Cheatography

Symfony Cheat Sheet

by Mikael Peigney (Mika56) via cheatography.com/3808/cs/821/

Ressources

Symfony Homepage	http://symfony.com/
KnpBundles	http://knpbundles.com/

Folders

/app	Contains configuration, cache, logs, ... Everything that is not the source code
/src	Contains our bundles
/vendor	Contains all libs we use in our project
/web	Contains all "public" files, like images, CSS, Javascript, ... Also contains the "main controler", app.php

Bundle folders

/Controller	Contains our controlers
/DependencyInjection	Contains informations about our bundle
/Entity	Contains our models
/Form	Contains our forms
/Resources	Contains config files, public files and view (Twig) files
/Tests	Contains our Unit Test files

Composer

Install	php -r "eval('?>'.file_get_contents('http://getcomposer.org/installer'));"
Update composer	php composer.phar self-update
Update bundles	php composer.phar update

Forms generation

add($child, $type, $option)	Adds a form filed, where **$child** is the member, **$type** is the input type and **$options** the type's options
remove($name)	Removes the field with the given name
get($name)	Rturns a child by name
has($name)	Returns whether a field with the given name exists

To be used on a **FormBuilderInterface** (typically, **$builder** from **<Entity>Type::buildForm**

Forms field types

text/textarea	Standard input with type=text or textarea
email/url	

Console

generate:bundle	Generates a bundle
cache:clear	Clears application cache
doctrine:generate:entity	Generates a new Doctrine entity
doctrine:generate:entities	Generates an entity's methods with it's updated content
doctrine:schema:create	Creates database's schema
doctrine:schema:update	Update the database with its new schema. Use with --dump-sql and --force
doctrine:fixtures:load	Loads fixtures into database. Use it with --append to append datas instead of replacing
doctrine:generate:form	Generates a <Entity>Type form

Routes

HelloTheWorld	Route name
pattern	The pattern to match
defaults	The bundle and controller to use
requirements	List of required parameters and their format

Example:

```
HelloTheWorld:
  pattern: /hello-world/{name}.{format}
  defaults: { _controller: MyBundle:Blog:index, format: html }
  requirements:
    name: \w
    format: html|xml
```

Request parameters

$_GET['tag']	$request->query->get('tag')
$_POST['tag']	$request->request->get('tag')
$_COOKIE['tag']	$request->cookies->get('tag')
$_SERVER['tag']	$request->server->get('tag')
Route parameters	$request->attributes->get('tag') or $tag

Note
$request = $this->get('request');

Twig

{{ id }}	Show variable *id* content
{{ id|upper }}	Show variable *id* content in uppercase. Works with a lot of other filters

Doctrine

@ORM\Entity((repositoryClass="Me\Bundle\Entity\Repository")	Declares a class as an ORM entity, with Me\Bundle\Entity\Repository being its repository
@ORM\Table(name="table_name")	Changes a table name
@ORM\Column(type="string")	Declares an attribute as being a table column
@ORM\OneToOne(targetEntity="Me\Bundle\Entity\Column", cascade={"persist"})	Declares a One-To-One relation. *cascade* cascades some operations (such as *persist* or *remove*) to the relation
@ORM\OneToMany(targetEntity="Me\Bundle\Entity\Column")	Declares a One-To-Many relation
@ORM\ManyToOne(targetEntity="Me\Bundle\Entity\Column")	Declares a Many-To-One relation
@ORM\ManyToMany(targetEntity="Me\Bundle\Entity\Column")	Declares a Many-To-Many relation
@ORM\HasLifecycleCallbacks()	Declares that the Entity has callbacks

Doctrine Column parameters

type	Column type
name	Column name
length	Column length, only for string type
unique	Defines the column as unique
nullable	Allows the column to contain *null*
precision	Number of total digits, for decimal type
scale	Number of digits after point, for decimal type

Doctrine column types

string	Every string up to 255 chars
integer	Numbers up to 2.147.483.647
boolean	Boolean values; *true* and *false*
decimal	Decimal numbers
date/time/datetime	
text	Strings with no chars limit
object/array	Stores a PHP object/array with serialize/unserialize
float	Number with floating-point

Standard type=text field with proper validation

integer/number/percent
type=text field, with validation

money
type=text field with currency symbol near the field

password
type=password field

search
type=search field

choice
Multi-usage field. Can make select, radio or checkboxes

entity
Creates a field with all entities from one class

country/language/locale/timezone

date/datetime/time

birthday
Same as datetime, but with more recent years

checkbox/radio

file

collection/repeated

hidden

csrf

{% if condition %} {% elseif condition %} {% else %} {% endif %}
if/elseif/else structure

{% render url('latest_articles', { 'max': 3 }) %}
Renders another action

Doctrine Query Builder

$query->select('a')
Set columns to get

$query->addSelect('a')
Adds columns to get

$query->leftJoin('alias.column', 'column_alias')
Joins an Entity to the request, just like a **LEFT JOIN**

$query->where('a.column = ?')
Sets the **WHERE** conditions

$query->addWhere('a.column = ?')

$query->orWhere('a.column = ?')

$query->setParameter(1, $something)
Sets a parameter. First argument can be the parameter position or name.

$query->setParameters(array('name' => $value,))

$query->groupBy('a.column')
Sets the **GROUP BY** DQL value

$query->addGroupBy('a.column')

$query->orderBy('a.column')
Sets the **ORDER BY** DQL value

$query->addOrderBy('a.column')

Note :
$query = $this->createQueryBuilder('alias');

Doctrine Callbacks

PrePersist
Executed just before a *persist()*. Therefore, the *$id* isn't available, but all changes made to the Entity will be persisted in database

PostPersist
Executed avec a *flush()* which had a *persist()* on that Entity. *$id* is now available, but changes aren't saved

PreUpdate
Executed just before a *flush()*

PostUpdate

PreRemove
Executed before a *flush()* with a *remove()* on that Entity

PostRemove
Executed after a *flush()*. *$id* is not available anymore

PostLoad
Executed after the Entity has been loaded or reloaded (*refresh()*)

Doctrine Extensions

Tree
This extension automates the tree handling process and adds some tree specific functions on repository

Translatable
Gives you a very handy solution for translating records into diferent languages. Easy to setup, easier to use

Sluggable
Urlizes your specified fields into single unique slug

Timestampable
Updates date fields on create, update and even property change

Loggable
Helps tracking changes and history of objects, also supports version managment

Sortable	Makes any document or entity sortable
Softdeletable	Allows to implicitly remove records
Uploadable	Provides file upload handling in entity fields

Add the following to **composer.json**:
"stof/doctrine-extensions-bundle": "dev-master"

Cheatographer

Mikael Peigney (Mika56)
cheatography.com/mika56/

Cheat Sheet

This cheat sheet was published on 1st February, 2013 and was last updated on 5th February, 2013.