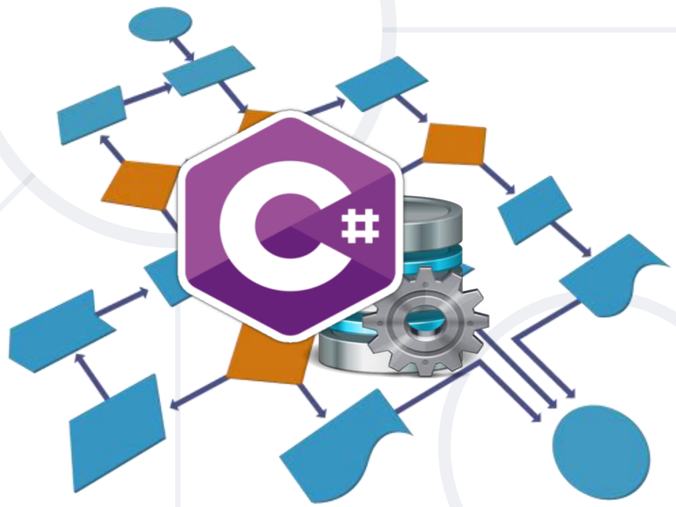
C# Auto Mapping Objects

Manual Mapping

and AutoMapper Library



SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

Have a Question?

SoftUni Foundation

sli.do

#csharp-db

Data Transfer Objects

Definition and Usage

What is a Data Transfer Object?

- A **DTO** is an object that **carries data** between processes
  - Used to **aggregate** only the **needed information** in a single call
  - Example: In web applications, between the **server** and **client**
- Doesn't contain any logic – only **stores values**

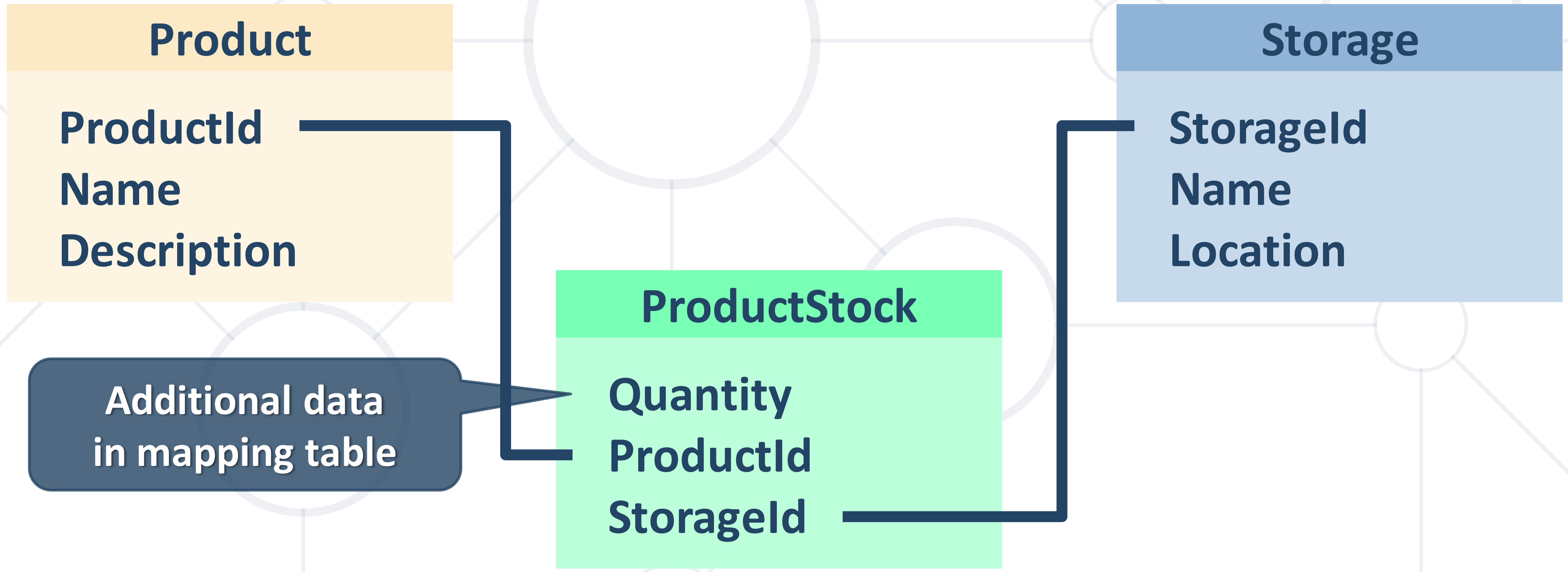
```
public class ProductDTO
{
  public string Name { get; set; }
  public int StockQty { get; set; }
}
```

DTO Usage Scenarios

- **Remove** circular references
- **Hide** particular properties that **clients** are not supposed to view
- **Omit** some properties in order to **reduce** payload **size**
- **Flatten** object graphs that contain nested objects to make them more convenient for clients
- **Decouple** your service layer from your database layer

Manual Mapping

- Relationship Diagram



Manual Mapping (2)

- Get product name and stock quantity in a new DTO object

```
var product =
context.Products.FirstOrDefault();
var productDto = new ProductDTO
{
  Name = product.Name,
  StockQty = product.ProductStocks
    .Sum(ps => ps.Quantity)
};
```

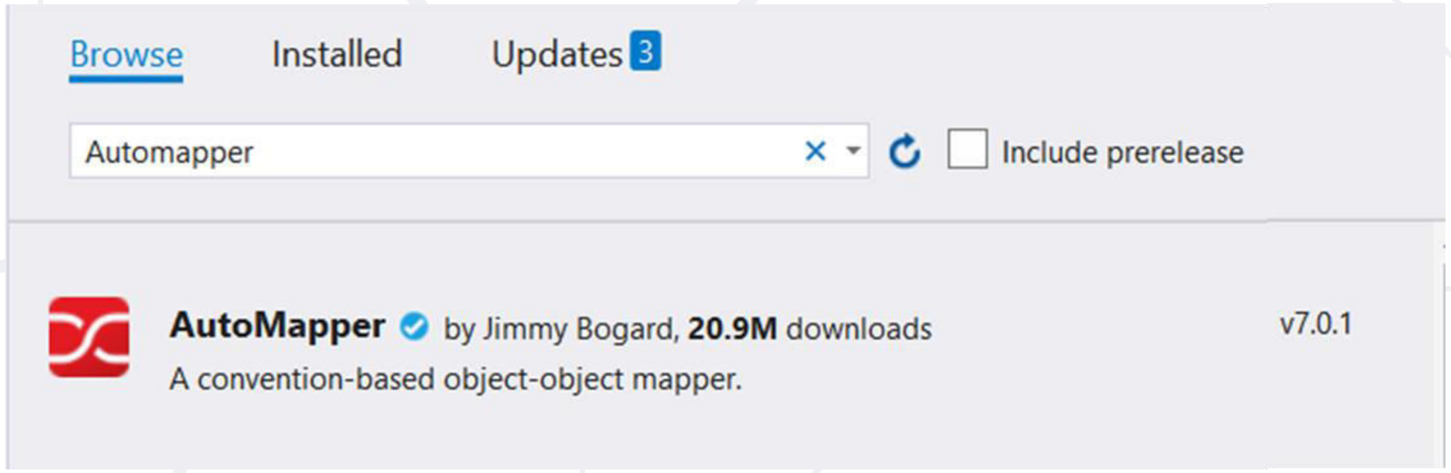
Aggregate information from mapping table

AutoMapper Library

Automatic Translation of Domain Objects

What is AutoMapper?

- Library to eliminate **manual mapping** code
- Available as a **NuGet** Package
- Official **GitHub Page**



Automapper

AutoMapper by Jimmy Bogard, **20.9M** downloads v7.0.1
A convention-based object-object mapper.

```
Install-Package AutoMapper
```

Initialization and Configuration

- AutoMapper offers a **static service** for use and configuration
  - Add mappings between objects and DTOs

Source

```
Mapper.Initialize(cfg => cfg.CreateMap<Product,
ProductDTO>());
```

Target

- Properties will be mapped **by name**

```
var product = context.Products.FirstOrDefault();
ProductDTO dto = Mapper.Map<ProductDTO>(product);
```

Multiple Mappings

- You can configure all mapping configurations at once

```
Mapper.Initialize(cfg =>
{
  cfg.CreateMap<Product, ProductDTO>();
  cfg.CreateMap<Order, OrderDTO>();
  cfg.CreateMap<Client, ClientDTO>();
  cfg.CreateMap<SupportTicket, TicketDTO>();
});
```

Custom Member Mapping

- Map properties that don't match naming convention

```
Mapper.Initialize(cfg =>
   cfg.CreateMap<Product, ProductDTO>()
      .ForMember(dto => dto.StockQty,
                 opt => opt.MapFrom(src =>
                   src.ProductStocks.Sum(p =>
p.Quantity))));
```

Destination property

Source

Flattening Complex Properties

- **AutoMapper** can also be used to flatten complex properties

```
Mapper.Initialize(cfg =>
   cfg.CreateMap<Event, CalendarEventViewModel>()
      .ForMember(dest => dest.Date,
                 opt => opt.MapFrom(src => src.Date.Date))
      .ForMember(dest => dest.Hour,
                 opt => opt.MapFrom(src => src.Date.Hour))
      .ForMember(dest => dest.Minute,
                 opt => opt.MapFrom(src => src.Date.Minute)));
```

Flattening Complex Objects

- **Flattening** of related objects is automatically supported

```
public class OrderDTO
{
  public string ClientName { get; set; }
  public decimal Total { get; set; }
}
```

- **AutoMapper** understands ClientName is the **Name** of a **Client**

```
Mapper.Initialize(cfg => cfg.CreateMap<Order, OrderDTO>());
OrderDTO dto = Mapper.Map<Order, OrderDTO>(order);
```

Unflattening Complex Objects

- **Unflattening** of related objects is automatically supported

```
public class OrderDTO
{
  public string ClientName { get; set; }
  public decimal Total { get; set; }
}
```

- **AutoMapper** understands ClientName is the **Name** of a **Client,** but to unflatten it, it needs **ReverseMap**

```
Mapper
    .Initialize(cfg => cfg.CreateMap<Order, OrderDTO>()
    .ReverseMap());
```

Mapping Collections

- EF Core uses **IQueryable<T>** for all DB operations
  - AutoMapper can work with IQueryable<T> to map classes
- Using AutoMapper to map an entire DB collection:

```
var posts = context.Posts
  .Where(p => p.Author.Username == "gosho")
  .ProjectTo<PostDto>()
  .ToArray();
```

IQueryable<Post>

IQueryable<PostDto>

- Works like an automatic **.Select()**
  - EF Core generates **optimized SQL** (like with an **anonymous object**)

AutoMapper.Collection

- Adds ability to map collections to existing collections **without re-creating** the collection object
- Will **Add/Update/Delete** items from a preexisting collection object based on user defined equivalency between the collection's generic item type from the source collection and the destination collection

```
Mapper
    .Initialize(cfg => cfg.AddCollectionMappers());
```

AutoMapper.Collection.EntityFrameworkCore

- **Automapper.Collection.EntityFrameworkCore** will help you mapping of EntityFrameowrk Core DbContext-object

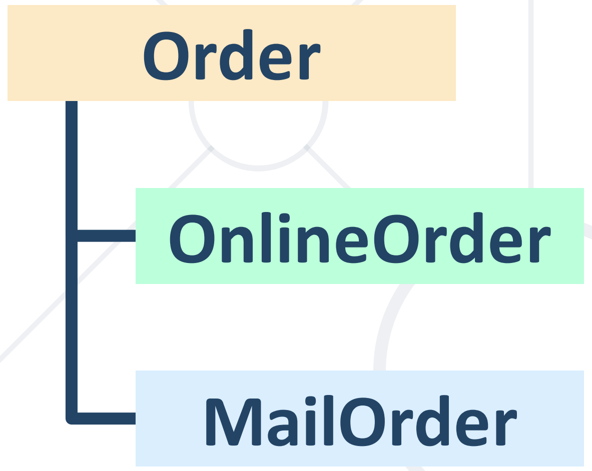
```
Mapper.Initialize(cfg =>
{
     cfg.AddCollectionMappers();
     cfg.SetGeneratePropertyMaps
       <GenerateEntityFrameworkCorePrimaryKeyPropertyMaps<Context>>();
     // Configuration code
});
```

- Comparing to a single existing Entity for updating

```
dbContext.Orders.Persist().InsertOrUpdate<OrderDTO>(newOrderDto);
dbContext.Orders.Persist().InsertOrUpdate<OrderDTO>(existingOrderDto);
dbContext.Orders.Persist().Remove<OrderDTO>(deletedOrderDto);
dbContext.SubmitChanges();
```

Inheritance Mapping

- Inheritance chains are defined via Include()
- AutoMapper chooses the most appropriate child class



Register with parent

```
Mapper.Initialize(cfg => {
   cfg.CreateMap<Order, OrderDto>()
      .Include<OnlineOrder, OnlineOrderDto>()
      .Include<MailOrder, MailOrderDto>();
   cfg.CreateMap<OnlineOrder, OnlineOrderDto>();
   cfg.CreateMap<MailOrder, MailOrderDto>();
});
```

Child mapping

Mapping Profiles

- We can extract our configuration to a class (called a **profile**)

```
public class ForumProfile : Profile
{
  public ForumProfile()
  {
    CreateMap<Post, PostDto>();
    CreateMap<Category, CategoryDto>();
  }
}
```

using AutoMapper;

- Using our configuration class:

```
Mapper.Initialize(cfg => cfg.AddProfile<ForumProfile>());
```

Summary

- To reduce round-trip latency and payload size, data is transformed into a **DTO**
- **AutoMapper** is a library that automates this process and reduces boilerplate code
- Complex objects can be **flattened** to fractions of their sizes

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech we know sports

telenor

SoftwareGroup doing it right

NETPEAK

SmartIT

Postbank Решения за твоето утре

INDEAVR Serving the high achievers

INFRAGISTICS®

STEMO® Computer Systems & Software

SUPERHOSTING.BG

SoftUni Organizational Partners

SoftUni Foundation

IS ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

Lukanet.com

WORLD OF MYTHS

License