Getting started with Doctrine2

Doctrine2 Installation

Define the following requirement in your `composer.json` file:

```
{ "require": { "doctrine/orm": "*" } }
```

Then call `composer install` from your command line. For more details consult Doctrine2 documentation or Composer documentation.

Doctrine2 configuration

Class loading

Autoloading is taken care of by Composer. You just have to include the composer autoload file in your project:

```
<?php // bootstrap.php // Include Composer Autoload (relative to project root). require_once
"vendor/autoload.php";
```

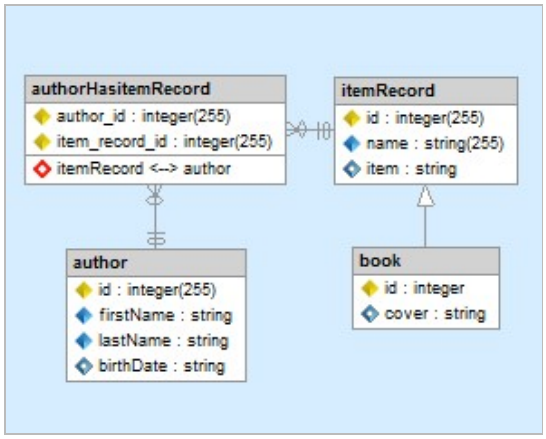
For more details check the documentation.

Entity Manager

Once you have prepared the class loading, you acquire an EntityManager instance. The EntityManager class is the primary access point to ORM functionality provided by Doctrine.

Annotations | XML | YML | More info | Links

```
<?php // bootstrap.php require_once "vendor/autoload.php"; use Doctrine\ORM\Tools\Setup; use
Doctrine\ORM\EntityManager; $paths = array("/path/to/entity-files"); $isDevMode = false; // the connection
configuration $dbParams = array( 'driver' => 'pdo_mysql', 'user' => 'root', 'password' => '', 'dbname' =>
'foo', ); $config = Setup::createAnnotationMetadataConfiguration($paths, $isDevMode); $entityManager =
EntityManager::create($dbParams, $config);
```



Generated by Skipper

Command line tool

You need to register your applications EntityManager to the console tool to make use of the tasks by creating a `cli-config.php` file with the following content:

version 2.4 and newer | version 2.3 and older

```
<?php use Doctrine\ORM\Tools\Console\ConsoleRunner; // replace with file to your own project bootstrap
require_once 'bootstrap.php'; // replace with mechanism to retrieve EntityManager in your app
$entityManager = GetEntityManager(); return ConsoleRunner::createHelperSet($entityManager);
```

For more details check the documentation.

Doctrine2 basic use

Generating Doctrine2 model from database

```
$ php doctrine orm:convert-mapping --from-database yml /path/to/mapping-path-converted-to-yml
```

For more details check the documentation.

Creating database tables from Doctrine2 model

Generate the database schema:

```
$ php doctrine orm:schema-tool:create
```

Update the database schema:

```
$ php doctrine orm:schema-tool:update
```

For more details [check the documentation](#).

Using Doctrine2 with Symfony2

Installation

Add required bundles to `composer.json`:

```
"require": { .... .... "doctrine/orm": "*", "doctrine/doctrine-bundle": "*", },
```

And enable the bundles in the Kernel:

```
class AppKernel extends Kernel { public function registerBundles() { $bundles = array( ..... new
Doctrine\Bundle\DoctrineBundle\DoctrineBundle(), ); ..... return $bundles; } }
```

Configuration

Configure `app/config/config.yml` and add Doctrine section:

```
doctrine: dbal: driver: %database_driver% host: %database_host% port: %database_port% dbname:
%database_name% user: %database_user% password: %database_password% charset: UTF8 orm:
auto_generate_proxy_classes: %kernel.debug% auto_mapping: true
```

Configure `app/config/parameters.yml` and set database connection parameters:

```
parameters: database_driver: pdo_mysql database_host: 127.0.0.1 database_port: ~ database_name: symfony
database_user: UsernameHere database_password: PasswordHere
```

Configure your schema

You can use annotations for schema definition. Entities can be stored at `src\Acme\SampleBundle\Entity\User.php`:

```
<?php namespace Acme\SampleBundle\Entity; use Doctrine\ORM\Mapping AS ORM; /** * @ORM\Entity *
@ORM\Table(name="acme_user") */ class Event { /** * @ORM\Id * @ORM\Column(type="integer") *
@ORM\GeneratedValue(strategy="AUTO") */ private $id; /** * @ORM\Column(type="string", unique=true,
length=64, nullable=false) */ private $name; }
```

Getters and setters can be generated simply by running:

```
php app/console doctrine:generate:entities Acme
```

Using Doctrine2 with Symfony1.4

Installation

First we need to install the plugin from SVN with the following command from the root of your project:

```
$ svn co http://svn.symfony-project.org/plugins/sfDoctrinePlugin/branches/1.3-2.0/
plugins/sfDoctrine2Plugin
```

Now you just need to enable the plugin:

```
class ProjectConfiguration extends sfProjectConfiguration { public function setup() { $this-
>enablePlugins('sfDoctrine2Plugin'); } }
```

Configuration

Configure `config/databases.yml` for database connection:

```
all: doctrine: class: sfDoctrineDatabase param: options: driver: pdo_mysql user: root password: dbname:
doctrine
```

Configure Your Schema

Below is an example of a simple User entity:

```
# config/doctrine/schema.yml Models\User: type: entity table: user id: id: type: integer generator:
strategy: AUTO fields: username: type: string length: 255 password: type: string length: 255
```

Basic use

Writing Data Fixtures

The times of using YAML for data fixtures is no longer. Instead, you are only required to use plain PHP for loading your data fixtures.

```
// data/fixtures/fixtures.php $em = $this->getEntityManager(); $admin = new \Models\User(); $admin-
>username = 'admin'; $admin->password = 'changeme';
```

Building Doctrine

Now you're ready to build everything. The following command will build models, forms, filters, database and load data fixtures.

```
$ php symfony doctrine:build --all --and-load
```

Updating Schema

If you change your schema mapping information and want to update the database you can easily do so by running the following command after changing your mapping information.

```
$ php symfony doctrine:build --all-classes --and-update-schema
```

Using Doctrine2 with Zend Framework 2

Installation

```
php composer.phar require doctrine/doctrine-orm-module:0.7.*
```

```
php composer.phar require zendframework/zend-developer-tools:dev-master
```

```
cp vendor/zendframework/zend-developer-tools/config/zenddevelopertools.local.php.dist
config/autoload/zdt.local.php
```

Enable the modules in `config/application.config.php`:

```
return array( 'modules' => array( 'ZendDeveloperTools', 'DoctrineModule', 'DoctrineORMModule',
'Application', ), // [...] );
```

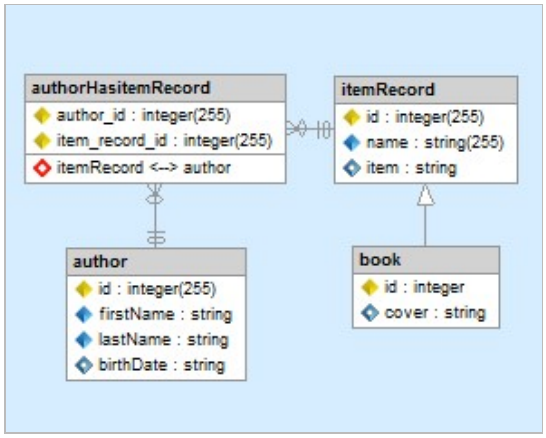
Doctrine2 Model Elements

Schema file structure

Doctrine2 uses one `*.dcm.xml` schema file for each entity. The file is structured like this:

Annotations | XML | YML | Links

```
<?php use Doctrine\ORM\Mapping AS ORM; /** * @ORM\Entity(repositoryClass="Doctrine\ORM\EntityRepository") *
@ORM\Table */ class itemRecord { /** * @ORM\Id */ private $id; /** * @ORM\Column */ private $name;
```



Generated by Skipper

You can go and see how to do this in few clicks.

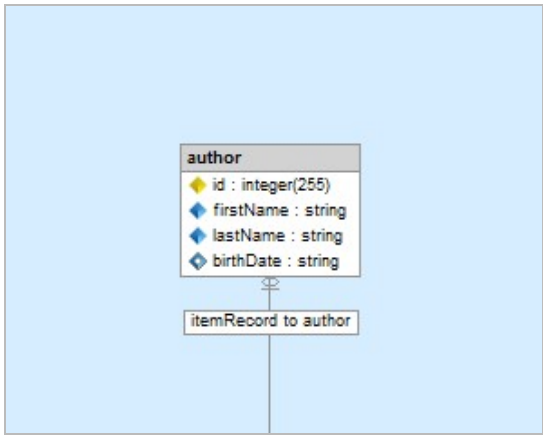
Entity

Simple entity

Simple entity with a primary key and several fields:

Annotations | XML | YML | Links

```
<?php use Doctrine\ORM\Mapping AS ORM; /** * @ORM\Entity */ class author { /** * @ORM\Id *
@ORM\Column(type="integer") * @ORM\GeneratedValue(strategy="AUTO") */ private $id; /** *
@ORM\Column(type="string", nullable=false) */ private $firstName; /** * @ORM\Column(type="string",
nullable=false) */ private $lastName; /** * @ORM\Column(type="string", nullable=true) */ private
$birthDate;
```

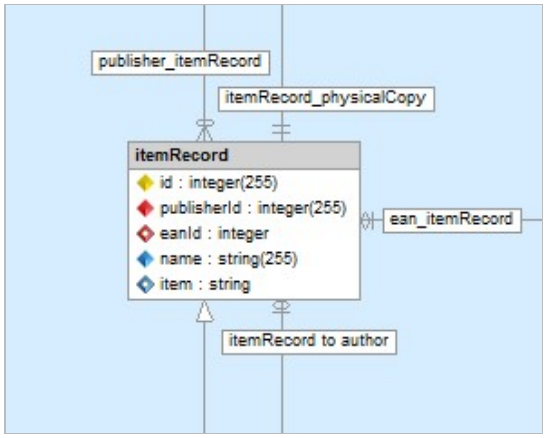


Entity with all options defined

Entity with all options defined:

Annotations | XML | YML | Links

```
<?php use Doctrine\ORM\Mapping AS ORM; /** * * @ORM\Table( * schema="item_record", * name="item_record", *
options={ * "charset":"utf8", * "collate":"utf8_unicode_ci", * "comment":"library record of a work", *
"temporary":false, * "engine":"InnoDB" * }, * indexes={ * @ORM\Index(name="ix_name", columns=
{"itemRecord_name"}), * @ORM\Index(name="ix_name_publisher", columns=
{"itemRecord_publisherId","itemRecord_name"}) * }, * uniqueConstraints={ *
@ORM\UniqueConstraint(name="ix_name_ean", columns={"itemRecord_name","eanId"}), *
@ORM\UniqueConstraint(name="ix_ean_publisher", columns={"itemRecord_publisherId","eanId"}) * } * ) *
@ORM\DiscriminatorMap( *
{"itemRecord"="itemRecord","book"="book","magazine"="magazine","audioRecord"="audioRecord"} * ) *
@ORM\DiscriminatorColumn(name="item", type="string") * @ORM\InheritanceType("JOINED") * * * *
@ORM\HasLifecycleCallbacks * @ORM\ChangeTrackingPolicy("DEFERRED_IMPLICIT") *
@ORM\Entity(repositoryClass="Doctrine\ORM\EntityRepository") */ class itemRecord { /** * @ORM\Id *
@ORM\Column() */ private $id; /** * @ORM\Column() */ private $name; /** * @ORM\OneToOne() */ private $ean;
/** * @ORM\OneToMany() */ private $physicalCopy; /** * @ORM\ManyToOne() * @ORM\JoinColumn() */ private
$publisher; /** * @ORM\ManyToMany() */ private $author; /** * @ORM\PostPersist */ public function
sendOptinMail() { } }
```



Generated by Skipper

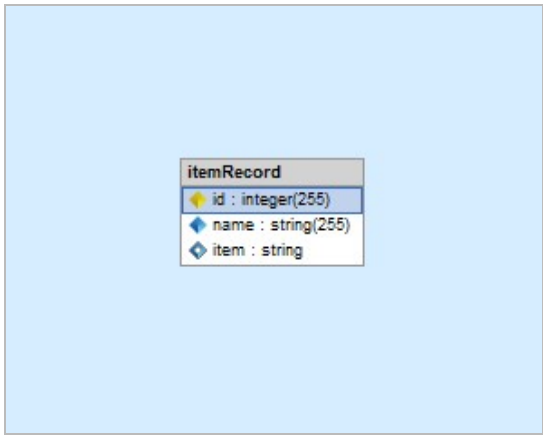
You can go and see how to do this in few clicks.

Id

Primary key definition:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class itemRecord { /** * @ORM\Id * @ORM\Column( * type="integer", *
name="itemRecord_id" * length=255 * ) * @ORM\GeneratedValue(strategy="AUTO") */ private $id;
```

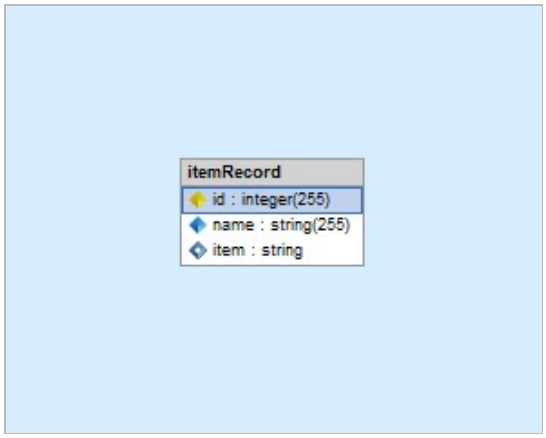


Generated by Skipper

Primary key with all base properties set:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class itemRecord { /** * @ORM\Id * @ORM\Column( * type="integer", *
name="itemRecord_id", * length=255 * columnDefinition="itemRecord_id", * precision=3, * scale=3, * options=
{"unsigned":true,"comment":"this is primary key","version":2} * ) * @ORM\Version *
@ORM\GeneratedValue(strategy="SEQUENCE") */ private $id;
```



Generated by Skipper

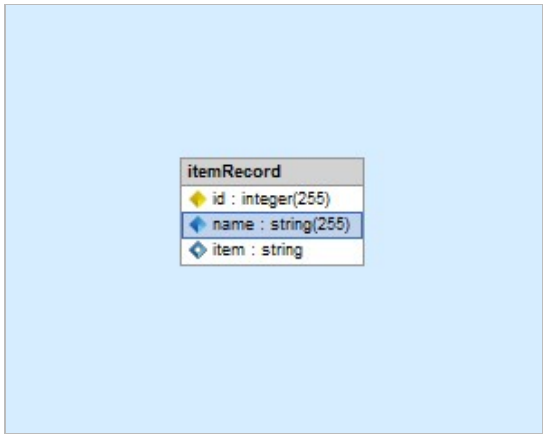
You can go and see how to do this in few clicks.

Field

Regular field definition:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class itemRecord { /** * @ORM\Column( * type="string", * unique=true, * length=255 * ) */ private $name;
```

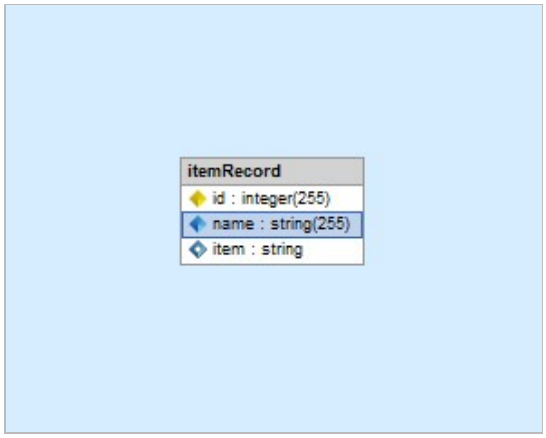


Generated by Skipper

Regular field with all options set:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class itemRecord { /** * @ORM\Column( * type="string", * length=255, * nullable=false, * name="itemRecord_name", * columnDefinition="itemRecord_name", * precision=1, * scale=1, * options= {"comment":"this is field","unsigned":true,"version":3} * ) */ private $item; }
```



Generated by Skipper

You can go and see how to do this in few clicks.

Index

Indexes

Non-unique index:

Annotations | XML | YML | Links

```
/** * @ORM\Entity * @ORM\Table( * indexes={@ORM\Index(name="ix_name_last", columns={"lastName"})} * ) */ class author
```

Generated by Skipper

Unique index definition:

Annotations | XML | YML | Links

```
/** * @ORM\Entity * @ORM\Table( * uniqueConstraints=
{@ORM\UniqueConstraint(name="ix_first_name_last_name_date", columns={"firstName","lastName","birthDate"})}
* ) */ class author
```

Generated by Skipper

You can go and see how to do this in few clicks.

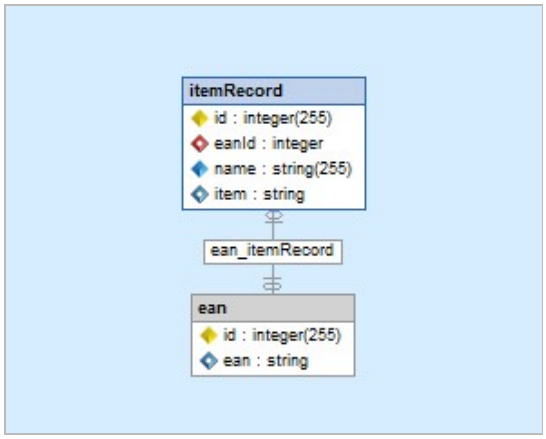
Association

One to One

One to one owner side:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class itemRecord { /** * @ORM\OneToOne( * targetEntity="ean", *
inversedBy="itemRecord" * ) * @ORM\JoinColumn(name="eanId", referencedColumnName="id", unique=true) */
private $ean; }
```

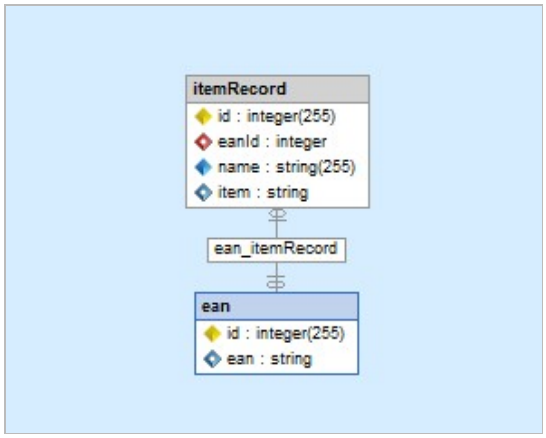


Generated by Skipper

One to one inverse side:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class ean { /** * @ORM\Id * @ORM\Column(type="integer") *
@ORM\GeneratedValue(strategy="AUTO") */ private $id; /** * @ORM\OneToOne( * targetEntity="itemRecord", *
mappedBy="ean" * ) */ private $itemRecord; }
```



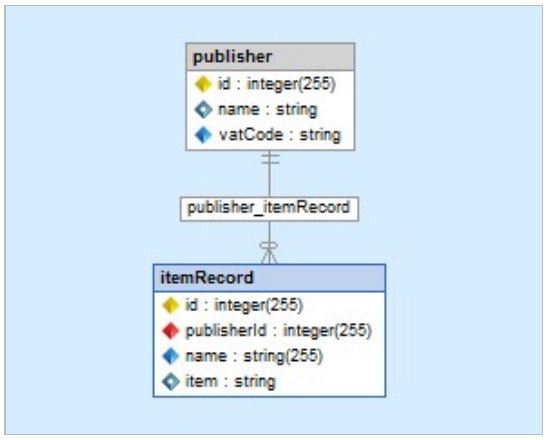
Generated by Skipper

Many to One

Many to one owner side:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class itemRecord { /** * @ORM\ManyToOne( * targetEntity="publisher", * inversedBy="itemRecord" * ) * @ORM\JoinColumn( * name="publisherId", * referencedColumnName="id", * nullable=false * ) */ private $publisher; }
```

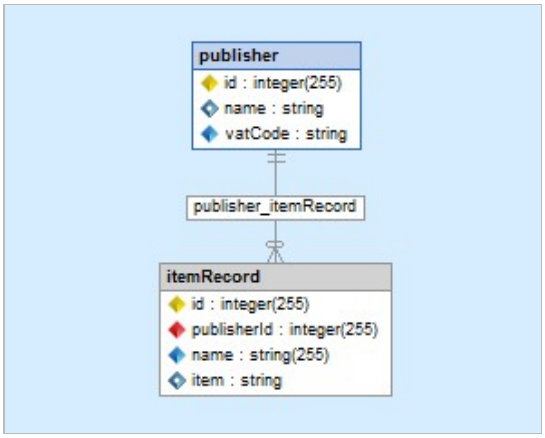


Generated by Skipper

Many to one inverse side:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class publisher { /** * @ORM\Id */ private $id; /** * @ORM\OneToMany( * targetEntity="itemRecord", * mappedBy="publisher" * ) */ private $itemRecord; }
```



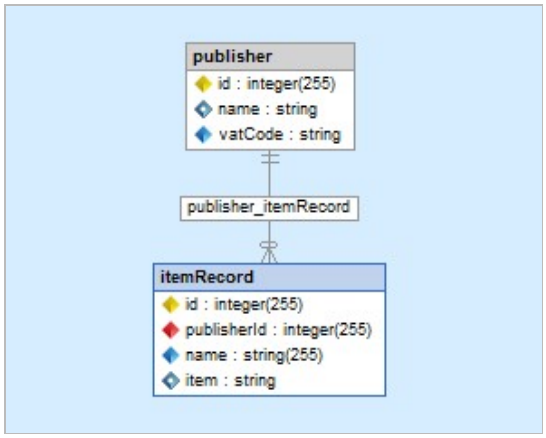
Generated by Skipper

Association with all options enabled

Many to one owner side with all properties:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class itemRecord { /** * @ORM\ManyToOne( * targetEntity="publisher", *
inversedBy="itemRecord", * fetch="EXTRA_LAZY", * orphanRemoval=true, * cascade=
{"all","merge","persist","refresh","remove"} * ) * @ORM\JoinColumn( * name="publisherId", *
referencedColumnName="id", * nullable=false, * columnDefinition="itemRecord_publisherId", *
onDelete="CASCADE", * onUpdate="RESTRICT" * ) */ private $publisher; }
```

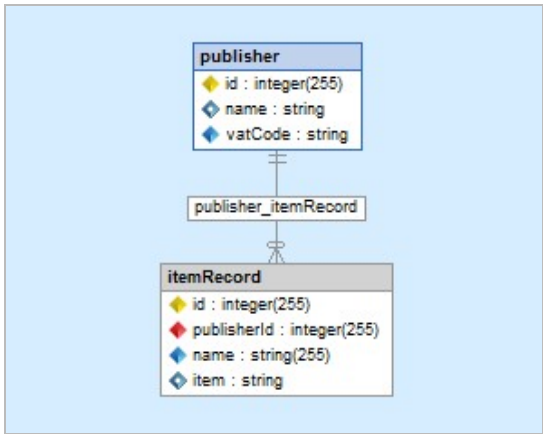


Generated by Skipper

Many to one inverse side with all properties:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class publisher { /** * @ORM\Id * @ORM\Column( * type="integer" * ) *
@ORM\GeneratedValue(strategy="SEQUENCE") */ private $id; /** * @ORM\OneToMany( * targetEntity="itemRecord",
* mappedBy="publisher", * fetch="EAGER", * indexBy="id", * cascade=
{"all","merge","persist","refresh","remove"} * ) * @ORM\OrderBy({"id"="ASC"}) */ private $itemRecord; }
```



Generated by Skipper

You can go and see how to do this in few clicks.

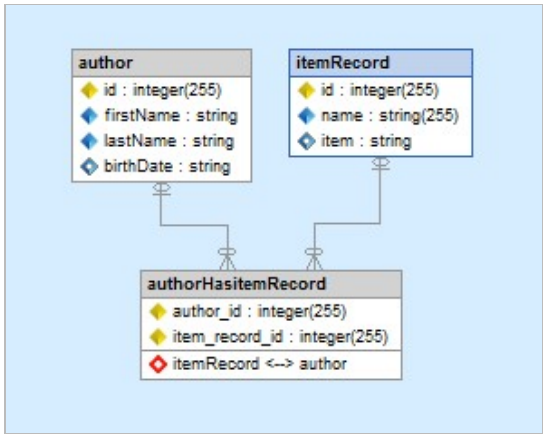
MN Association

Many to Many

Many to many owner side:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class author { /** * @ORM\Id */ private $id; /** *
@ORM\ManyToMany(targetEntity="itemRecord", inversedBy="author") * @ORM\JoinTable( *
name="itemRecordHasAuthor", * joinColumns={ * @ORM\JoinColumn( * name="authorId", *
referencedColumnName="id", * nullable=false * ) * }, * inverseJoinColumns={@ORM\JoinColumn(name="bookId",
referencedColumnName="itemRecord_id", nullable=false)} * ) */ private $itemRecord; }
```

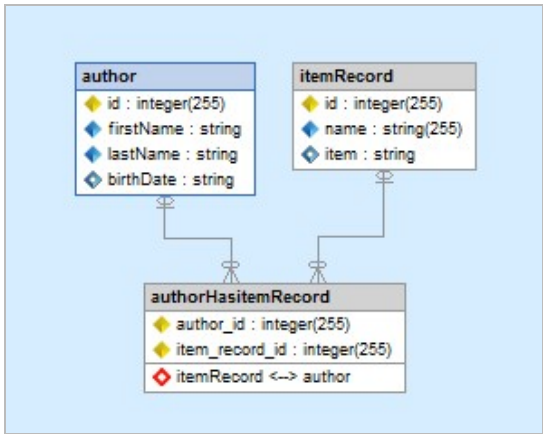


Generated by Skipper

Many to many inverse side:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class itemRecord { /** * @ORM\ManyToMany(targetEntity="author", mappedBy="itemRecord")
*/ private $author; }
```



Generated by Skipper

Many to many with all options enabled:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class author { /** * @ORM\Id */ private $id; /** *
@ORM\ManyToMany(targetEntity="itemRecord", inversedBy="author", cascade={"all","refresh"}) *
@ORM\JoinTable( * name="itemRecordHasAuthor", * joinColumns={ * @ORM\JoinColumn( * name="authorId", *
referencedColumnName="id", * nullable=false, * fetch="EAGER", * onDelete="CASCADE", * onUpdate="RESTRICT" *
) * }, * inverseJoinColumns={@ORM\JoinColumn(name="bookId", referencedColumnName="itemRecord_id",
nullable=false)} * ) * @ORM\OrderBy({"id"="ASC"}) */ private $itemRecord; }
```

MN Entity

Does not exist as a Doctrine2 object, it is handled internally.

You can go and see how to do this in few clicks.

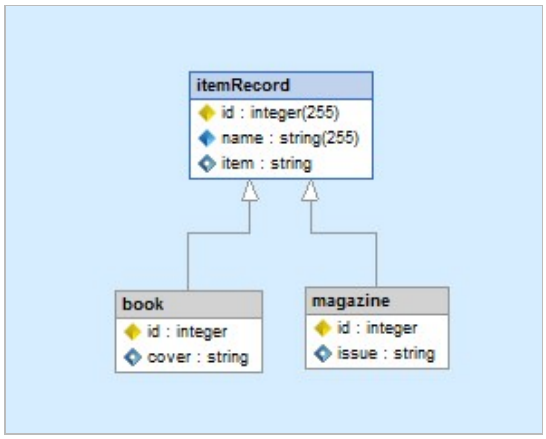
Inheritance

Single table inheritance

Single table inheritance parent:

Annotations | XML | YML | Links

```
/** * @ORM\Entity * @ORM\InheritanceType("SINGLE_TABLE") * @ORM\DiscriminatorColumn(name="item",
type="string") * @ORM\DiscriminatorMap( *
{"itemRecord"="itemRecord","book"="book","magazine"="magazine","audioRecord"="audioRecord"} * ) */ class
itemRecord
```

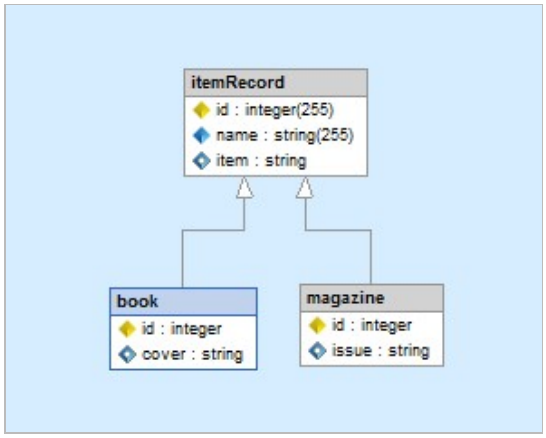


Generated by Skipper

Single table inheritance child:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class book extends itemRecord
```



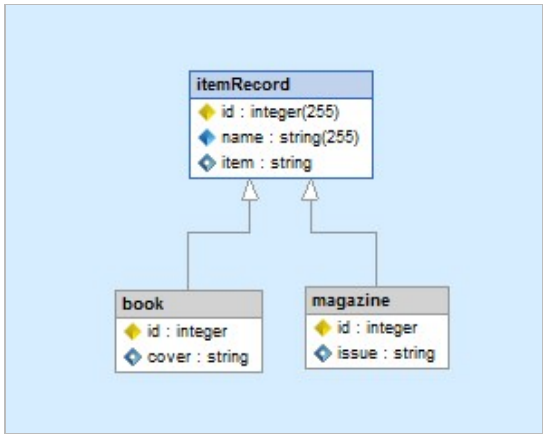
Generated by Skipper

Class table inheritance

Class table inheritance parent:

Annotations | XML | YML | Links

```
/** * @ORM\Entity * @ORM\InheritanceType("JOINED") * @ORM\DiscriminatorColumn(name="item", type="string") *
@ORM\DiscriminatorMap( *
{"itemRecord"="itemRecord","book"="book","magazine"="magazine","audioRecord"="audioRecord"} * ) */ class
itemRecord
```

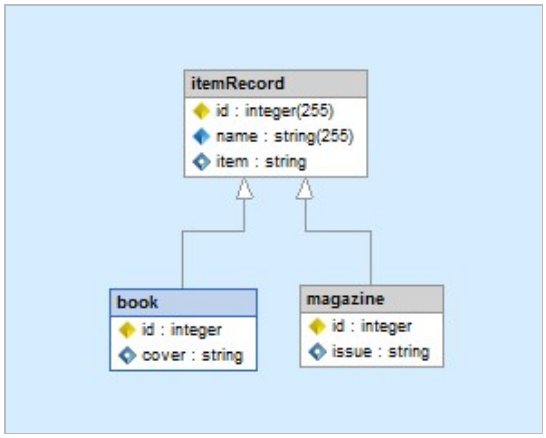


Generated by Skipper

Class table inheritance child:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class book extends itemRecord
```



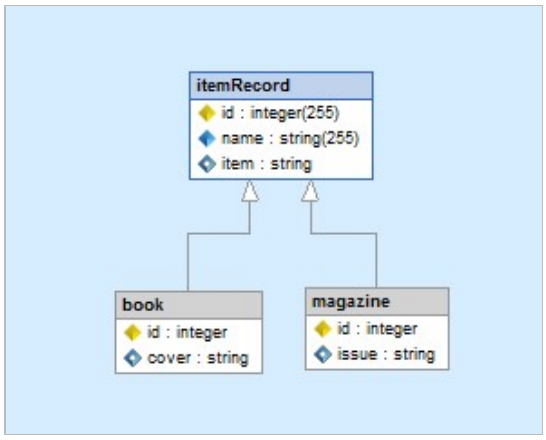
Generated by Skipper

Mapped superclass inheritance

Mapped superclass inheritance parent:

Annotations | XML | YML | Links

```
/** * * @ORM\MappedSuperclass(repositoryClass="Doctrine\ORM\EntityRepository") */ class itemRecord
```

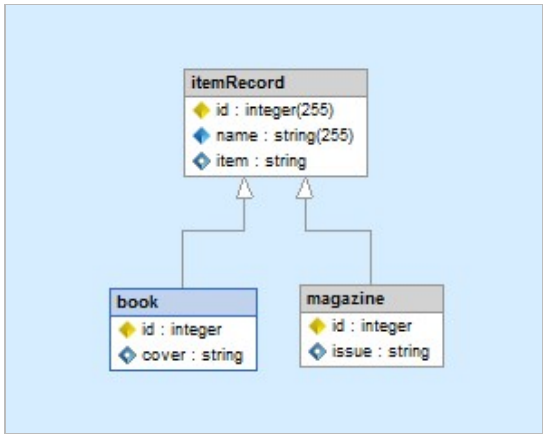


Generated by Skipper

Mapped supperclass inheritance child:

Annotations | XML | YML | Links

```
/** * @ORM\Entity */ class book extends itemRecord
```



Generated by Skipper

You can go and see how to do this in few clicks.

Doctrine Extensions

Installing Doctrine extensions for Symfony2

Install Symfony-standard edition with composer:

- `git clone git://github.com/KnpLabs/symfony-with-composer.git example`
- `cd example && rm -rf .git && php bin/vendors install`
- ensure your application loads and meets requirements

Add the **gedmo/doctrine-extensions** into **composer.json**

```
{ "require": { "php": ">=5.3.2", "symfony/symfony": ">=2.0.9,<2.1.0-dev", "doctrine/orm": ">=2.1.0,<2.2.0-dev", "twig/extensions": "*", "symfony/assetic-bundle": "*", "sensio/generator-bundle": "2.0.*", "sensio/framework-extra-bundle": "2.0.*", "sensio/distribution-bundle": "2.0.*", "jms/security-extra-bundle": "1.0.*", "gedmo/doctrine-extensions": "dev-master" }, "autoload": { "psr-0": { "Acme": "src/" } } }
```

Update vendors: `php composer.phar update gedmo/doctrine-extensions` Configure your database connection parameters: `app/config/parameters.ini`

Mapping Doctrine2 for Symfony2

If you use **translatable**, **tree** or **loggable** extension you will need to map those abstract mappedsuperclasses. Add mapping info to your `doctrine.orm` configuration, edit `app/config/config.yml` :

```
doctrine: dbal: # your dbal config here orm: auto_generate_proxy_classes: %kernel.debug% auto_mapping:
true # only these lines are added additionally mappings: translatable: type: annotation alias: Gedmo
prefix: Gedmo\Translatable\Entity # make sure vendor library location is correct dir:
"%kernel.root_dir%/../vendor/gedmo/doctrine-extensions/lib/Gedmo/Translatable/Entity"
```

After that, running **php app/console doctrine:mapping:info** you should see the output:

```
Found 3 entities mapped in entity manager default: [OK]
Gedmo\Translatable\Entity\MappedSuperclass\AbstractPersonalTranslation [OK]
Gedmo\Translatable\Entity\MappedSuperclass\AbstractTranslation [OK] Gedmo\Translatable\Entity\Translation
```

Note: there is **Gedmo\Translatable\Entity\Translation** which is not a super class, in that case if you create doctrine schema, it will add **ext_translations** table, which might not be useful to you also. To skip mapping of these entities, you can map **only superclasses**

```
mappings: translatable: type: annotation alias: Gedmo prefix: Gedmo\Translatable\Entity # make sure vendor
library location is correct dir: "%kernel.root_dir%/../vendor/gedmo/doctrine-
extensions/lib/Gedmo/Translatable/Entity/MappedSuperclass"
```

The configuration above, adds a **/MappedSuperclass** into directory depth, after running **php app/console doctrine:mapping:info** you should only see now:

```
Found 2 entities mapped in entity manager default: [OK]
Gedmo\Translatable\Entity\MappedSuperclass\AbstractPersonalTranslation [OK]
Gedmo\Translatable\Entity\MappedSuperclass\AbstractTranslation
```

To map every extension use:

```
# only orm config branch of doctrine orm: auto_generate_proxy_classes: %kernel.debug% auto_mapping: true #
only these lines are added additionally mappings: translatable: type: annotation alias: Gedmo prefix:
Gedmo\Translatable\Entity # make sure vendor library location is correct dir:
"%kernel.root_dir%/../vendor/gedmo/doctrine-extensions/lib/Gedmo/Translatable/Entity" loggable: type:
annotation alias: Gedmo prefix: Gedmo\Loggable\Entity dir: "%kernel.root_dir%/../vendor/gedmo/doctrine-
extensions/lib/Gedmo/Loggable/Entity" tree: type: annotation alias: Gedmo prefix: Gedmo\Tree\Entity dir:
"%kernel.root_dir%/../vendor/gedmo/doctrine-extensions/lib/Gedmo/Tree/Entity"
```

Listener services for Doctrine2

Edit and create an *yml* service file in your `app/config/doctrine_extensions.yml`

```
# services to handle doctrine extensions # import it in config.yml services: # KernelRequest listener
extension.listener: class: Acme\DemoBundle\Listener\DoctrineExtensionListener calls: - [ setContainer, [
@service_container ] ] tags: # translatable sets locale after router processing - { name:
kernel.event_listener, event: kernel.request, method: onLateKernelRequest, priority: -10 } # loggable
hooks user username if one is in security context - { name: kernel.event_listener, event: kernel.request,
method: onKernelRequest } # Doctrine Extension listeners to handle behaviors gedmo.listener.tree: class:
```

```
Gedmo\Tree\TreeListener tags: - { name: doctrine.event_subscriber, connection: default } calls: - [
setAnnotationReader, [ @annotation_reader ] ] gedmo.listener.translatable: class:
Gedmo\Translatable\TranslatableListener tags: - { name: doctrine.event_subscriber, connection: default }
calls: - [ setAnnotationReader, [ @annotation_reader ] ] - [ setDefaultLocale, [ %locale% ] ] - [
setTranslationFallback, [ false ] ] gedmo.listener.timestampable: class:
Gedmo\Timestampable\TimestampableListener tags: - { name: doctrine.event_subscriber, connection: default }
calls: - [ setAnnotationReader, [ @annotation_reader ] ] gedmo.listener.sluggable: class:
Gedmo\Sluggable\SluggableListener tags: - { name: doctrine.event_subscriber, connection: default } calls:
- [ setAnnotationReader, [ @annotation_reader ] ] gedmo.listener.sortable: class:
Gedmo\Sortable\SortableListener tags: - { name: doctrine.event_subscriber, connection: default } calls: -
[ setAnnotationReader, [ @annotation_reader ] ] gedmo.listener.loggable: class:
Gedmo\Loggable\LoggableListener tags: - { name: doctrine.event_subscriber, connection: default } calls: -
[ setAnnotationReader, [ @annotation_reader ] ]
```

Note: You will need to create `Acme\DemoBundle\Listener\DoctrineExtensionListener` if you use **loggable** or **translatable** behaviors. This listener will set the `locale used` from request and `username` to loggable.

```
<?php // file: src/Acme/DemoBundle/Listener/DoctrineExtensionListener.php namespace
Acme\DemoBundle\Listener; use Symfony\Component\HttpKernel\Event\GetResponseEvent; use
Symfony\Component\DependencyInjection\ContainerAwareInterface; use
Symfony\Component\DependencyInjection\ContainerInterface; class DoctrineExtensionListener implements
ContainerAwareInterface { /** * @var ContainerInterface */ protected $container; public function
setContainer(ContainerInterface $container = null) { $this->container = $container; } public function
onLateKernelRequest(GetResponseEvent $event) { $translatable = $this->container-
>get('gedmo.listener.translatable'); $translatable->setTranslatableLocale($event->getRequest()-
>getLocale()); } public function onKernelRequest(GetResponseEvent $event) { $securityContext = $this-
>container->get('security.context', ContainerInterface::NULL_ON_INVALID_REFERENCE); if (null !==
$securityContext && null !== $securityContext->getToken() && $securityContext-
>isGranted('IS_AUTHENTICATED_REMEMBERED')) { $loggable = $this->container->get('gedmo.listener.loggable');
$loggable->setUsername($securityContext->getToken()->getUsername()); } } }
```

Do not forget to import **doctrine_extensions.yml** in your **app/config/config.yml** etc.:

```
# file: app/config/config.yml imports: - { resource: parameters.yml } - { resource: security.yml } - {
resource: doctrine_extensions.yml } # ... configuration follows
```

Installing Doctrine extensions for Zend Framework 2

Add DoctrineModule, DoctrineORMModule and DoctrineExtensions to composer.json file:

```
{ "require": { "php": ">=5.3.3", "zendframework/zendframework": "2.1.*", "doctrine/doctrine-module":
"0.*", "doctrine/doctrine-orm-module": "0.*", "gedmo/doctrine-extensions": "2.3.*", } }
```

Then run `composer.phar update`.

Configuring Doctrine extensions for Zend Framework 2

Declaring appropriate subscribers in Event Manager settings. With entity mapping options module configuration file should look like this:

```
return array( 'doctrine' => array( 'eventmanager' => array( 'orm_default' => array( 'subscribers' =>
array( // pick any listeners you need 'Gedmo\Tree\TreeListener',
'Gedmo\Timestampable\TimestampableListener', 'Gedmo\Sluggable\SluggableListener',
'Gedmo\Loggable\LoggableListener', 'Gedmo\Sortable\SortableListener' ), ), ), 'driver' => array(
'my_driver' => array( 'class' => 'Doctrine\ORM\Mapping\Driver\AnnotationDriver', 'cache' => 'array',
'paths' => array(__DIR__ . '/../src/MyModule/Entity') ), 'orm_default' => array( 'drivers' => array(
'MyModule\Entity' => 'my_driver' ), ), ), ), );
```

Behaviors

Tree

Tree nested behavior will implement the standard Nested-Set behavior on your Entity.

Annotations | XML | YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; /** *
@Gedmo\Tree(type="nested") * @ORM\Table(name="categories") * use repository for handy tree functions *
@ORM\Entity(repositoryClass="Gedmo\Tree\Entity\Repository\NestedTreeRepository") */ class Category { /** *
@ORM\Column(name="id", type="integer") * @ORM\Id * @ORM\GeneratedValue */ private $id; /** *
@ORM\Column(name="title", type="string", length=64) */ private $title; /** * @Gedmo\TreeLeft *
@ORM\Column(name="lft", type="integer") */ private $lft; /** * @Gedmo\TreeLevel * @ORM\Column(name="lvl",
type="integer") */ private $lvl; /** * @Gedmo\TreeRight * @ORM\Column(name="rgt", type="integer") */
private $rgt; /** * @Gedmo\TreeRoot * @ORM\Column(name="root", type="integer", nullable=true) */ private
$root; /** * @Gedmo\TreeParent * @ORM\ManyToOne(targetEntity="Category", inversedBy="children") *
@ORM\JoinColumn(name="parent_id", referencedColumnName="id", onDelete="CASCADE") */ private $parent; /** *
@ORM\OneToMany(targetEntity="Category", mappedBy="parent") * @ORM\OrderBy({"lft" = "ASC"}) */ private
$children; public function getId() { return $this->id; } public function setTitle($title) { $this->title =
$title; } public function getTitle() { return $this->title; } public function setParent(Category $parent =
null) { $this->parent = $parent; } public function getParent() { return $this->parent; } }
```

For more details [check the documentation](#).

Translatable

Translatable behavior offers a very handy solution for translating specific record fields in different languages.

Annotations | XML | YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; use
Gedmo\Translatable\Translatable; /** * @ORM\Table(name="articles") * @ORM\Entity */ class Article
implements Translatable { /** @ORM\Id @ORM\GeneratedValue @ORM\Column(type="integer") */ private $id; /** *
@Gedmo\Translatable * @ORM\Column(name="title", type="string", length=128) */ private $title; /** *
@Gedmo\Translatable * @ORM\Column(name="content", type="text") */ private $content; /** * @Gedmo\Locale *
Used locale to override Translation listener`s locale * this is not a mapped field of entity metadata, just
a simple property */ private $locale; public function getId() { return $this->id; } public function
setTitle($title) { $this->title = $title; } public function getTitle() { return $this->title; } public
function setContent($content) { $this->content = $content; } public function getContent() { return $this-
>content; } public function setTranslatableLocale($locale) { $this->locale = $locale; } }
```

For more details check the documentation.

Sluggable

Sluggable behavior will build the slug of predefined fields on a given field which should store the slug.

Annotations | XML | YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; /** *
@ORM\Table(name="articles") * @ORM\Entity */ class Article { /** * @ORM\Id * @ORM\GeneratedValue *
@ORM\Column(type="integer") */ private $id; /** * @ORM\Column(length=64) */ private $title; /** *
@ORM\Column(length=16) */ private $code; /** * @Gedmo\Slug(fields={"title", "code"}) *
@ORM\Column(length=128, unique=true) */ private $slug; public function getId() { return $this->id; } public
function setTitle($title) { $this->title = $title; } public function getTitle() { return $this->title; }
public function setCode($code) { $this->code = $code; } public function getCode() { return $this->code; }
public function getSlug() { return $this->slug; } }
```

For more details check the documentation.

Timestampable

Timestampable behavior will automate the update of date fields on your Entities or Documents.

Annotations | XML | YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; /** *
@ORM\Entity */ class Article { /** @ORM\Id @ORM\GeneratedValue @ORM\Column(type="integer") */ private $id;
/** * @ORM\Column(type="string", length=128) */ private $title; /** * @ORM\Column(name="body",
type="string") */ private $body; /** * @var datetime $created * * @Gedmo\Timestampable(on="create") *
@ORM\Column(type="datetime") */ private $created; /** * @var datetime $updated * *
@Gedmo\Timestampable(on="update") * @ORM\Column(type="datetime") */ private $updated; /** * @var datetime
$contentChanged * * @ORM\Column(name="content_changed", type="datetime", nullable=true) *
@Gedmo\Timestampable(on="change", field={"title", "body"}) */ private $contentChanged; public function
getId() { return $this->id; } public function setTitle($title) { $this->title = $title; } public function
getTitle() { return $this->title; } public function setBody($body) { $this->body = $body; } public function
getBody() { return $this->body; } public function getCreated() { return $this->created; } public function
getUpdated() { return $this->updated; } public function getContentChanged() { return $this->contentChanged;
} }
```

For more details check the documentation.

Blameable

Blameable behavior will automate the update of username or user reference fields on your Entities or Documents.

Annotations XML YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; /** *
@ORM\Entity */ class Article { /** @ORM\Id @ORM\GeneratedValue @ORM\Column(type="integer") */ private $id;
/** * @ORM\Column(type="string", length=128) */ private $title; /** * @ORM\Column(name="body",
type="string") */ private $body; /** * @var string $createdBy * * @Gedmo\Blameable(on="create") *
@ORM\Column(type="string") */ private $createdBy; /** * @var string $updatedBy * *
@Gedmo\Blameable(on="update") * @ORM\Column(type="string") */ private $updatedBy; /** * @var datetime
$contentChangedBy * * @ORM\Column(name="content_changed_by", type="string", nullable=true) *
@Gedmo\Timestampable(on="change", field={"title", "body"}) */ private $contentChangedBy; public function
getId() { return $this->id; } public function setTitle($title) { $this->title = $title; } public function
getTitle() { return $this->title; } public function setBody($body) { $this->body = $body; } public function
getBody() { return $this->body; } public function getCreatedBy() { return $this->createdBy; } public
function getUpdatedBy() { return $this->updatedBy; } public function getContentChangedBy() { return $this-
>contentChangedBy; } }
```

For more details check the documentation.

Loggable

Loggable behavior tracks your record changes and is able to manage versions.

Annotations XML YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; /** *
@Entity * @Gedmo\Loggable */ class Article { /** * @ORM\Column(name="id", type="integer") * @ORM\Id *
@ORM\GeneratedValue(strategy="IDENTITY") */ private $id; /** * @Gedmo\Versioned * @ORM\Column(name="title",
type="string", length=8) */ private $title; public function getId() { return $this->id; } public function
setTitle($title) { $this->title = $title; } public function getTitle() { return $this->title; } }
```

For more details check the documentation.

Sortable

Sortable behavior will maintain a position field for ordering entities.

Annotations XML YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; /** *
@ORM\Table(name="items") *
@ORM\Entity(repositoryClass="Gedmo\Sortable\Entity\Repository\SortableRepository") */ class Item { /**
@ORM\Id @ORM\GeneratedValue @ORM\Column(type="integer") */ private $id; /** * @ORM\Column(name="name",
type="string", length=64) */ private $name; /** * @Gedmo\SortablePosition * @ORM\Column(name="position",
type="integer") */ private $position; /** * @Gedmo\SortableGroup * @ORM\Column(name="category",
type="string", length=128) */ private $category; public function getId() { return $this->id; } public
function setName($name) { $this->name = $name; } public function getName() { return $this->name; } public
function setPosition($position) { $this->position = $position; } public function getPosition() { return
```

```
$this->position; } public function setCategory($category) { $this->category = $category; } public function getCategory() { return $this->category; } }
```

For more details check the documentation.

Softdeleteable

SoftDeleteable behavior allows to “soft delete” objects, filtering them at SELECT time by marking them as with a timestamp, but not explicitly removing them from the database.

Annotations | XML | YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; /** * @ORM\Entity * @Gedmo\SoftDeleteable(fieldName="deletedAt", timeAware=false) */ class Article { /** * @ORM\Column(name="id", type="integer") * @ORM\Id * @ORM\GeneratedValue(strategy="IDENTITY") */ private $id; /** * @ORM\Column(name="title", type="string") */ private $title; /** * @ORM\Column(name="deletedAt", type="datetime", nullable=true) */ private $deletedAt; public function getId() { return $this->id; } public function setTitle($title) { $this->title = $title; } public function getTitle() { return $this->title; } public function getDeletedAt() { return $this->deletedAt; } public function setDeletedAt($deletedAt) { $this->deletedAt = $deletedAt; } }
```

For more details check the documentation.

Uploadable

Uploadable behavior provides the tools to manage the persistence of files with Doctrine 2, including automatic handling of moving, renaming and removal of files and other features.

Annotations | XML | YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; /** * @ORM\Entity * @Gedmo\Uploadable(path="/my/path", callback="myCallbackMethod", filenameGenerator="SHA1", allowOverwrite=true, appendNumber=true) */ class File { /** * @ORM\Column(name="id", type="integer") * @ORM\Id * @ORM\GeneratedValue(strategy="IDENTITY") */ private $id; /** * @ORM\Column(name="path", type="string") * @Gedmo\UploadableFilePath */ private $path; /** * @ORM\Column(name="name", type="string") * @Gedmo\UploadableFileName */ private $name; /** * @ORM\Column(name="mime_type", type="string") * @Gedmo\UploadableFileMimeType */ private $mimeType; /** * @ORM\Column(name="size", type="decimal") * @Gedmo\UploadableFileSize */ private $size; public function myCallbackMethod(array $info) { // Do some stuff with the file.. } // Other methods.. }
```

For more details check the documentation.

ReferenceIntegrity

ReferenceIntegrity behavior will automate the reference integrity for referenced documents.

Annotations | YML

```
<?php namespace Document; use Doctrine\ODM\MongoDB\Mapping\Annotations as ODM; use Gedmo\Mapping\Annotation as Gedmo; /** * @ODM\Document(collection="types") */ class Type { /** * @ODM\Id */ private $id; /** * @ODM\String */ private $title; /** * @ODM\ReferenceOne(targetDocument="Article", mappedBy="type") * @Gedmo\ReferenceIntegrity("nullify") * @var Article */ protected $article; // ... }
```

For more details check the documentation.

IpTraceable

IpTraceable behavior will automate the update of IP trace on your Entities or Documents.

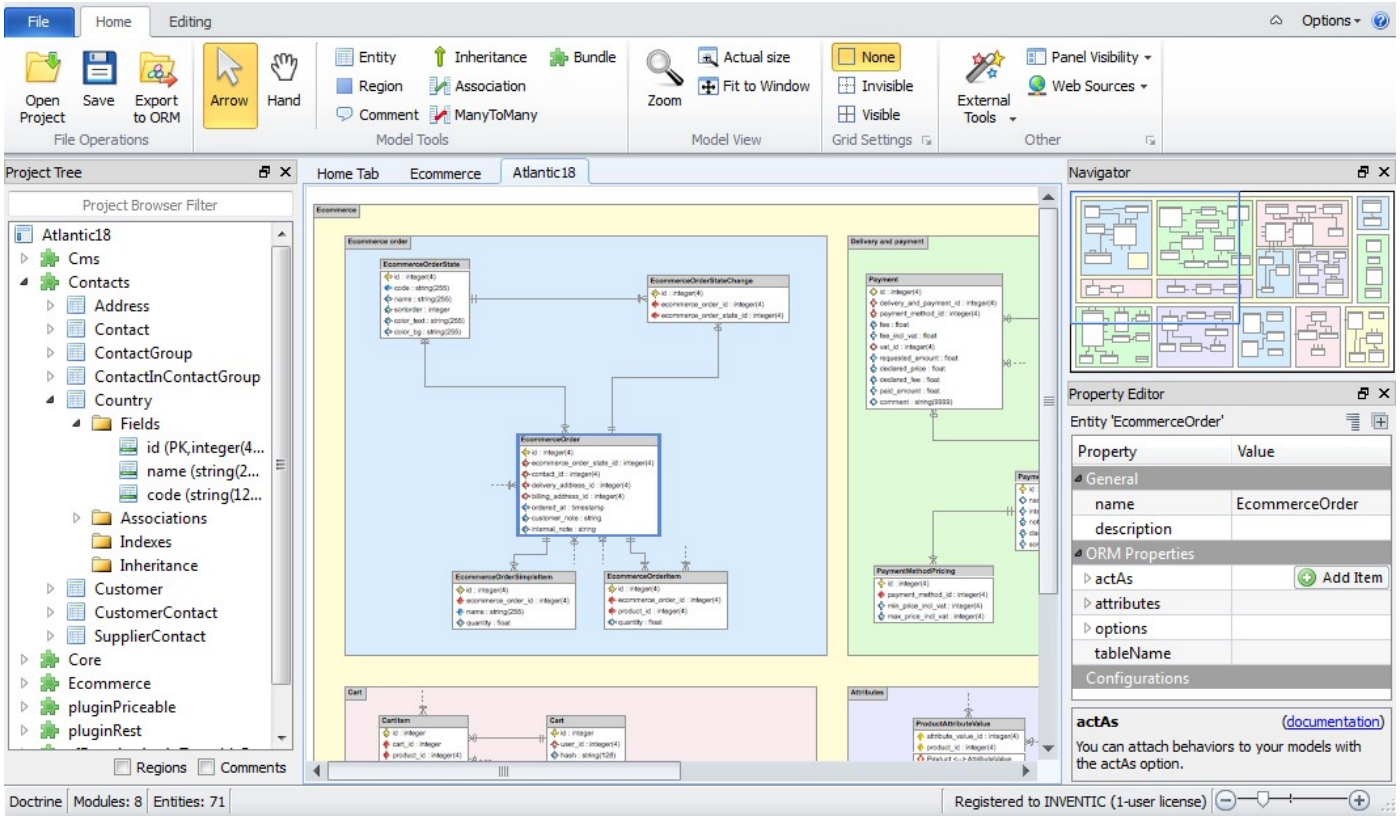
Annotations | XML | YML

```
<?php namespace Entity; use Gedmo\Mapping\Annotation as Gedmo; use Doctrine\ORM\Mapping as ORM; /** * @ORM\Entity */ class Article { /** @ORM\Id @ORM\GeneratedValue @ORM\Column(type="integer") */ private $id; /** * @ORM\Column(type="string", length=128) */ private $title; /** * @ORM\Column(name="body", type="string") */ private $body; /** * @var string $createdFromIp * * @Gedmo\IpTraceable(on="create") * @ORM\Column(type="string", length=45, nullable=true) */ private $createdFromIp; /** * @var string $updatedFromIp * * @Gedmo\IpTraceable(on="update") * @ORM\Column(type="string", length=45, nullable=true) */ private $updatedFromIp; /** * @var datetime $contentChangedFromIp * * @ORM\Column(name="content_changed_by", type="string", nullable=true, length=45) * @Gedmo\IpTraceable(on="change", field={"title", "body"}) */ private $contentChangedFromIp; public function getId() { return $this->id; } public function setTitle($title) { $this->title = $title; } public function getTitle() { return $this->title; } public function setBody($body) { $this->body = $body; } public function getBody() { return $this->body; } public function getCreatedFromIp() { return $this->createdFromIp; } public function getUpdatedFromIp() { return $this->updatedFromIp; } public function getContentChangedFromIp() { return $this->contentChangedFromIp; } }
```

For more details check the documentation.

Skipper

Visit website

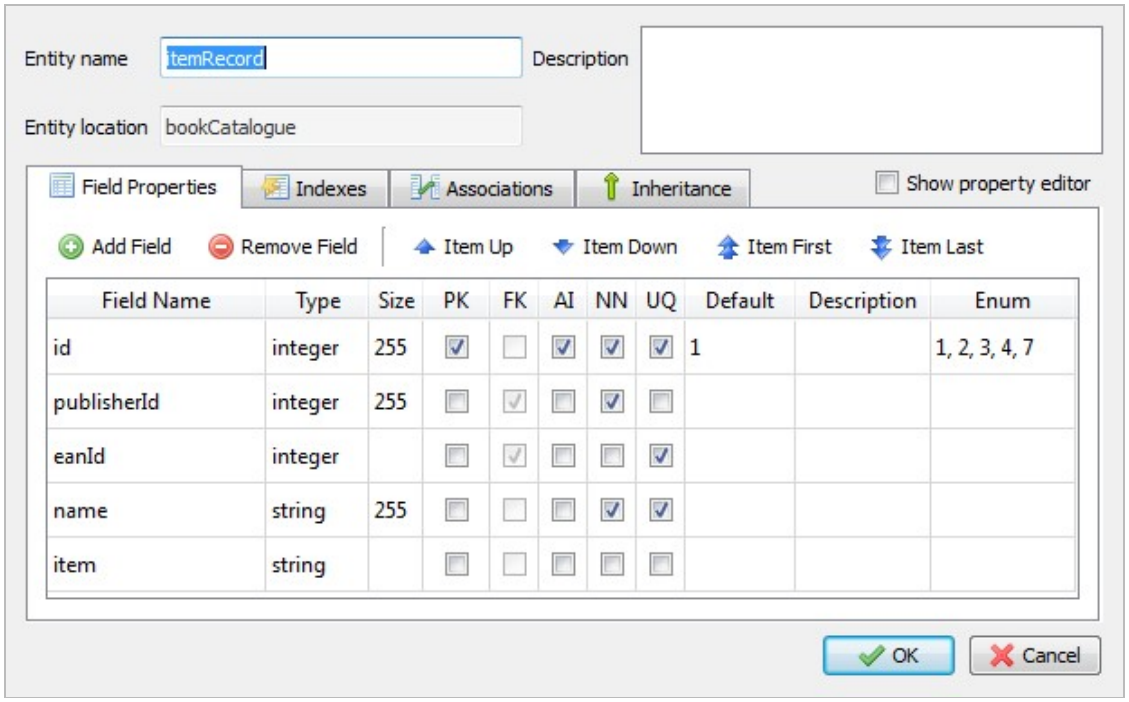


Element definitions on this page were modelled and generated by Skipper, visual schema editor for ORM frameworks.

Skipper greatly simplifies work with Doctrine 2 and saves huge amount of time. Every example entity and relation used in this Cheatsheet can be achieved with just a few clicks with Skipper. Generated code is clean and elegant and complies with all coding standards.

To learn how Skipper works visit the product tour.

Export to Doctrine 2 definitions



Entity editor

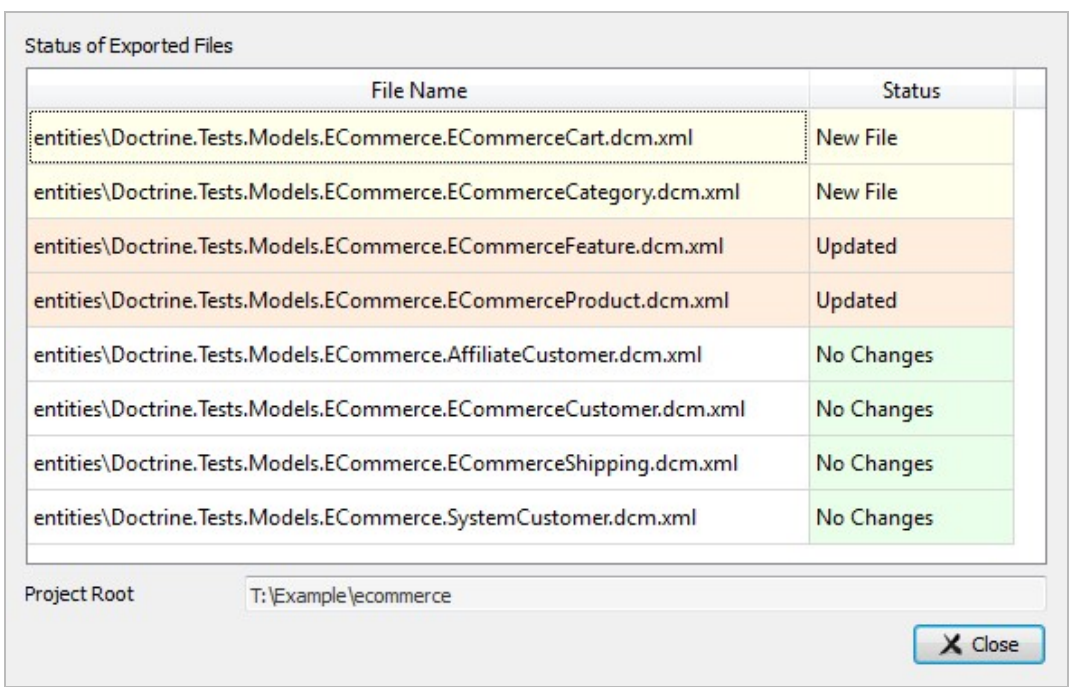
Fork me on GitHub

Skipper allows to model and export the definition for every Doctrine 2 element and its properties. Further advantages of automated export are:

- Editing and generating of definitions is fully repeatable.
- Standardized definitions are immediately ready-to-use.
- All typos and syntax errors are 100% eliminated.

Useful links:
Project export - more info about Skipper definitions export
Export to Doctrine 2 - how to export your changes to definition schema files

Import of a project



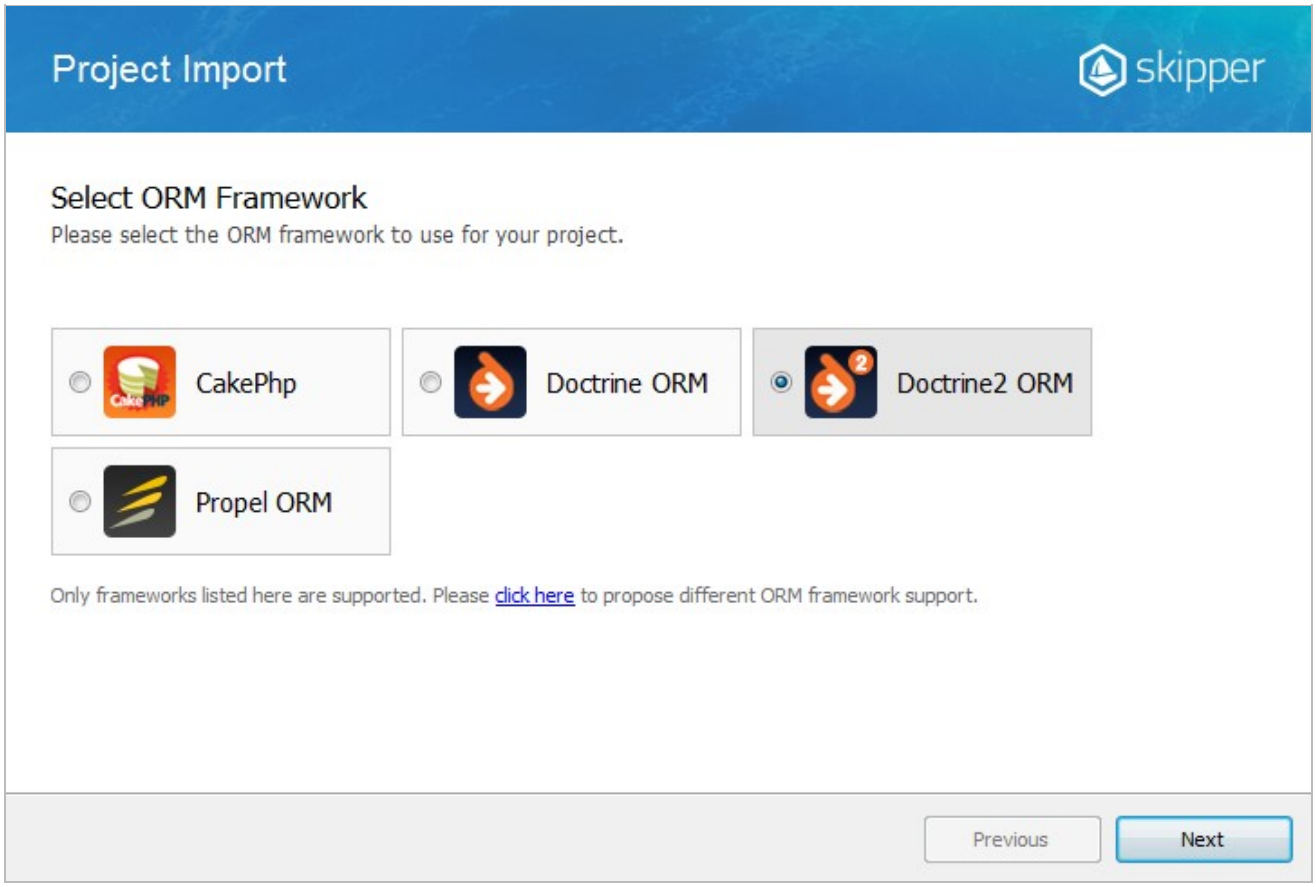
Export dialog

Any existing Doctrine 2 project can be simply and quickly imported to Skipper. This enables:

- To visualize logic of any project.
- To start to use application in any phase of the project.

Useful links:
Project import - general information about import feature
Doctrine 2 project import - how to import existing project to Skipper

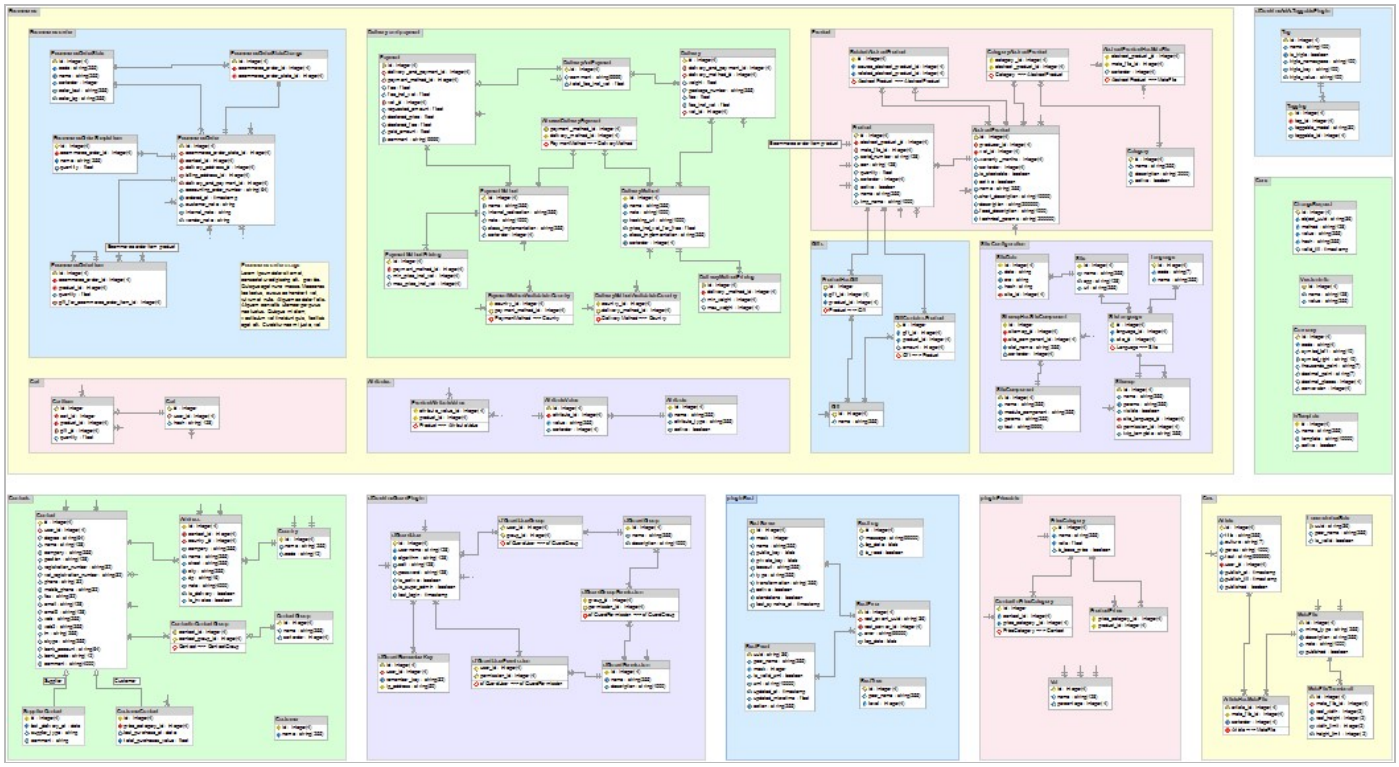
Summary of Skipper benefits



Import dialog

- Allows to create and maintain the project four times faster.
- Replaces manual definitions writing and reduces errors.
- Displays the model schema in a form of interactive enhanced ER diagram.
- Emphasizes the creative part of a project and eliminates the stereotype.
- Increases work comfort.
- Provides quality project documentation.
- Reduces requirements on knowledge and experience of programmers.
- Simplifies the cooperation between team members.

Skipper download



Atlantic model

You can try Skipper during its 14-day evaluation period. Trial version offers full application functionality without any limitation and no credit card is needed.

Download trial version from the tool websites at www.skipper18.com/download.

DOWNLOAD
SKIPPER TRIAL

See also

- Symfony website
- Zend framework website
- Doctrine2 project
- Doctrine extensions

- Symfony Cheatsheet
- Roll'n'Api

- l3pp4rd's developer blog

Do you know any other helpful or interesting sites that should be linked here?
Let us know: developers@ormcheatsheet.com

Found a typo? Something is wrong or missing in the Cheatsheet? Just fork and edit it!

ORM Cheat Sheet

Created by Inventic developed by community

If you want to leave feedback, contact us developers@ormcheatsheet.com