Base Distribution - Story # 10273

Status:	Resolved	**Priority:**	Should have
Author:	Karsten Dambekalns	**Category:**	
Created:	2010-10-15	**Assigned To:**	Karsten Dambekalns
Updated:	2011-05-05	**Due date:**	
Subject:	As Roger, I want to have Doctrine2 for persistence evaluated		

Description

- our DB structure would be compatible with other packages
- persistence is actually really usable afterwards :)
- they offered their help
- if Doctrine supports versioning, we should use it really soon

Subtasks:

Task # 11157: Create prototype to assess issues when using Doctrine2	**Resolved**

Related issues:

related to Base Distribution - Story # 11946: As Roger, I want to have Doctri...	**Resolved**

History

#1 - 2010-10-21 10:40 - Sebastian Kurfuerst

- Tracker changed from Task to Story

- Project changed from TYPO3.Flow to Core Team

- Category deleted (Persistence)

- Target version deleted (1.0 alpha 13)

#2 - 2010-10-21 10:41 - Sebastian Kurfuerst

- Target version set to 711

- Position deleted (944)

- Position set to 1

- Position changed from 1 to 9438

#3 - 2010-10-21 10:45 - Sebastian Kurfuerst

- Subject changed from Explore use of Doctrine2 for persistence to As Roger, I want to have Doctrine2 for persistence evaluated

- Position deleted (9438)

- Position set to 5469

#4 - 2010-11-03 19:36 - Benjamin Eberlei

So whats the status on this one and how can we help? :-)

#5 - 2010-11-03 20:40 - Karsten Dambekalns

Benjamin Eberlei wrote:

> *So whats the status on this one and how can we help? :-)*

Well, current status is I need to dig into Doctrine2 to see how we can use it :) Although planned for Sprint 5 we will have to push it to the next one, I guess. So, it would be really cool if we could meet in the second half of November, either virtually or in real life...

#6 - 2010-11-15 12:56 - Robert Lemke

- Target version changed from 711 to 732

- Position deleted (5476)

- Position set to 3

#7 - 2010-11-23 20:52 - Benjamin Eberlei

Hey Karsten,

i guess virtually is simpler. I am currently swamped with work :-)

I am mostly hanging in #doctrine-dev on Freenode IRC under the "beberlei" nick, so we can meet up there maybe and start digging into the details?

#8 - 2010-11-27 16:13 - Benjamin Eberlei

Ok, so i prototyped a first version of this integration:

https://github.com/beberlei/flow3-doctrine2

However digging deeper into the Flow3 persistence layer I found some shortcomings that I can't overcome now:

- With Doctrine 2 you HAVE to specify at least one field as @Identifier, however for example in the Blog Example App there is sometimes neither @uuid nor @identifier specified in the models. Is this not a requirement in the Flow Persistence layer? Does each entity have both UUID and Identifier fields? I.e. IDentifier fields are the primary key, and the UUID is a globally unique identifier?
- The association management with "@var SplObjectStorage<type>" or "@var array<type>" has considerable shortcomings. You cannot guess from this information if the Relation should be OneToMany or ManyToMany. Additionally with this notation you cant specifiy in two entities if the association is bi-directional, or if there are two uni-directional relations pointing from two entities to each other.
- Doctrine2 does not support Value Objects
- Doctrine2 by default sets all associations as lazy. @Lazy is implicit. @Eager would be interesting.

Personally I would prefer to see an implementation that uses the Doctrine Annotations, not the Flow3 ones. They are simple but allow for much more control. I do love Flow3s focus on the domain model. But a mapping layer that is too simple will annoy developers big-time.

#9 - 2010-11-29 20:06 - Benjamin Eberlei

I just realized there is a non-simple to solve clash between Doctrine generated proxies and Flow3 Generation of objects.

#10 - 2010-11-29 23:32 - Benjamin Eberlei

I refactored the integration after a lengthy discussion with Karsten and committed it to the repository.

1. Taking http://misko.hevery.com/2008/09/30/to-new-or-not-to-new/ at heart I now assume Entities are never created through the ObjectManager. That simplifies Doctrine integration alot.
2. I implemented all the missing methods (simpler without the object manager dependency for entities).
3. I dropped UUID support. I propose to change PersistenceManagerInterface::getObjectByIdentifier($identifier) to PersistenceManagerInterface::getObjectByIdentifier($className, $identifier) or at least ::getObjectByIdentifier($identifier, $className = null)
4. I added support for the getCountByQuery() method through a DQL to SQL Rewrite that allows this.
5. FLOW3MetadataDriver now extends the AnnotationDriver mapping all additional primitive properties using the (@var) annotations. Support for @var array and @var SplObjectStorage was dropped. They lack in detail to specify relations.

The Query object is now working exactly as the FLOW3 Persistence API specifies.

Additionally I propose the following changes:

- Add a method markDirty() to PersistenceManagerInterface
- Only execute persistAll() if the PersistenceManagerInterface was marked dirty before. This is much better than starting to mark actions as "read-only" because people will obviously forget this and start complaining about performance. In contrast if you forget to mark a persistence manager as dirty you will realize something went wrong because no changes have been written to the database.

And integration into the flow3 workflow:

- In Development Mode use Doctrine\ORM\Tools\SchemaValidator to check if the mappings are valid and the database schema up to date.
- You can use ClassMetadata accessible from $em->getClassMetadata($className) as a basis to write an admin generator.

#11 - 2010-11-30 17:48 - Karsten Dambekalns

Benjamin Eberlei wrote:

> *1. Taking http://misko.hevery.com/2008/09/30/to-new-or-not-to-new/ at heart I now assume Entities are never created through the ObjectManager. That simplifies Doctrine integration alot.*

I read this, and conclude it deals too much with Java specifics and focuses on constructor injection only. For FLOW3 it is no problem to have constructor arguments you must supply manually and dependencies that are injected automatically at the same time. And the "field reference" problem he points out assumes two things:

1. whenever something is injected as constructor argument, it is kept in a field - which is neither always the case, nor always a problem (we have @transient, and singletons will not be persisted anyway)
2. setter injection does not exist - but we have it. see above for the "field reference problem".

Now, why would I want DI in an entity? Assume we have a Customer object that needs access to some service, let's say to generate "top customer program membership number". You could do that externally, but why not simply call makeTopCustomerMember() on the Customer instead of using a factory and whatnot.

The second thing we use the ObjectManager for is to make sure AOP is woven in. And AOP is at the heart of our security model, thus it must be possible to write a policy that blocks access to methods on a model depending on a role or setting.

As we agreed the "real" solution would be to use new instead of the ObjectManager. See #11169 for an exploration task.

> *5. FLOW3MetadataDriver now extends the AnnotationDriver mapping all additional primitive properties using the (@var) annotations. Support for @var array and @var SplObjectStorage was dropped. They lack in detail to specify relations.*

I would like to keep support for them. If I have onedirectional relations @var array<Foo\Bar> should tell enough, no? Same for bidirectional relations I want to manually manage (for whatever reason). Plus, even if I add more info using @OneToMany it would be nice if it would take the collection type (inside the angled brackets) as default for targetEntity.

Also, looking at the code of FLOW3AnnotationsDriver it seems you misunderstood getUuidPropertyName() - the uuid property is (despite the name, sorry!) not required to be a UUID, it can be anything you want to use **instead** of the generated UUID, e.g. a customer number.

So much for now. :)

#12 - 2010-12-01 00:53 - Benjamin Eberlei

The blog entry by misko speaks about the difference of injectables (AudioDevice, MusicPlayer) vs newables (Song). A Song is clearly an Entity. Evans talks about this in detail on the Entity and Service chapters in DDD.

The requirement to have it being instantiated through the Object Manager means you always need the OM for your domain to work. Even in the tests. I don't doubt that the object manager has no problems with constructor arguments. I just think that objectManager::create ia a huge difference in dependency nightmare compared to just new.

There is no need to have an service injected into an entity upon construction. The reasons are simple:

1. An entity is only ever created through an repository, so the repository can always pass a service to an entity.
2. Controllers and services control the use-cases / stories of the domain. These retrieve entities from a repository and can pass other services into the entities. No need to have them injectable upon construction.

Your Top customer program membership number example shows this quite easily, double dispatch pattern is the key here:

```
public function makeTopCustomerMember(CustomerNumberServiceInterface $service)
{
    $this->topCustomerMember = $service->generateNumber($this);
}
```

The CustomerNumberServiceInterface can be injected into the controller, a service-layer class or the repository and then passed to each entity.

I do get your desire to have entities AOPable, but i would rather go with an optional dependency on php-test-helper extension then and tell people to use this (maybe even take only the AOP relevant part out of it and start a new extension). The performance hit is probably less than constructing each and every entity through the object-manager, minus the negative impact of having a dirty domain model that depends on the object-manager (and through that on hundrets of other potential dependencies).

#13 - 2010-12-01 11:29 - Karsten Dambekalns

Benjamin Eberlei wrote:

> *The requirement to have it being instantiated through the Object Manager means you always need the OM for your domain to work. Even in the tests.*

In the unit tests no real ObjectManager is ever used, all test targets are created with new or getMock(). And an Entity indeed should never use the OM but have everything injected and/or passed in through some service or container. If that is the case, you can use every class without the OM, but of course must inject the dependencies yourself (in a factory or with some other DI container).

> *1. An entity is only ever created through an repository, so the repository can always pass a service to an entity.*

What makes you think that? A factory should be used if object creation is complex enough to warrant it, but if possible direct creation would be better.

> *I do get your desire to have entities AOPable, but i would rather go with an optional dependency on php-test-helper extension then and tell people to use this (maybe even take only the AOP relevant part out of it and start a new extension).*

Telling people they can have security "done right" only when they install some (obscure) extension is a drawback. Anyway, we'll look into making new possible.

#14 - 2010-12-01 11:45 - Benjamin Eberlei

If Security is an important aspect of a domain on the entity level and implemented by the Object Manager you cannot get around using the Object Manager and all its dependencies in the tests.

In my opinion Security on the level of Entity Methods seems a lot like micromanaging. Controller and Service layer security is probably the only thing people want. Allowing entity level security "because we can" is sort of shooting with canons at birds. Just because something is possible doesnt mean its a good idea to encourge/support users to do it.

#15 - 2011-01-06 14:37 - Sebastian Kurfuerst

- Status changed from Accepted to Resolved
- Position deleted (16)
- Position set to 16

#16 - 2011-05-05 13:49 - Robert Lemke

- Project changed from Core Team to Base Distribution
- Target version deleted (732)