# Which layer of the application should contain DTO implementation

asp.net asp.net-mvc c# entity-framework wcf

English (en)

## Question

Lately I've been hearing a lot about DTOs and how useful they are but I can't find a good example of using it in ASP.NET context.

Let's say I use three tier architecture:

1. Data layer(using Entity Framework)
2. Business Layer(WCF Service)
3. Presentation Layer (MVC 4.0 web application)

Where should I convert from the EF Employee object to an EmployeeDTO POCO?

Lets say I do the conversion in the Data Access layer but what happens in the WCF service? Should it then be converted to another `DataMember` object and when it get's to the UI layer(MVC web app) should it then be converted for the third time to a model? I would appreciate it if someone could please clear this for me

Denys Wessels

## Accepted Answer

In similar situation I used to put dto's into *Core* which is known to all three. So you have

```
     Core
      |
 -----------
 |    |    |
DAL   BL   PL
```

Each layer can operate with `Core.Dto.Employee`. Each layer also exposes `Core.Dto.Employee` externally in its API. But internally each layer can transform/adapt `Core.Dto.Employee`, e.g. you read from database `EF.Employee` and later convert it to `Core.Dto.Employee`. Transformation is contained by the layer's boundary.

If you have several different models to represent same thing throughout the layers, for example PL wants `PL.Employee` and DAL operates on `EF.Employee`, you will end up with a mess.

oleksii

## Popular Answer

Your service layer exposes DTO's. This means that in the service layer you define data contracts as you would like them to be exposed to the outside world. In most cases they are flattened entities that not necessarily have the same structure as your database entities.

It is the responsibility of your service layer to use business/data layer and construct the DTO's that you expose to the outside world.

What you use in your business and data layer depends on the architecture. You could have a domain model that is mapped with code first. In that case, the service layer maps domain entities to data contracts (DTO's). If you don't have a domain model (anemic model), then you could as well just map the database directly to your DTO's.

The ASP.NET MVC site consumes the service, and maps the DTO's it receives to dedicates view models that are then passed to the specific view.

In addition, you may decide to also split queries from commands. This is a good approach because the DTO's that you get back as the reqult of a query are totally different than commands that you send to the service. A command only contains what's needed to execute the command and contain the business intend what you want to achieve, while a query returns a flattened model of what you need in the UI.

Other remarks:

- Don't expose your database entities.
- Don't convert in business layer, as it's not business logic.

L-Four

View more on Stack Overflow