


DDD Mapping Entity Framework Data Model to Domain models

c# domain-driven-design entity-framework entity-framework-6

English (en) ▼



Question

I understand that Entity data models should be separated from real domain models, to avoid coupling between infrastructural concerns and domain itself, but i was wondering if all domain properties do not have public setters, how can we map from data model to domain model, especially if repository implementation resides in infrastructural part of project, so we can't use internal property setters.

```
class DomainModel
{
    string SomeProperty {get; private set;}
}
```

[Robert](#)

Accepted Answer

In a schema where you have an intermediate "Data Model", Entity Framework no longer has control over how your domain entities are instantiated. Your Repository has. So they don't necessarily need public setters, you can also rehydrate them using the constructor.

One such technique is explained here : <https://vaughnvernon.co/?p=879>

Note that a simpler alternative can be to avoid the additional data model and use private setters (see <https://lostechies.com/jimmybogard/2014/04/29/domain-modeling-with-entity-framework-scorecard/>), if you consider the little impact EF will have on your entities a reasonable tradeoff.

[guillaume31](#)

Fastest Entity Framework Extensions ➤

[+ Bulk Insert ➤](#) [- Bulk Delete ➤](#) [↺ Bulk Update ➤](#) [🔗 Bulk Merge ➤](#)

Popular Answer

I understand that Entity data models should be separated from real domain models

Not true

avoid coupling between infrastructural concerns and domain itself

True

You *CAN* use EF to directly persist your domain models, however you should not couple your domain models to EF.

What does this look like?

Let's say you have a project called **Domain** which has your models in it, and another called **DataStore** which has your repositories to persist said models (using EF)

Now usually with EF you would use attributes and all sorts of nonsense on the models you want to persist, but this pollutes & couples those models to EF as a storage mechanism & adds dependancies to EF from your pure **Domain** project - this is what we are trying to avoid.

EF6 to the rescue, have a look at the [EF6 FluentApi](#), you are able to configure your Domain models to work with EF without adding any EF specific attributes or dependancies to the `Domain` project.

Primary Key?

```
modelBuilder.Entity<OfficeAssignment>().HasKey(t =>
    t.InstructorID);
```

Index?

```
modelBuilder
    .Entity<Department>()
    .Property(t => t.Name)
    .HasColumnAnnotation("Index", new IndexAnnotation(new
        IndexAttribute()));
```

Now this will quickly become a pain if you do this for each and every object, BUT if you stick to some conventions (or use a base type to make this even easier)

Then you can use reflection to do this `auto-magically` for all Entities in your domain.

Good luck

[shenku](#)

 [View more on Stack Overflow](#)

Licensed under: [CC-BY-SA](#) with [attribution](#)
Not affiliated with [Stack Overflow](#)