return array(
       'data_class' => 'Acme\TaskBundle\Entity\Task',
     );
  }

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

data_class

name of the class that holds the underlying data
- for embedded forms
- enables form to see type definitions of the class

Always define! (best practice)

Handle Submission

Controller

src/Acme/TaskBundle/Controller/DefaultController.php

```php
// add this new use statement at the top of the class
use Acme\TaskBundle\Form\Type\TaskType;

public function newAction()
{
  $task = // ...
  $form = $this->createForm(new TaskType(), $task);

  // ...
}
```

Data Class

Entity

src/Acme/TaskBundle/Entity/Task.php

```php
namespace Acme\TaskBundle\Entity;

class Task
{
  protected $task;
  protected $dueDate;

  public function getTask()
  {
    return $this->task;
  }
  public function setTask($task)
  {
    $this->task = $task;
  }

  // ...
}
```

Render

View

src/Acme/TaskBundle/Resources/views/Default/new.html.twig

TWIG

```twig
<form action="{{ path('task_new') }}" method="post"
                    {{ form_enctype(form) }}>
  {{ form_widget(form) }}

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

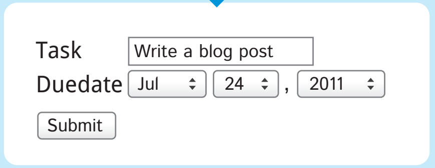
OR

src/Acme/TaskBundle/Resources/views/Default/new.html.php

PHP

```php
<form action="<?php echo $view['router']->generate('task_new') ?>"
    method="post" <?php echo $view['form']->enctype($form) ?> >
  <?php echo $view['form']->widget($form) ?>

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



You can render the entire form with just one line (like above) or customize every part of your form.

Everything can be customized (Using fragments and themes)

To customize any part of how a form renders:
Twig: override the appropriate block
PHP: override the existing template by creating a new one

Isn't a form responsibility

Validate

Validation is done by adding a set of rules (called constraints) to a class.

Calling $form->isValid() is a shortcut that asks the $task object whether or not it has valid data.

src/Acme/TaskBundle/Entity/Task.php

Annotations

```php
use Symfony\Component\Validator\Constraints as Assert;

class Task
{
  /**
   * @Assert\NotBlank()
   */
  public $task;

  /**
   * @Assert\NotBlank()
   * @Assert\Type("\DateTime")
   */
  protected $dueDate;
}
```

OR

src/Acme/TaskBundle/Resources/config/validation.yml

YAML

```yaml
Acme\TaskBundle\Entity\Task:
  properties:
    task:
      - NotBlank: ~
    dueDate:
      - NotBlank: ~
      - Type: \DateTime
```

You can use annotations, YAML, XML or PHP.
See http://symfony.com/doc/current/book/forms.html

Working with Symfony2 Forms

There are 3 ways!

Symfony

3 USING A FORM WITHOUT A DATA CLASS

For simple forms (get back an array of the submitted data)

Create / Handle Submission

Controller

src/Acme/ContactBundle/Controller/ContactController.php

```
// make sure you've imported the Request namespace above the class
use Symfony\Component\HttpFoundation\Request
// ...

public function contactAction(Request $request)
{
  $defaultData = array('message' => 'Type your message here');
  $form = $this->createFormBuilder($defaultData)
    ->add('name', 'text')
    ->add('email', 'email')
    ->add('message', 'textarea')
    ->getForm();

    if ($request->getMethod() == 'POST') {
      $form->bindRequest($request);

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

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

By default, $form->getData() return an array instead of an object. There are two ways to 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.

Validate

Validation is done by adding the constraints yourself, and pass them into your form.

Without Form Class

```
// import the namespaces above your controller class
use Symfony\Component\Validator\Constraints\Email;
use Symfony\Component\Validator\Constraints\MinLength;
use Symfony\Component\Validator\Constraints\Collection;

$collectionConstraint = new Collection(array(
  'name' => new MinLength(5),
  'email' => new Email(array(
              'message' => 'Invalid email address')),
));

// create a form, no default values, pass in the constraint option
$form = $this->createFormBuilder(null, array(
  'validation_constraint' => $collectionConstraint,
))->add('email', 'email')
  // ...
;
```

With Form Class

```
namespace Acme\TaskBundle\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilder;
use Symfony\Component\Validator\Constraints\Email;
use Symfony\Component\Validator\Constraints\MinLength;
use Symfony\Component\Validator\Constraints\Collection;

class ContactType extends AbstractType
{
  // ...

  public function getDefaultOptions(array $options)
  {
    $collectionConstraint = new Collection(array(
      'name' => new MinLength(5),
      'email' => new Email(array(
                  'message' => 'Invalid email address')),
    ));

    return array('validation_constraint' => $collectionConstraint);
  }
}
```

If you're using a form class, override the getDefaultOptions method to specify the option

When you call $form->isValid(), the constraints setup here are run against your form's data.

Accessing POST values directly through the request object

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

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 framework.

Render

View

src/Acme/TaskBundle/Resources/views/Default/new.html.twig

TWIG

```
<form action="{{ path('task_new') }}" method="post"
                    {{ form_enctype(form) }}>
  {{ form_widget(form) }}

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

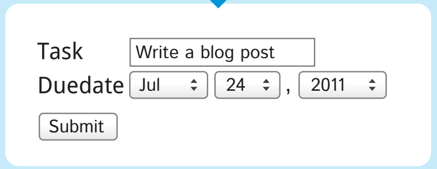
OR

src/Acme/TaskBundle/Resources/views/Default/new.html.php

PHP

```
<form action="<?php echo $view['router']->generate('task_new') ?>"
    method="post" <?php echo $view['form']->enctype($form) ?> >
  <?php echo $view['form']->widget($form) ?>

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



Task: Write a blog post
Duedate: Jul 24 , 2011
Submit

You can render the entire form with just one line (like above) or customize every part of your form.

Everything can be customized (Using fragments and themes)

To customize any part of how a form renders:
Twig: override the appropriate block
PHP: override the existing template by creating a new one