

Entity Manager in a controller

<code>\$this->getDoctrine()->getManager();</code>	Get default entity manager
<code>\$this->getDoctrine()->getManagerForClasses(MyEntity::class)</code>	Get entity specific manager

Repositories in a controller

`$this->getDoctrine()->getRepository(MyEntity::class);`

Doctrine Query

<code>getResult(\$hydrationMode = self::HYDRATE_OBJECT)</code>	Retrieve a collection
<code>getSingleResult(\$hydrationMode = null)</code>	Retrieve a single result or exception
<code>getOneOrNullResult(\$hydrationMode = null)</code>	Retrieve a result or NULL
<code>getArrayResult()</code>	Retrieve an array
<code>getScalarResult()</code>	Retrieves a flat/rectangular result set of scalar values
<code>getSingleScalarResult()</code>	Retrieves a single scalar value or exception
<code>AbstractQuery::HYDRATE_OBJECT</code>	Object graph default
<code>AbstractQuery::HYDRATE_ARRAY</code>	Array graph
<code>AbstractQuery::HYDRATE_SCALAR</code>	Flat rectangular result with scalars
<code>AbstractQuery::HYDRATE_SINGLE_SCALAR</code>	Single scalar value
<code>AbstractQuery::HYDRATE_SIMPLEOBJECT</code>	Very simple object fast

`setCacheMode($cacheMode)`

<code>Cache::MODE_GET</code>	may read, not write
<code>Cache::MODE_PUT</code>	no read, write all
<code>Cache::MODE_NORMAL</code>	read and write

Doctrine Query (cont)

<code>Cache::MODE_REFRESH</code>	no read, only refresh
<code>setCacheable(\$trueFalse)</code>	Enable/disable result caching

Magic repository functions

<code>find(\$id)</code>	Returns one entity
<code>findOneBy(\$criteria, \$orderBy)</code>	Returns one entity
<code>findBy(\$criteria, \$orderBy, \$limit, \$offset)</code>	Returns collection
<code>\$criteria = ['field' => 'value']</code>	Single field / value
<code>\$criteria = ['field' => ['value1', 'value2']]</code>	Single where-in
<code>\$criteria = ['field1' => 'value', 'field2' => 'value2']</code>	multiple fields / values
<code>\$repository = \$this->getDoctrine()->getRepository(Product::class);</code> <code>\$product = \$repository->findOneBy(['name' => 'Keyboard']);</code>	

Doctrine criteria - filtering collections

<code>where(\$Expression)</code>	Replaces previous statement
<code>[and/or]Where(\$Expression)</code>	Adds AND /OR where statement
<code>orderBy(\$array)</code>	Sets OrderBy
<code>setFirstResult(\$firstResult)</code>	Sets first result
<code>setMaxResult(\$max)</code>	Sets Maximum results
<code>\$userCollection = \$group->getUsers();</code> <code>\$criteria = Criteria::create()</code> <code>->where(Criteria::expr()->eq("birthday", "1982-02-17"))</code> <code>->orderBy(array("username" => Criteria::ASC))</code> <code>->setFirstResult(0)</code> <code>->setMaxResults(20)</code> <code>;</code> <code>\$birthdayUsers = \$userCollection->matching(\$criteria);</code>	

Doctrine ExpressionBuilder and Expr

andX (\$arg1, \$arg2, ...)	Multiple arguments AND
orX (\$arg1, \$arg2, ...)	Multiple arguments OR
[n]eq (\$field, \$value)	[Not] Equal
gte	Greater than [or equal]
lte	Less than [or equal]
isNull (\$field)	Is Null
[not]in (\$field, array \$values)	[not] In
memberOf (\$field, \$value)	ExpressionBuilder only Member of
isMemberOf (\$field, \$value)	Expr only Member of
isNotNull (\$field)	Expr only Is not Null
between (\$field, \$x, \$y)	Expr only Between \$x and \$y
trim (\$field)	Expr only Trim
concat (\$x, \$y)	Expr only Concat
literal (\$string)	Expr only Literal
lower (\$field)	Expr only Lower case
upper (\$field)	Expr only Upper case
count (\$field)	Expr only count

Doctrine Expr

```
Criteria::expr()->orX(
Criteria::expr()->eq('id', $facilityId),
Criteria::expr()->eq('active', TRUE)
);
```

Doctrine ExpressionBuilder

```
$qb = $this->createQueryBuilder('f');
```

```
$qb->where($qb->expr()->eq('id', '?arg1'))
->setParameter('arg1', $facilityId);
```

```
$qb->select($qb->expr()->substring('o.orderNumber', 4, 3));
```

```
$qb->select($qb->expr()->concat('p1.surname', $qb->expr()->literal(' '), 'p1.firstName');
```

Doctrine Query Builder

```
setParameter($parameter, $value)
```

```
addCriteria(Criteria $criteria)
```

```
[get/set]MaxResults($maxResults)
```

```
[get/set]FirstResult($firstResult)
```

```
getQuery()
```

Doctrine Query Builder (cont)

```
add($dqlPartName, $dqlPart, $append = false)
```

\$dqlPartName: select, from, join, set, where, groupBy, having, orderBy

Wrappers for add():

```
[add]select($select= null)
```

```
delete($delete = null, $alias = null)
```

```
update($update = null, $alias = null)
```

```
set($key, $value)
```

```
from($from, $alias, $indexBy = null)
```

```
[inner/left]join($join, $alias, $conditionType = null, $condition = null,
$indexBy = null)
```

```
[and/or]where($where)
```

```
[add]groupBy($groupBy)
```

```
[and/or]having($having)
```

```
[add]orderBy($field, $order = null)
```

Doctrine raw SQL

```
$Connection= $EntityManager->getConnection();
```

Get connection

```
$Statement = $Connection->prepare(string $sql);
```

Prepare statement from string

```
$Statement->bindValue(string|int $param, $value, $type = null);
```

Bind a value to a corresponding name

```
$Statement->execute(?array $params = null);
```

Execute the statement

```
$Statement->fetchAll(int $fetchStyle = PDO::FETCH_BOTH);
```

Fetch all results

```
$Statement->fetchOne(int $fetchStyle = PDO::FETCH_BOTH);
```

Fetch single result

```
$Statement->rowCount();
```

Return the number of rows

Doctrine raw SQL (cont)

```
$Statement->free(): void          Free stored result memory

$connection = $entityManager->getConnection();

$sql = 'SELECT * FROM events WHERE start_date >= :startdate';

$statement = $conn->prepare($sql);
$statement->bindValue('startdate', $startDate->format('Y-m-d H:i:s'));
$statement->execute();

return $statement->fetchAll();
```

Authorization

IS_AUTHENTICATE-D_FULLY	User has successfully authenticated, not via 'remember me cookie'
IS_AUTHENTICATE-D_REMEMBERED	All logged in users
IS_REMEMBERED	Only users authenticated using the remember me functionality
IS_IMPERSONATOR	When the current user is impersonating another user in this session

All roles you assign to a user must begin with the ROLE_ prefix in order for the default symfony RoleVoter to vote. Other prefixes require a custom voter.

Authorization in a controller

```
# Example of using wrapper #
public function hello($name)
{
    // The second parameter is used to specify on what object the role is
    // tested.
    $this->denyAccessUnlessGranted('ROLE_ADMIN', null, 'Unable to
    access this page!');
    // ...
}

# Example of using AuthorizationChecker
use Symfony\Component\Security\Core\Authorization\AuthorizationCheckerInterface;
use Symfony\Component\Security\Core\Exception\AccessDeniedException;
public function hello($name, AuthorizationCheckerInterface $authChecker)
{
```

Authorization in a controller (cont)

```
if (false === $authChecker->isGranted('ROLE_ADMIN')) {
    throw new AccessDeniedException('Unable to access this page!');
}
// ...
}

# Example of using annotation #
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;
/**
 * @Security("has_role('ROLE_ADMIN')")
 */
public function hello($name)
{
    // ...
}
```

Persisting Entities

```
// get the entity manager that manages this entity
$em = $this->getDoctrine()->
getManagerForClass(MyEntity::class);
// create a new entity
$entity = new MyEntity();
// populate / alter properties
$entity->setName('Default Entity');
// tell Doctrine you want to (eventually) save
$em->persist($entity);
// actually executes the queries
$em->flush();
```

Authorization via security.yaml

```
# config/packages/security.yaml
security:
    access_control:
        # public access only for /
```

Authorization via security.yaml (cont)

```
- { path: ^/$, roles: PUBLIC_ACCESS }
# public access to login page /login
- { path: ^/login, roles: PUBLIC_ACCESS }
# or require ROLE_ADMIN or ROLE_USER_ADMIN for
/admin/users*
- { path: ^/admin/users', roles: [ROLE_USER_ADMIN, ROLE_A-
DMIN] }
# require ROLE_ADMIN for /admin*
- { path: ^/admin, roles: ROLE_ADMIN }
# authenticated users only to the rest /*
- { path: ^/, roles: IS_AUTHENTICATED_FULLY
```

No limit on amount of URL patterns. Each is a regular expression.
First match will be used.

Prepend the path with ^ to ensure only URLs beginning with the
pattern are matched.

Authorization in template

```
{% if is_granted('ROLE_ADMIN') %}
<a href="#">Delete</a>
{% endif %}
```

Create a new form in a controller

```
# Create a new form with a default name #
$form = $this->createForm($data)
->add('dueDate', null, array(
'widget' => 'single_text'))
->add('save', SubmitType::class)
->getForm();
# Create a form with a non-default name #
$form = $this->container->get('form.factory')
->createNamedBuilder(
'form1', FormType::class, $data)
->add('dueDate', null, array(
'widget' => 'single_text'))
->add('save', SubmitType::class)
->getForm();
```

Create a form from a class in a controller

```
# Create a form from a form class with a default name #
$form = $this->createForm(TaskForm::class, $data, array(
'action' => $this->generateUrl('target_route'),
'method' => 'GET',
));
# Create a form from a form class with a non-default name #
$form = $this->container->get('form.factory')
->createNamed('form1', TaskForm::class, $data, array(
'action' => $this->generateUrl('target_route'),
'method' => 'GET',
));
```

Form Options

'action' => "	Where to send the form's data on submission (usually a URI)
'allow_extra_fields' => false	Allow additional fields to be submitted
'error_mapping' => array('match- ingCityAndZipCode' => 'city')	Modify the target of a validation error
'extra_fields_message' => 'This form should not contain extra fields.'	Validation error message if additional fields are submitted
'inherit_data' => false	Inherited data from parent form or not
'method' => 'POST'	HTTP method used to submit the form
'post_max_size_message' => 'The uploaded file was too large.'	Validation message for size of post form data

Form Options (cont)

'validation_groups' => Disable the Validation of Submitted Data
false

```
public function configureOptions(OptionsResolver $resolver) {
    $resolver->setDefaults(array(
        # Options go here #
    ));
}
```

Log Levels / Console Verbosity (OutputInterface)

emergency() System is unusable.

alert() Action must be taken immediately.

critical() Critical conditions.

error() Runtime errors that do not require immediate action but should typically be logged and monitored. VERBOSITY_QUIET (-1) -q / stderr

warning() Exceptional occurrences that are not errors. VERBOSITY_NORMAL (0) (non) / stdout

notice() Normal but significant events. VERBOSITY_VERBOSE (1) -v

info() Interesting events. VERBOSITY_QUIET (2) -vv

debug() Detailed debug information. VERBOSITY_DEBUG (3) -vvv

```
use Psr\Log\LoggerInterface;
```

```
public function index(LoggerInterface $logger) {
    $logger->info('I just got the logger');
}
```

Console

bin/console List available commands and show the Symfony version

server:run Run the built-in web server

assets:install --symlink Install bundle assets as a symlink or hardcopy

debug:autowire Lists classes/interfaces you can use for autowiring

debug:config Dumps the current configuration for an extension

debug:container Displays current services for an application

debug:form Lists classes/interfaces you can use for autowiring

debug:route Displays current routes for an application

Usage:

php bin/console command [options] [arguments]

Options:

-h, --help Display this help message

-q, --quiet Do not output any message

-n, --no-interaction Do not ask any interactive question

-e, --env=ENV The environment name [default: "dev"]

--no-debug Switches off debug mode

-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Basic form

```
class TaskForm extends AbstractType
{
    public function buildForm(
        FormBuilderInterface $builder,
        array $options)
    {
        $builder
            ->add('dueDate', DateType::class, array(
                'widget' => 'single_text',
            ));
    }
}
```



Basic form (cont)

```
'label' => 'Due Date',
'required' => false,
'attr' => array('maxlength' => 10),
'constraints' => array(new Length(
array('max' => 10)))
))
->add('save', SubmitType::class);
}
public function configureOptions(
OptionsResolver $resolver)
{
$resolver->setDefaults(array(
'method' => 'GET',
));
}
}
```

TwigBridge - Forms

<code>{{ form(view, variables) }}</code>	Render whole form
<code>{{ form_start(view, variables) }}</code>	Render start tag
<code>{{ form_errors(view) }}</code>	Render global errors
<code>{{ form_row(view, variables) }}</code>	Render all the fields
<code>{{ form_rest(view, variables) }}</code>	Render all other fields
<code>{{ form_end(view, variables) }}</code>	Render end tag + all other fields
<code>{{ form_row(view.field) }}</code>	Render field label, error and widget
<code>{{ form_label(view.field, label, variables) }}</code>	Render field label

TwigBridge - Forms (cont)

<code>{{ form_errors(view.field) }}</code>	Render field error
<code>{{ form_widget(view.field, variables) }}</code>	Render field widget
<pre>{# render a label, but display 'Your Name' and add a "foo" class to it #} {{ form_label(form.name, 'Your Name', {'label_attr': {'class': 'foo'}}) }}</pre>	
<pre>{# render a widget, but add a "foo" class to it #} {{ form_widget(form.name, {'attr': {'class': 'foo'}}) }}</pre>	
<pre>{# render a field row, but display a label with text "foo" #} {{ form_row(form.name, {'label': 'foo'}) }}</pre>	

Response from a controller

<code>render(\$view, \$parameters, \$response = null)</code>	Render the template and return a Response
<code>json(\$data, \$status = 200, \$headers = array(), \$context = array())</code>	Encode data and return a Json response
<code>file(\$file, \$fileName = null, \$disposition = ResponseHeaderBag::DISPOSITION_ATTACHMENT)</code>	Return a file response
<code>redirectToRoute(\$route, \$parameters = array(), \$status = 302)</code>	Redirect to route
<code>redirect(\$url, \$status = 302)</code>	Redirect to external URL
<code>forward(\$controller, \$path = array(), \$query = array())</code>	Forwards the request to another controller

```
return $this->render('admin/post/show.html.twig', [
    'post' => $post,
    'delete_form' => $deleteForm->createView(),
]);

return $this->json($data);
```

