**AutoMapper in C#**

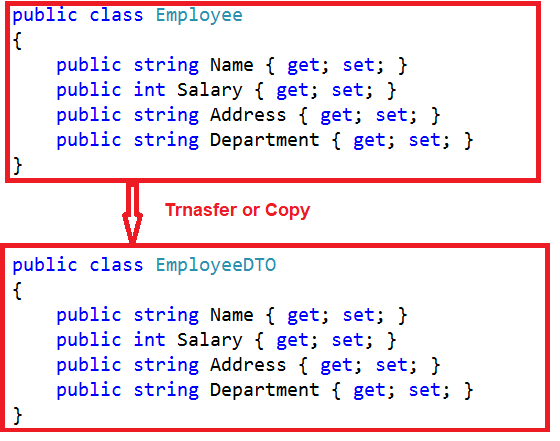
**AutoMapper in C# with Examples**

In this article, I am going to discuss **AutoMapper in C#** with some simple examples. At the end of this article, you will be having a very good understanding of the following pointers.

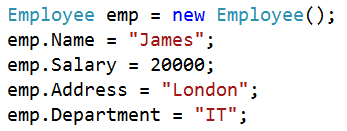
1. **Why do we need to use AutoMapper in C#?**
2. **What is AutoMapper in C#?**
3. **How do I use AutoMapper in C#?**
4. **Multiple Examples to Understand AutoMapper in C#.**
5. **What will happen if the source and destination property names are different?**
6. **How to map two properties when the names are different using automapper?**

**Why do we need AutoMapper in C#?**

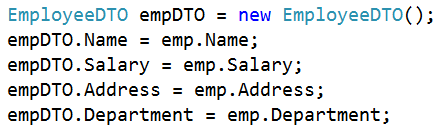
Let’s understand why do we need automapper in C# with an example. Let’s say we have two classes such as **Employee** and **EmployeeDTO** as shown in the below image.



Now, if you want to copy the data or transfer the data from the **Employee** object to the **EmployeeDTO** object, then in the traditional approach first you need to create and populate the Employee object as shown in the below image.



Once you have the employee object, then you need to create the **EmployeeDTO** object as well as need to copy the data from the employee object to the employee DTO object as shown in the below image.



**The complete example is given below.**

The following example, copy the data from one object to another object in the traditional approach i.e. without using the C# automapper.

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

Employee emp = new Employee**()**;

emp.Name = "James";

emp.Salary = 20000;

emp.Address = "London";

emp.Department = "IT";

EmployeeDTO empDTO = new EmployeeDTO**()**;

empDTO.Name = emp.Name;

empDTO.Salary = emp.Salary;

empDTO.Address = emp.Address;

empDTO.Department = emp.Department;

Console.WriteLine**(**"Name:" + empDTO.Name + ", Salary:" + empDTO.Salary + ", Address:" + empDTO.Address + ", Department:"+ empDTO.Department**)**;

Console.ReadLine**()**;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**}**

**}**

That’s it. Now if you run the application, then you will get the output as expected. But, tomorrow what will you do if the data i.e. the properties in the class are increased. Then you need to write the code for each property data moving from the source class to the destination class. That means the Mapping of code is done again and again between the source and the destination.

In real-time projects, many times we need to map the objects between the UI/Domain or Service/Domain layers. Mapping the objects between them is very hectic using the traditional approach that we discussed in the above example. So, is there any simplest solution by which we can map two objects? Yes, there is, and the solution is **AutoMapper**.

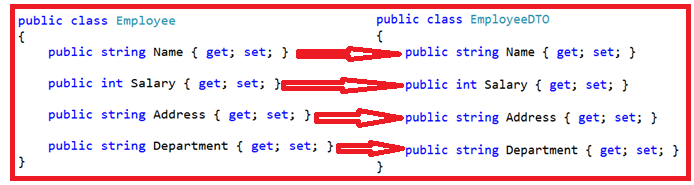
**What is AutoMapper in C#?**

The **AutoMapper in C#** is a mapper between two objects. That is AutoMapper is an object-object mapper. It maps the properties of two different objects by transforming the input object of one type to the output object of another type.

It also provides some interesting facts to take the dirty work out of figuring out how to map an object of type A with an object of type B as long as the object of type B follows AutoMapper’s established convention. As we progress in this course, you will see the power of automapper.

**How do I use AutoMapper in C#?**

Let us understand how to use automapper in C# with a simple example using a console application. We are going to map the same **Employee** class with **EmployeeDTO** class that we discussed in the first example. We need to map each **Employee properties** to the correspondent **EmployeeDTO properties** using AutoMapper as shown in the below image.



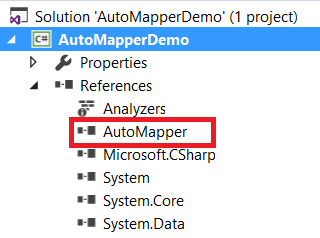
**Let’s discuss the step-by-step procedure to use AutoMapper in C#.**

**Step1: Installing the AutoMapper library**

The AutoMapper is an open-source library present in [**GitHub**](https://github.com/AutoMapper). To install this library, open the Package Manager Console window and then type the following command and press enter key to install the AutoMapper library in your project:

**PM> Install-Package AutoMapper**

Once you installed the **AutoMapper** library, then it will add a reference to the **AutoMapper dll** which you can find in the project references section as shown in the below image.

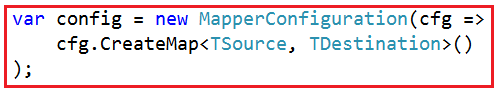


Once you installed the Automapper in your project, then you can use it in many different ways. We will discuss all these options in a later article. In this article, I am going to discuss some simple examples to make you understand how exactly the AutoMapper works in a project.

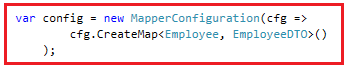
We are going to work with the same example i.e. we are going to Map the **Employee object** with the **EmployeeDTO object** but here we are going to use **AutoMapper**.

**Step2: Initializing or Configuring the AutoMapper**

Once you have defines your types (i.e. classes) then you can create a mapper for the two types using the constructor of MapperConfiguration class. You can create only one **MapperConfiguration instance per AppDomain** and should be instantiated during the application start-up. The syntax to create the MapperConfiguration instance is given below.

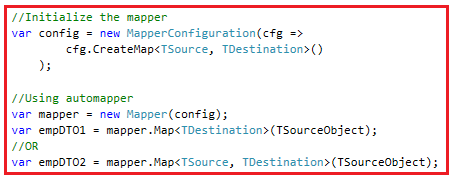


The type on the left is the source type i.e. **TSource**, in our example, it will be going to **Employee object**, and the type on the right is the destination type i.e. **TDestination**, in our example, it will be going to **EmployeeDTO object**. So, two maps the **Employee** with **EmployeeDTO**, you need to create the mapper configuration as shown below.



**Using AutoMapper:**

The following image shows how to initialize, and use automapper in C#.



**Below is the complete code**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Initialize the mapper

var config = new MapperConfiguration**(**cfg =**>**

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

**)**;

//Creating the source object

Employee emp = new Employee

**{**

Name = "James",

Salary = 20000,

Address = "London",

Department = "IT"

**}**;

//Using automapper

var mapper = new Mapper**(**config**)**;

var empDTO = mapper.Map**<**EmployeeDTO**>(**emp**)**;

//OR

//var empDTO2 = mapper.Map<Employee, EmployeeDTO>(emp);

Console.WriteLine**(**"Name:" + empDTO.Name + ", Salary:" + empDTO.Salary + ", Address:" + empDTO.Address + ", Department:" + empDTO.Department**)**;

Console.ReadLine**()**;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**}**

**}**

When you run the application, then it will display the data as expected.

**What will happen if the source and destination property names are different?**

Let’s have a look at changing the property name of the source and destination object. Let’s change the **EmployeeDTO** class **Name** and **Department** property to **FullName** and **Dept** and run the application and see what happened.

**Below is the complete example.**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Initialize the mapper

var config = new MapperConfiguration**(**cfg =**>**

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

**)**;

//Creating the source object

Employee emp = new Employee

**{**

Name = "James",

Salary = 20000,

Address = "London",

Department = "IT"

**}**;

//Using automapper

var mapper = new Mapper**(**config**)**;

var empDTO = mapper.Map**<**EmployeeDTO**>(**emp**)**;

//OR

//var empDTO2 = mapper.Map<Employee, EmployeeDTO>(emp);

Console.WriteLine**(**"Name:" + empDTO.FullName + ", Salary:" + empDTO.Salary + ", Address:" + empDTO.Address + ", Department:" + empDTO.Dept**)**;

Console.ReadLine**()**;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string FullName **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**public** string Dept **{** **get**; **set**; **}**

**}**

**}**

With the above changes in place, now if you run the application, then you should get the following output.

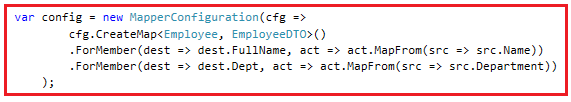
What will happen if the source and destination property names are different?

From the above output, it clearly shows that the **Name** and **Department** are empty which means these two properties are not mapped from the Source type to the Destination type.

**Note:** When the property names are different in Source and Destination types, then by default the C# Automapper will not map those properties

**How to map two properties when the names are different using automapper?**

The answer is by using the **ForMember** option. So, to the Map the **Name** property with the **FullName** property and the **Department** property with the **Dept** property, you need to map these two properties in the mapping configuration as shown below.



**Note:** We will discuss the **ForMember** and **MapForm** options in detail in our upcoming articles.

**Below is the complete code.**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Initialize the mapper

var config = new MapperConfiguration**(**cfg =**>**

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

.ForMember**(**dest =**>** dest.FullName, act =**>** act.MapFrom**(**src =**>** src.Name**))**

.ForMember**(**dest =**>** dest.Dept, act =**>** act.MapFrom**(**src =**>** src.Department**))**

**)**;

//Creating the source object

Employee emp = new Employee

**{**

Name = "James",

Salary = 20000,

Address = "London",

Department = "IT"

**}**;

//Using automapper

var mapper = new Mapper**(**config**)**;

var empDTO = mapper.Map**<**EmployeeDTO**>(**emp**)**;

//OR

//var empDTO2 = mapper.Map<Employee, EmployeeDTO>(emp);

Console.WriteLine**(**"Name:" + empDTO.FullName + ", Salary:" + empDTO.Salary + ", Address:" + empDTO.Address + ", Department:" + empDTO.Dept**)**;

Console.ReadLine**()**;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string FullName **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**public** string Dept **{** **get**; **set**; **}**

**}**

**}**

Now run the application and you should see the output as expected.

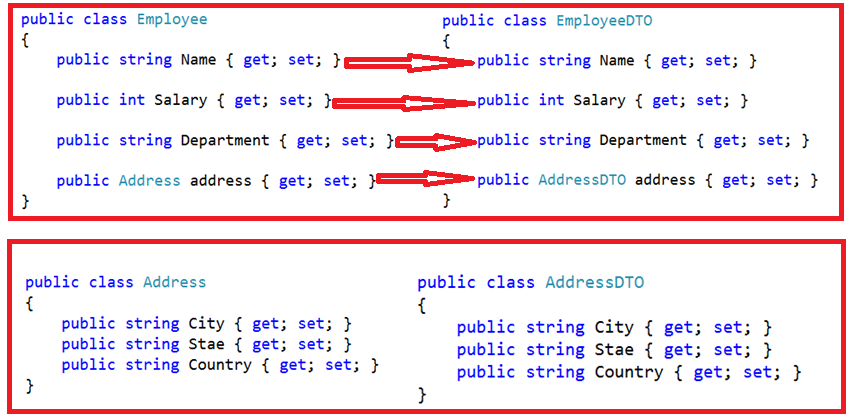
**AutoMapper Complex Mapping in C#**

**AutoMapper Complex Mapping in C# with Examples**

In this article, I am going to discuss the **AutoMapper Complex Mapping in C#** with some examples. Please read our previous article before proceeding to this article where we discussed the basics of [**Automapper in C#**](https://dotnettutorials.net/lesson/automapper-in-c-sharp/) with some simple examples. At the end of this article you will understand what is AutoMapper Complex Mapping and when and how to use **AutoMapper Complex Mapping in C#**with examples.

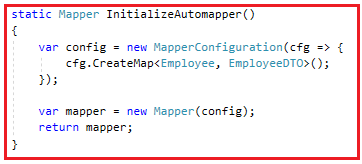
**What is** **AutoMapper Complex Mapping in C#?**

When both the type involved in the mapping contains properties of the complex type then in such scenarios we need to use the AutoMapper Complex Mapping in C#**.**Let us understand the AutoMapper Complex Mapping with an example. We are going to use the following four classes for this demo.



**Business Requirement**:

Our requirement is to map the **Employee** object to **EmployeeDTO** object. To make this demo simple, here we created both the classes with the same property names. But the thing that we need to keep in mind here is, we created the address property as a complex type. Then we are creating a static method i.e. **InitializeAutomapper** where we write the mapping code as shown in the below image.



**Below is the complete code.**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Step1: Create and populate the Employee object

Address empAddres = new Address**()**

**{**

City = "Mumbai",

Stae = "Maharashtra",

Country = "India"

**}**;

Employee emp = new Employee

**{**

Name = "James",

Salary = 20000,

Department = "IT",

address = empAddres

**}**;

var mapper =InitializeAutomapper**()**;

var empDTO = mapper.Map**<**EmployeeDTO**>(**emp**)**;

Console.WriteLine**(**"Name:" + empDTO.Name + ", Salary:" + empDTO.Salary + ", Department:" + empDTO.Department**)**;

Console.WriteLine**(**"City:" + empDTO.address.City + ", State:" + empDTO.address.Stae + ", Country:" + empDTO.address.Country**)**;

Console.ReadLine**()**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>** **{**

cfg.CreateMap**<**Employee, EmployeeDTO**>()**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** Address address **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** AddressDTO address **{** **get**; **set**; **}**

**}**

**public** **class** Address

**{**

**public** string City **{** **get**; **set**; **}**

**public** string Stae **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

**public** **class** AddressDTO

**{**

**public** string City **{** **get**; **set**; **}**

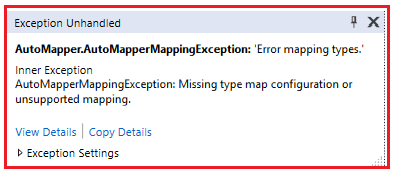
**public** string Stae **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

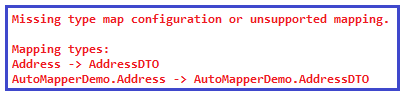
**}**

**}**

Now, when you run the application, you will get the following runtime exception.

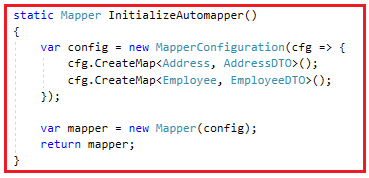


Now, if you go the inner exception and check the message property, then it clearly shows that mapping type configuration is missing for Address and AddresDTO. This is because in our mapper configuration, we have specified the Mapper for Employee and EmployeeDTO but not for Address and AddressDTO type.



**How to solve the above Problem?**

In order to solve the above problem, you need to configure the mapping between the Address and AddressDTO, before the Employee and EmployeeDTO Mapping. So, change the **InitializeAutomapper**method as shown below.

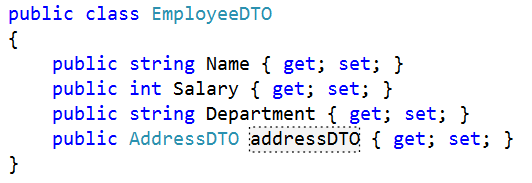


With the above changes in place, now, if you run the application and then you will get the output as expected as shown in the below image.

AutoMapper with Nested Types

**What happen if we changes the complex type property name involved in automapper?**

Let us understand this with an example. Let’s change the complex property address to addressDTO of the EmployeeDTO class as shown below.



**Below is the complete code.**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

Address empAddres = new Address**()**

**{**

City = "Mumbai",

Stae = "Maharashtra",

Country = "India"

**}**;

Employee emp = new Employee

**{**

Name = "James",

Salary = 20000,

Department = "IT",

address = empAddres

**}**;

var mapper =InitializeAutomapper**()**;

var empDTO = mapper.Map**<**EmployeeDTO**>(**emp**)**;

Console.WriteLine**(**"Name:" + empDTO.Name + ", Salary:" + empDTO.Salary + ", Department:" + empDTO.Department**)**;

Console.WriteLine**(**"City:" + empDTO.addressDTO.City + ", State:" + empDTO.addressDTO.Stae + ", Country:" + empDTO.addressDTO.Country**)**;

Console.ReadLine**()**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>** **{**

cfg.CreateMap**<**Address, AddressDTO**>()**;

cfg.CreateMap**<**Employee, EmployeeDTO**>()**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** Address address **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** AddressDTO addressDTO **{** **get**; **set**; **}**

**}**

**public** **class** Address

**{**

**public** string City **{** **get**; **set**; **}**

**public** string Stae **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

**public** **class** AddressDTO

**{**

**public** string City **{** **get**; **set**; **}**

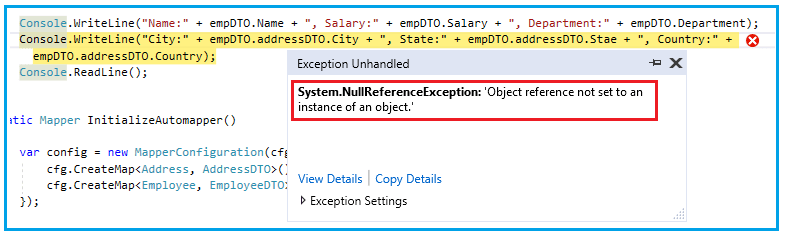
**public** string Stae **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

**}**

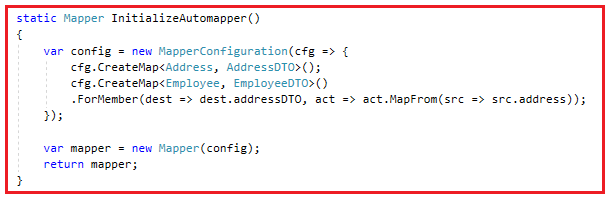
**Now run the application. It should give you the following error.**



This is because the property name for **addressDTO** is not found in the **Employee** object. So we get the null reference error.

**How to solve the above problem?**

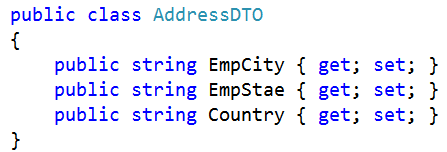
To solve the above issue we need to map the **address** property to **addressDTO** property in the mapper configuration as shown in the below image.



With the above changes in automapper config, now run the application and it will give you the result as expected.

**What happen if the Complex type property names are different?**

Let us understand this with an example. Let’s modify the property names in the **AddressDTO** class as shown below.



**Below is the complete example.**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

Address empAddres = new Address**()**

**{**

City = "Mumbai",

Stae = "Maharashtra",

Country = "India"

**}**;

Employee emp = new Employee

**{**

Name = "James",

Salary = 20000,

Department = "IT",

address = empAddres

**}**;

var mapper =InitializeAutomapper**()**;

var empDTO = mapper.Map**<**EmployeeDTO**>(**emp**)**;

Console.WriteLine**(**"Name:" + empDTO.Name + ", Salary:" + empDTO.Salary + ", Department:" + empDTO.Department**)**;

Console.WriteLine**(**"City:" + empDTO.addressDTO.EmpCity + ", State:" + empDTO.addressDTO.EmpStae + ", Country:" + empDTO.addressDTO.Country**)**;

Console.ReadLine**()**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>** **{**

cfg.CreateMap**<**Address, AddressDTO**>()**;

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

.ForMember**(**dest =**>** dest.addressDTO, act =**>** act.MapFrom**(**src =**>** src.address**))**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** Address address **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** AddressDTO addressDTO **{** **get**; **set**; **}**

**}**

**public** **class** Address

**{**

**public** string City **{** **get**; **set**; **}**

**public** string Stae **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

**public** **class** AddressDTO

**{**

**public** string EmpCity **{** **get**; **set**; **}**

**public** string EmpStae **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

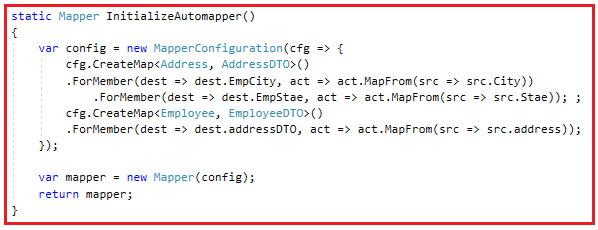
**}**

**}**

When you run the application, it will not give you any error, but it will not map the **City** and **State** property as shown in the below output.

AutoMapper with Nested Types

This is because we mapped the **Address** object with the **AddressDTO** object, but we had not mapped the **City** and **State** properties with **EmpCity** and **EmpState** properties. Let’s map the above two properties and see what happens. To map the above two properties we need to change the **InitializeAutomapper** class as shown below.



**The complete code is given below:**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

Address empAddres = new Address**()**

**{**

City = "Mumbai",

Stae = "Maharashtra",

Country = "India"

**}**;

Employee emp = new Employee

**{**

Name = "James",

Salary = 20000,

Department = "IT",

address = empAddres

**}**;

var mapper =InitializeAutomapper**()**;

var empDTO = mapper.Map**<**EmployeeDTO**>(**emp**)**;

Console.WriteLine**(**"Name:" + empDTO.Name + ", Salary:" + empDTO.Salary + ", Department:" + empDTO.Department**)**;

Console.WriteLine**(**"City:" + empDTO.addressDTO.EmpCity + ", State:" + empDTO.addressDTO.EmpStae + ", Country:" + empDTO.addressDTO.Country**)**;

Console.ReadLine**()**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>** **{**

cfg.CreateMap**<**Address, AddressDTO**>()**

.ForMember**(**dest =**>** dest.EmpCity, act =**>** act.MapFrom**(**src =**>** src.City**))**

.ForMember**(**dest =**>** dest.EmpStae, act =**>** act.MapFrom**(**src =**>** src.Stae**))**; ;

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

.ForMember**(**dest =**>** dest.addressDTO, act =**>** act.MapFrom**(**src =**>** src.address**))**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** Address address **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** AddressDTO addressDTO **{** **get**; **set**; **}**

**}**

**public** **class** Address

**{**

**public** string City **{** **get**; **set**; **}**

**public** string Stae **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

**public** **class** AddressDTO

**{**

**public** string EmpCity **{** **get**; **set**; **}**

**public** string EmpStae **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

**}**

With the above changes in place, now run the application and see the output as expected as shown in the below image.

AutoMapper Complex Mapping in C#

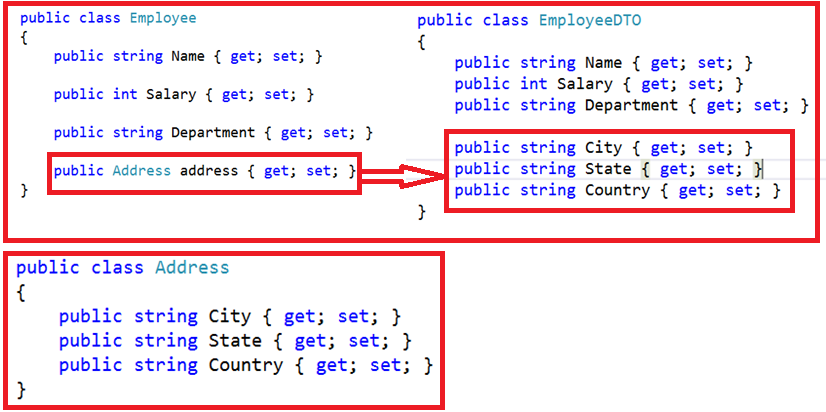
**Mapping Complex type to Primitive Type using AutoMapper in C#**

**Mapping Complex type to Primitive Type using AutoMapper in C#**

In this article, I am going to discuss **how to map Complex type to Primitive Type using AutoMapper in C#** with examples. Please read our previous article where we discussed the [**Automapper Complex Mapping**](https://dotnettutorials.net/lesson/automapper-with-nested-types/)with some examples. At the end of this article, you will understand and when and how to map complex type to primitive type using automapper.

**When to map Complex type to Primitive Type using AutoMapper in C#?**

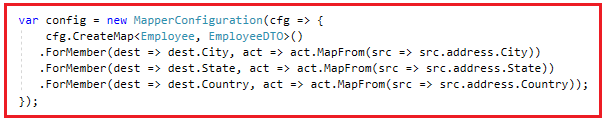
When one class contains primitive types or you can say the simple types and the other class contains complex type involved in the mapping then in such scenarios we need to Map the Complex Type to the Primitive Types. Let us understand how to map Complex type to Primitive Type using AutoMapper in C# with an example. Here in this demo, we are going to use the following three classes (Employee, EmployeeDTO and Address).



As shown in the above image, here, we want to Map the complex property **address** of Employee class to the **City, State and Country** properties of **EmployeeDTO** class.

**How to map Complex type to Primitive Type using AutoMapper in C#?**

In order to map the complex type to the primitive types we need to use the ForMember and need to specify the source and target properties. Here, we need to map the city, state, and Country properties of Address object to the City, state, and Country properties of EmployeeDTO class. To do this you need initializes the Mapper configuration as shown below.



As shown in the above image we mapped each property from the complex type (**address**) of the source object (**Employee**) to the correspondent properties of the destination object (**EmployeeDTO**).

**The complete code of the above example.**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

Address empAddres = new Address**()**

**{**

City = "Mumbai",

State = "Maharashtra",

Country = "India"

**}**;

Employee emp = new Employee**()**;

emp.Name = "James";

emp.Salary = 20000;

emp.Department = "IT";

emp.address = empAddres;

var mapper =InitializeAutomapper**()**;

var empDTO = mapper.Map**<**EmployeeDTO**>(**emp**)**;

Console.WriteLine**(**"Name:" + empDTO.Name + ", Salary:" + empDTO.Salary + ", Department:" + empDTO.Department**)**;

Console.WriteLine**(**"City:" + empDTO.City + ", State:" + empDTO.State + ", Country:" + empDTO.Country**)**;

Console.ReadLine**()**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>** **{**

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

.ForMember**(**dest =**>** dest.City, act =**>** act.MapFrom**(**src =**>** src.address.City**))**

.ForMember**(**dest =**>** dest.State, act =**>** act.MapFrom**(**src =**>** src.address.State**))**

.ForMember**(**dest =**>** dest.Country, act =**>** act.MapFrom**(**src =**>** src.address.Country**))**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** Address address **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** string City **{** **get**; **set**; **}**

**public** string State **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

**public** **class** Address

**{**

**public** string City **{** **get**; **set**; **}**

**public** string State **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

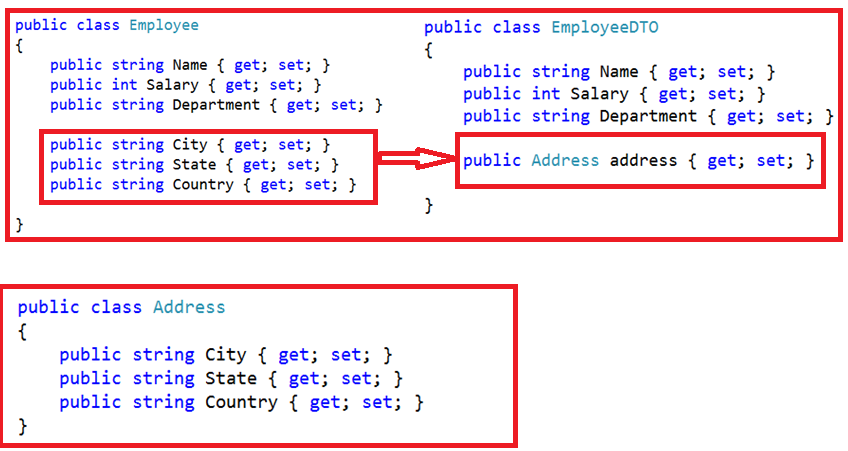
**}**

Now run the application and you will see the output as expected as shown below.

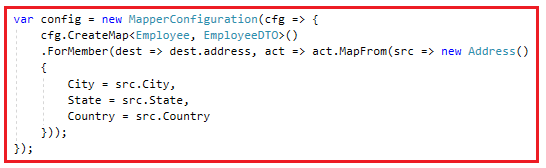
Mapping Complex type to Primitive Type using AutoMapper

**Mapping Primitive properties to a Complex type using Automapper:**

Now we want to map the primitive type properties of the source object to a complex type of the destination object as shown in the image below.



As you can see in the above image, now we want to map the **City, State, and Country** Properties of the source (**Employee**) object to the address property of the destination (**EmployeeDTO**) object. In order to achieve this you need to use **projection in AutoMapper**. So, you need to change the mapper configuration as shown below.



As you can see in the above image, we project an object (**Address type**) using the **MapForm** option and City, State and Country Values are coming from the Source object.

**Below is the complete example**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

Address empAddres = new Address**()**

**{**

City = "Mumbai",

State = "Maharashtra",

Country = "India"

**}**;

Employee emp = new Employee**()**;

emp.Name = "James";

emp.Salary = 20000;

emp.Department = "IT";

emp.City = "Mumbai";

emp.State = "Maharashtra";

emp.Country = "India";

var mapper =InitializeAutomapper**()**;

var empDTO = mapper.Map**<**EmployeeDTO**>(**emp**)**;

Console.WriteLine**(**"Name:" + empDTO.Name + ", Salary:" + empDTO.Salary + ", Department:" + empDTO.Department**)**;

Console.WriteLine**(**"City:" + empDTO.address.City + ", State:" + empDTO.address.State + ", Country:" + empDTO.address.Country**)**;

Console.ReadLine**()**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>** **{**

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

.ForMember**(**dest =**>** dest.address, act =**>** act.MapFrom**(**src =**>** new Address**()**

**{**

City = src.City,

State = src.State,

Country = src.Country

**}))**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Employee

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** string City **{** **get**; **set**; **}**

**public** string State **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** string Name **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** string Department **{** **get**; **set**; **}**

**public** Address address **{** **get**; **set**; **}**

**}**

**public** **class** Address

**{**

**public** string City **{** **get**; **set**; **}**

**public** string State **{** **get**; **set**; **}**

**public** string Country **{** **get**; **set**; **}**

**}**

**}**

When you run the application, it will display the output as expected as shown below.

Mapping Complex type to Primitive Type using AutoMapper

**AutoMapper ReverseMap in C#**

**AutoMapper ReverseMap in C# with Examples**

In this article, I am going to discuss the **AutoMapper ReverseMap in C#** with some examples. Please read our previous article before proceeding to this article where we discussed [**how to map Complex type to Primitive Type using AutoMapper in C#**](https://dotnettutorials.net/lesson/mapping-complex-type-to-primitive-type-using-automapper/) with examples. This is one of the most important concepts to understand in AutoMapper and at the end of this article, you will understand how automapper reverse mapping in c# works.

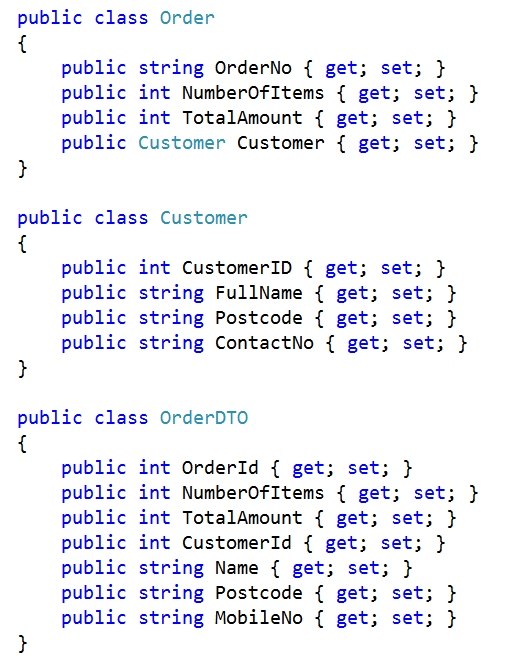
**What is AutoMapper Reverse Mapping in C#?**

The Automapper Reverse Mapping is nothing but the two-way mapping which is also called as bidirectional mapping.

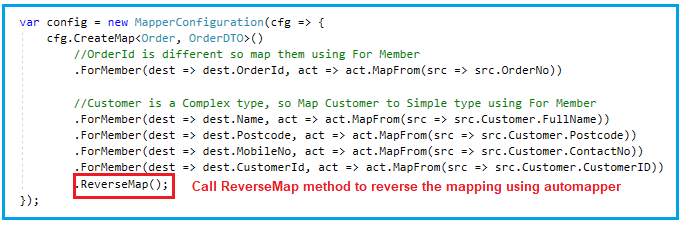
As of now, the mapping we discussed are one directional means if we have two types let’s say Type A and Type B, then we Map Type A with Type B. But using Automapper Reverse mapping it is also possible to Map Type B with Type A.

**Let us understand C# AutoMapper Reverse Mapping with an example.**

We are going to use the following three classes for this demo.



To implement the Reverse Mapping using Automapper in C# we need to use the **ReverseMap** option as shown in the below image.



**Below is the complete example.**

The code self explained. So, please go through the comment lines.

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Step1: Initialize the Mapper

var mapper = InitializeAutomapper**()**;

//Step2: Create the Order Request

var OrderRequest = CreateOrderRequest**()**;

//Step3: Map the OrderRequest object to Order DTO

var orderDTOData = mapper.Map**<**Order, OrderDTO**>(**OrderRequest**)**;

//Step4: Print the OrderDTO Data

Console.WriteLine**(**"After Mapping - OrderDTO Data"**)**;

Console.WriteLine**(**"OrderId : " + orderDTOData.OrderId**)**;

Console.WriteLine**(**"NumberOfItems : " + orderDTOData.NumberOfItems**)**;

Console.WriteLine**(**"TotalAmount : " + orderDTOData.TotalAmount**)**;

Console.WriteLine**(**"CustomerId : " + orderDTOData.CustomerId**)**;

Console.WriteLine**(**"Name : " + orderDTOData.Name**)**;

Console.WriteLine**(**"Postcode : " + orderDTOData.Postcode**)**;

Console.WriteLine**(**"MobileNo : " + orderDTOData.MobileNo**)**;

Console.WriteLine**()**;

//Step5: modify the OrderDTO data

orderDTOData.OrderId = 10;

orderDTOData.NumberOfItems = 20;

orderDTOData.TotalAmount = 2000;

orderDTOData.CustomerId = 5;

orderDTOData.Name = "Smith";

orderDTOData.Postcode = "12345";

//Step6: Reverse Map

mapper.Map**(**orderDTOData, OrderRequest**)**;

//Step7: Print the Order Data

Console.WriteLine**(**"After Reverse Mapping - Order Data"**)**;

Console.WriteLine**(**"OrderNo : " + OrderRequest.OrderNo**)**;

Console.WriteLine**(**"NumberOfItems : " + OrderRequest.NumberOfItems**)**;

Console.WriteLine**(**"TotalAmount : " + OrderRequest.TotalAmount**)**;

Console.WriteLine**(**"CustomerId : " + OrderRequest.Customer.CustomerID**)**;

Console.WriteLine**(**"FullName : " + OrderRequest.Customer.FullName**)**;

Console.WriteLine**(**"Postcode : " + OrderRequest.Customer.Postcode**)**;

Console.WriteLine**(**"ContactNo : " + OrderRequest.Customer.ContactNo**)**;

Console.ReadLine**()**;

**}**

**private** **static** Order CreateOrderRequest**()**

**{**

**return** new Order

**{**

OrderNo = 101,

NumberOfItems = 3,

TotalAmount = 1000,

Customer = new Customer**()**

**{**

CustomerID = 777,

FullName = "James Smith",

Postcode = "755019",

ContactNo = "1234567890"

**}**,

**}**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>** **{**

cfg.CreateMap**<**Order, OrderDTO**>()**

//OrderId is different so map them using For Member

.ForMember**(**dest =**>** dest.OrderId, act =**>** act.MapFrom**(**src =**>** src.OrderNo**))**

//Customer is a Complex type, so Map Customer to Simple type using For Member

.ForMember**(**dest =**>** dest.Name, act =**>** act.MapFrom**(**src =**>** src.Customer.FullName**))**

.ForMember**(**dest =**>** dest.Postcode, act =**>** act.MapFrom**(**src =**>** src.Customer.Postcode**))**

.ForMember**(**dest =**>** dest.MobileNo, act =**>** act.MapFrom**(**src =**>** src.Customer.ContactNo**))**

.ForMember**(**dest =**>** dest.CustomerId, act =**>** act.MapFrom**(**src =**>** src.Customer.CustomerID**))**

.ReverseMap**()**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Order

**{**

**public** **int** OrderNo **{** **get**; **set**; **}**

**public** **int** NumberOfItems **{** **get**; **set**; **}**

**public** **int** TotalAmount **{** **get**; **set**; **}**

**public** Customer Customer **{** **get**; **set**; **}**

**}**

**public** **class** Customer

**{**

**public** **int** CustomerID **{** **get**; **set**; **}**

**public** string FullName **{** **get**; **set**; **}**

**public** string Postcode **{** **get**; **set**; **}**

**public** string ContactNo **{** **get**; **set**; **}**

**}**

**public** **class** OrderDTO

**{**

**public** **int** OrderId **{** **get**; **set**; **}**

**public** **int** NumberOfItems **{** **get**; **set**; **}**

**public** **int** TotalAmount **{** **get**; **set**; **}**

**public** **int** CustomerId **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

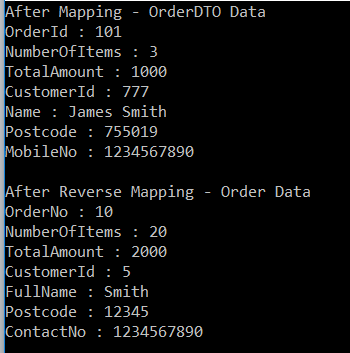
**public** string Postcode **{** **get**; **set**; **}**

**public** string MobileNo **{** **get**; **set**; **}**

**}**

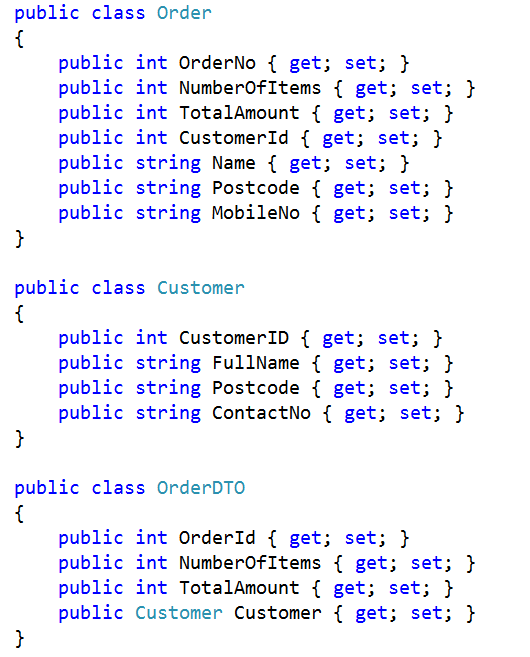
**}**

When you run the application, it displays the data as expected as shown in the below image.

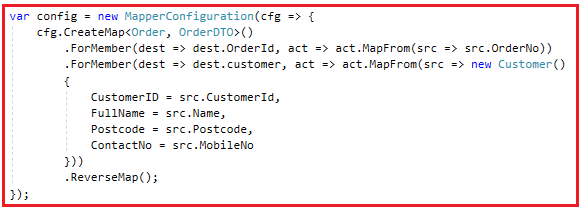


**Modifying Models:**

Let us modify the models as shown below. As you can see in the below image, now the complex type property is present in the **OrderDTO** class and primitive types are present in the **Order** class.



Let’s implement the **AutoMapper ReverseMap()** function and see whether we get the results as expected or not. Modify the Mapper initialization to implement mapping and reverse mapping for the above types as shown in the below image.



**Below is the code for the complete example**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Step1: Initialize the Mapper

var mapper = InitializeAutomapper**()**;

//Step2: Create the Order Request

var OrderRequest = CreateOrderRequest**()**;

//Step3: Map the OrderRequest object to Order DTO

var orderDTOData = mapper.Map**<**Order, OrderDTO**>(**OrderRequest**)**;

//Step4: Print the OrderDTO Data

Console.WriteLine**(**"After Mapping - OrderDTO Data"**)**;

Console.WriteLine**(**"OrderId : " + orderDTOData.OrderId**)**;

Console.WriteLine**(**"NumberOfItems : " + orderDTOData.NumberOfItems**)**;

Console.WriteLine**(**"TotalAmount : " + orderDTOData.TotalAmount**)**;

Console.WriteLine**(**"CustomerId : " + orderDTOData.customer.CustomerID**)**;

Console.WriteLine**(**"FullName : " + orderDTOData.customer.FullName**)**;

Console.WriteLine**(**"Postcode : " + orderDTOData.customer.Postcode**)**;

Console.WriteLine**(**"ContactNo : " + orderDTOData.customer.ContactNo**)**;

Console.WriteLine**()**;

//Step5: modify the OrderDTO data

orderDTOData.OrderId = 10;

orderDTOData.NumberOfItems = 20;

orderDTOData.TotalAmount = 2000;

orderDTOData.customer.CustomerID = 5;

orderDTOData.customer.FullName = "James Smith";

orderDTOData.customer.Postcode = "12345";

//Step6: Reverse Map

mapper.Map**(**orderDTOData, OrderRequest**)**;

//Step7: Print the Order Data

Console.WriteLine**(**"After Reverse Mapping - Order Data"**)**;

Console.WriteLine**(**"OrderNo : " + OrderRequest.OrderNo**)**;

Console.WriteLine**(**"NumberOfItems : " + OrderRequest.NumberOfItems**)**;

Console.WriteLine**(**"TotalAmount : " + OrderRequest.TotalAmount**)**;

Console.WriteLine**(**"CustomerId : " + OrderRequest.CustomerId**)**;

Console.WriteLine**(**"Name : " + OrderRequest.Name**)**;

Console.WriteLine**(**"Postcode : " + OrderRequest.Postcode**)**;

Console.WriteLine**(**"MobileNo : " + OrderRequest.MobileNo**)**;

Console.ReadLine**()**;

**}**

**private** **static** Order CreateOrderRequest**()**

**{**

**return** new Order

**{**

OrderNo = 101,

NumberOfItems = 3,

TotalAmount = 1000,

CustomerId = 777,

Name = "James Smith",

Postcode = "755019",

MobileNo = "1234567890"

**}**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>** **{**

cfg.CreateMap**<**Order, OrderDTO**>()**

.ForMember**(**dest =**>** dest.OrderId, act =**>** act.MapFrom**(**src =**>** src.OrderNo**))**

.ForMember**(**dest =**>** dest.customer, act =**>** act.MapFrom**(**src =**>** new Customer**()**

**{**

CustomerID = src.CustomerId,

FullName = src.Name,

Postcode = src.Postcode,

ContactNo = src.MobileNo

**}))**

.ReverseMap**()**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Order

**{**

**public** **int** OrderNo **{** **get**; **set**; **}**

**public** **int** NumberOfItems **{** **get**; **set**; **}**

**public** **int** TotalAmount **{** **get**; **set**; **}**

**public** **int** CustomerId **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**public** string Postcode **{** **get**; **set**; **}**

**public** string MobileNo **{** **get**; **set**; **}**

**}**

**public** **class** Customer

**{**

**public** **int** CustomerID **{** **get**; **set**; **}**

**public** string FullName **{** **get**; **set**; **}**

**public** string Postcode **{** **get**; **set**; **}**

**public** string ContactNo **{** **get**; **set**; **}**

**}**

**public** **class** OrderDTO

**{**

**public** **int** OrderId **{** **get**; **set**; **}**

**public** **int** NumberOfItems **{** **get**; **set**; **}**

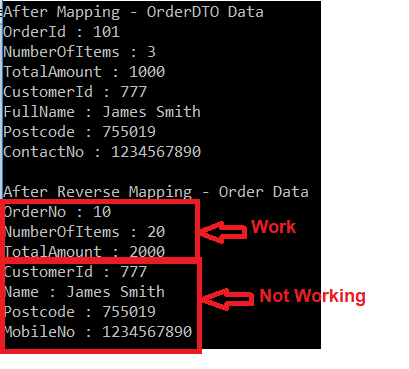
**public** **int** TotalAmount **{** **get**; **set**; **}**

**public** Customer customer **{** **get**; **set**; **}**

**}**

**}**

**Now run the application and see the output.**

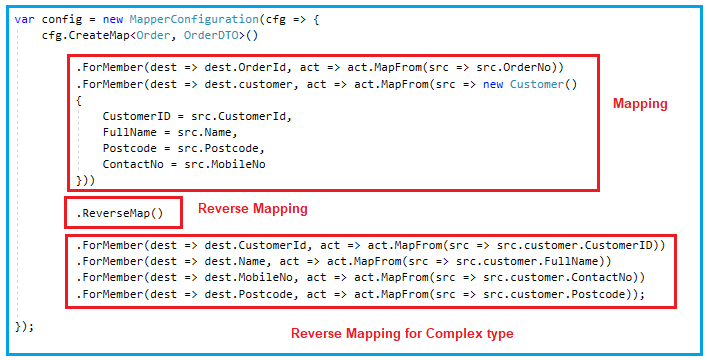


As you can see from the above output, the automapper reverse mapping work as expected for the primitive types but not working for the complex type.

So when both classes have members that are named equally then the **AutoMapper ReverseMap()** function works as expected. But if the classes contain members that are different and not mapped via the default (per naming) mapping then this does not work as expected.

**How to make the two mapping works as expected?**

If you want to make the two-way mapping work as expected then you need to do the Mapping via ForMember. So modify the Mapper initialization to tell the mapping for the complex type as shown in the below image.



**Below is the complete code.**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Step1: Initialize the Mapper

var mapper = InitializeAutomapper**()**;

//Step2: Create the Order Request

var OrderRequest = CreateOrderRequest**()**;

//Step3: Map the OrderRequest object to Order DTO

var orderDTOData = mapper.Map**<**Order, OrderDTO**>(**OrderRequest**)**;

//Step4: Print the OrderDTO Data

Console.WriteLine**(**"After Mapping - OrderDTO Data"**)**;

Console.WriteLine**(**"OrderId : " + orderDTOData.OrderId**)**;

Console.WriteLine**(**"NumberOfItems : " + orderDTOData.NumberOfItems**)**;

Console.WriteLine**(**"TotalAmount : " + orderDTOData.TotalAmount**)**;

Console.WriteLine**(**"CustomerId : " + orderDTOData.customer.CustomerID**)**;

Console.WriteLine**(**"FullName : " + orderDTOData.customer.FullName**)**;

Console.WriteLine**(**"Postcode : " + orderDTOData.customer.Postcode**)**;

Console.WriteLine**(**"ContactNo : " + orderDTOData.customer.ContactNo**)**;

Console.WriteLine**()**;

//Step5: modify the OrderDTO data

orderDTOData.OrderId = 10;

orderDTOData.NumberOfItems = 20;

orderDTOData.TotalAmount = 2000;

orderDTOData.customer.CustomerID = 5;

orderDTOData.customer.FullName = "James Smith";

orderDTOData.customer.Postcode = "12345";

//Step6: Reverse Map

mapper.Map**(**orderDTOData, OrderRequest**)**;

//Step7: Print the Order Data

Console.WriteLine**(**"After Reverse Mapping - Order Data"**)**;

Console.WriteLine**(**"OrderNo : " + OrderRequest.OrderNo**)**;

Console.WriteLine**(**"NumberOfItems : " + OrderRequest.NumberOfItems**)**;

Console.WriteLine**(**"TotalAmount : " + OrderRequest.TotalAmount**)**;

Console.WriteLine**(**"CustomerId : " + OrderRequest.CustomerId**)**;

Console.WriteLine**(**"Name : " + OrderRequest.Name**)**;

Console.WriteLine**(**"Postcode : " + OrderRequest.Postcode**)**;

Console.WriteLine**(**"MobileNo : " + OrderRequest.MobileNo**)**;

Console.ReadLine**()**;

**}**

**private** **static** Order CreateOrderRequest**()**

**{**

**return** new Order

**{**

OrderNo = 101,

NumberOfItems = 3,

TotalAmount = 1000,

CustomerId = 777,

Name = "James Smith",

Postcode = "755019",

MobileNo = "1234567890"

**}**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>** **{**

cfg.CreateMap**<**Order, OrderDTO**>()**

.ForMember**(**dest =**>** dest.OrderId, act =**>** act.MapFrom**(**src =**>** src.OrderNo**))**

.ForMember**(**dest =**>** dest.customer, act =**>** act.MapFrom**(**src =**>** new Customer**()**

**{**

CustomerID = src.CustomerId,

FullName = src.Name,

Postcode = src.Postcode,

ContactNo = src.MobileNo

**}))**

.ReverseMap**()**

.ForMember**(**dest =**>** dest.CustomerId, act =**>** act.MapFrom**(**src =**>** src.customer.CustomerID**))**

.ForMember**(**dest =**>** dest.Name, act =**>** act.MapFrom**(**src =**>** src.customer.FullName**))**

.ForMember**(**dest =**>** dest.MobileNo, act =**>** act.MapFrom**(**src =**>** src.customer.ContactNo**))**

.ForMember**(**dest =**>** dest.Postcode, act =**>** act.MapFrom**(**src =**>** src.customer.Postcode**))**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Order

**{**

**public** **int** OrderNo **{** **get**; **set**; **}**

**public** **int** NumberOfItems **{** **get**; **set**; **}**

**public** **int** TotalAmount **{** **get**; **set**; **}**

**public** **int** CustomerId **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**public** string Postcode **{** **get**; **set**; **}**

**public** string MobileNo **{** **get**; **set**; **}**

**}**

**public** **class** Customer

**{**

**public** **int** CustomerID **{** **get**; **set**; **}**

**public** string FullName **{** **get**; **set**; **}**

**public** string Postcode **{** **get**; **set**; **}**

**public** string ContactNo **{** **get**; **set**; **}**

**}**

**public** **class** OrderDTO

**{**

**public** **int** OrderId **{** **get**; **set**; **}**

**public** **int** NumberOfItems **{** **get**; **set**; **}**

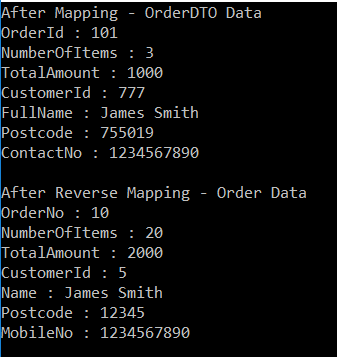
**public** **int** TotalAmount **{** **get**; **set**; **}**

**public** Customer customer **{** **get**; **set**; **}**

**}**

**}**

Now run the application and you will see the output as expected as shown in the below image.



**AutoMapper Conditional Mapping in C#**

**AutoMapper Conditional Mapping in C# with Examples**

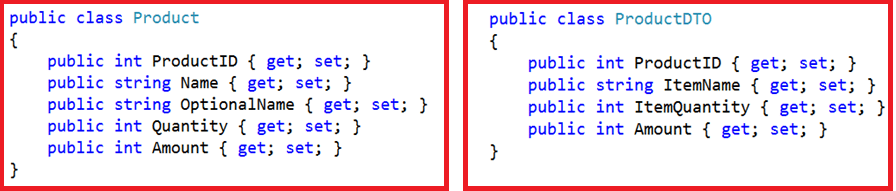
In this article, I am going to discuss the **AutoMapper Conditional Mapping in C#** with examples**.** Please read our previous article where we discussed the [**Reverse Mapping Using AutoMapper in C#**](https://dotnettutorials.net/lesson/reverse-mapping-using-automapper/)with examples. At the end of this article, you will understand what is AutoMapper Conditional Mapping and when and how to use conditional mapping in C#.

**What is AutoMapper Conditional Mapping?**

The [**AutoMapper in C#**](https://dotnettutorials.net/lesson/automapper-in-c-sharp/) allows us to add conditions to the properties of the source object that must be met before that property going to be mapped to the property of the destination object.  For example, if we want to map a property only if its value is greater than 0, then in such a situation we need to use C# **AutoMapper Conditional Mapping**.

**Example to understand Conditional Mapping in AutoMapper**

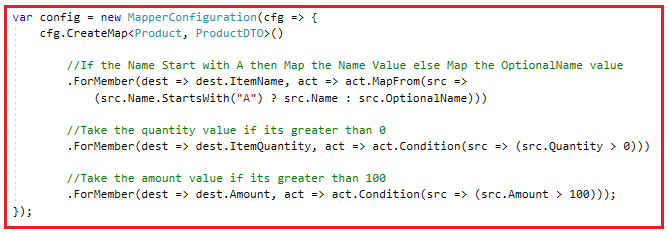
Let us understand Conditional Mapping using automapper with an example. We are going to use the following two classes in this demo.



**Business Requirement:**

1. We need to Map the **Name** property of the **Product** class to the **itemName** property of the **ProductDTO** class only if the Name value starts with the letter “**A**”, else Map the **OptionalName** property value of the **Product** class with the **ItemName** property of the **ProductDTO** class.
2. If the **Quantity** value is greater than **0** then only map it to the **ItemQuantity**
3. Similarly, if the **Amount** value is greater than 100 then only map it to the **Amount** property of the ProductDTO class.

To achieve this we need to use the AutoMapper Conditional Mapping. So Initializes the Mapper configuration as shown below to use the C# AutoMapper Conditional Mapping.



The **Condition** option in AutoMapper is used to add conditions to the properties that must be met before that property is going to map. The **MapFrom** option is used to perform the custom source and destination member mappings.

**Below is the complete code.**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

var mapper = InitializeAutomapper**()**;

Product product = new Product**()**

**{**

ProductID = 101,

Name = "Led TV",

OptionalName = "Product name not start with A",

Quantity = -5,

Amount = 1000

**}**;

var productDTO = mapper.Map**<**Product, ProductDTO**>(**product**)**;

Console.WriteLine**(**"After Mapping : Product"**)**;

Console.WriteLine**(**"ProductID : " + product.ProductID**)**;

Console.WriteLine**(**"Name : " + product.Name**)**;

Console.WriteLine**(**"OptionalName : " + product.OptionalName**)**;

Console.WriteLine**(**"Quantity : " + product.Quantity**)**;

Console.WriteLine**(**"Amount : " + product.Amount**)**;

Console.WriteLine**()**;

Console.WriteLine**(**"After Mapping : ProductDTO"**)**;

Console.WriteLine**(**"ProductID : " + productDTO.ProductID**)**;

Console.WriteLine**(**"ItemName : " + productDTO.ItemName**)**;

Console.WriteLine**(**"ItemQuantity : " + productDTO.ItemQuantity**)**;

Console.WriteLine**(**"Amount : " + productDTO.Amount**)**;

Console.ReadLine**()**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>**

**{**

cfg.CreateMap**<**Product, ProductDTO**>()**

//If the Name Start with A then Map the Name Value else Map the OptionalName value

.ForMember**(**dest =**>** dest.ItemName, act =**>** act.MapFrom**(**src =**>**

**(**src.Name.StartsWith**(**"A"**)** ? src.Name : src.OptionalName**)))**

//Take the quantity value if its greater than 0

.ForMember**(**dest =**>** dest.ItemQuantity, act =**>** act.Condition**(**src =**>** **(**src.Quantity **>** 0**)))**

//Take the amount value if its greater than 100

.ForMember**(**dest =**>** dest.Amount, act =**>** act.Condition**(**src =**>** **(**src.Amount **>** 100**)))**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Product

**{**

**public** **int** ProductID **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**public** string OptionalName **{** **get**; **set**; **}**

**public** **int** Quantity **{** **get**; **set**; **}**

**public** **int** Amount **{** **get**; **set**; **}**

**}**

**public** **class** ProductDTO

**{**

**public** **int** ProductID **{** **get**; **set**; **}**

**public** string ItemName **{** **get**; **set**; **}**

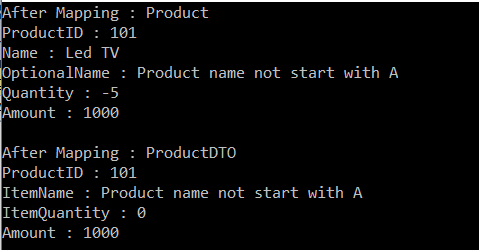
**public** **int** ItemQuantity **{** **get**; **set**; **}**

**public** **int** Amount **{** **get**; **set**; **}**

**}**

**}**

**When you run the application, it displays the following output**



**Note:** The C# AutoMapper Conditional Mapping is one of the important concepts of Automapper which is used in most of the real-time projects.

**AutoMapper Ignore Property in C#**

**AutoMapper Ignore Property in C# with Examples**

In this article, I am going to discuss how to use the **AutoMapper Ignore Property in C#** with examples. Please read our previous article where we discussed the [**AutoMapper Conditional Mapping in C#**](https://dotnettutorials.net/lesson/automapper-conditional-mapping-dotnet/) with some examples**.**At the end of this article, you will understand the need and use of AutoMapper Ignore Property in C# in detail.

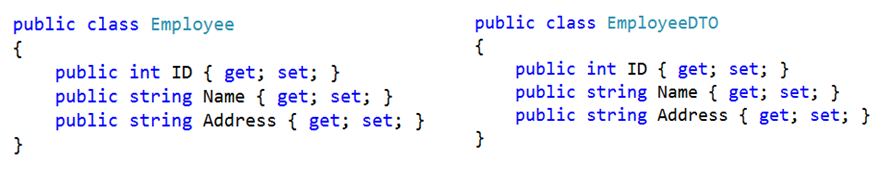
**Why do we need AutoMapper Ignore Property in C#?**

By default, AutoMapper tries to map all the properties from the source type to the destination type when both source and destination type property names are same. If you want some of the properties not to map with the destination type property then you need to use the **AutoMapper Ignore Property** in C#.

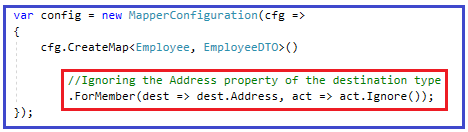
**Note:** If some of the properties are not available in the destination type, then the AutoMapper will not throw any exception when doing the mapping. However, it will throw an exception when you are using the **ValidateMapperConfiguration()** which we will discuss in our upcoming articles.

**Example to understand AutoMapper Ignore Property in C#**

Let us understand how to use the AutoMapper Ignore Property with an example. We are going to use the below **Employee** and **EmployeeDTO** classes for this demo.



Our Business Requirement is not to map the **Address Property** i.e. we need to **ignore the Address property** while doing the mapping between these two objects. To do so we need to use the **Ignore Property** with the **Address** property of the destination type while doing the mapper configuration as shown below in the below image.



As you can see in the above example, we have added the ForMember method to ignore the Address property while doing the mapping.

**Below is the complete example.**

**using** *System;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

var mapper = InitializeAutomapper**()**;

Employee employee = new Employee**()**

**{**

ID = 101,

Name = "James",

Address = "Mumbai"

**}**;

var empDTO = mapper.Map**<**Employee, EmployeeDTO**>(**employee**)**;

Console.WriteLine**(**"After Mapping : Employee"**)**;

Console.WriteLine**(**"ID : " + employee.ID + ", Name : " + employee.Name + ", Address : " + employee.Address**)**;

Console.WriteLine**()**;

Console.WriteLine**(**"After Mapping : EmployeeDTO"**)**;

Console.WriteLine**(**"ID : " + empDTO.ID + ", Name : " + empDTO.Name + ", Address : " + empDTO.Address**)**;

Console.ReadLine**()**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>**

**{**

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

//Ignoring the Address property of the destination type

.ForMember**(**dest =**>** dest.Address, act =**>** act.Ignore**())**;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** Employee

**{**

**public** **int** ID **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** **int** ID **{** **get**; **set**; **}**

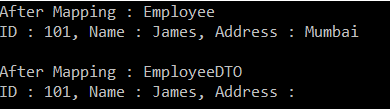
**public** string Name **{** **get**; **set**; **}**

**public** string Address **{** **get**; **set**; **}**

**}**

**}**

**When we run the application, it will give us the below output.**



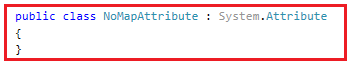
If you look at the output window you can see that the value for the Address property is empty even though the Address property for the Source type has value. So, the **AutoMapper Ignore()** method is used when you want to completely ignore the property in the mapping. The ignored property could be in either the source or the destination object.

**Best way to ignore multiple properties:**

But, it will be a tedious procedure if you want to ignore multiple properties from mapping. If that is your requirement then do consider creating an extension class which will ignore the properties based on data attribute specified in the model. Let us discuss the step by step procedure to implement this.

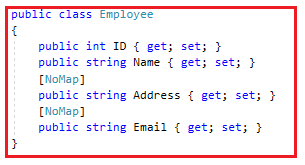
**Step1: Create a NoMap Attribute**

Create a class with the name NoMapAttribute by inheriting from the System.Attribute class so that we can use this class as an attribute.



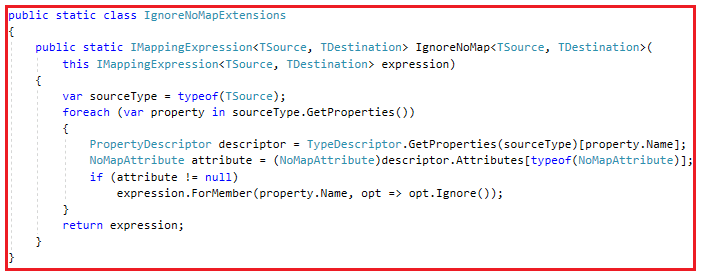
**Step2: Decorate the properties with the NoMap attribute**

Now, you need to decorate the **NoMap** attribute with the source type properties which you don’t want to map with the destination type. For example, if you don’t want to map the Address and Email property, then you need to decorate the NoMap attribute with Address and Email property of the Employee class as shown in the below image.



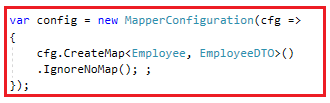
**Step3: Creating an extension method**

Create an extension class as shown below which will ignore the properties which will decorate with the **NoMap** attribute. Here, you need to import the **System.ComponentModel** namespace.



**Step4: Using the IgnoreNoMap extension method**

Finally you need to use the IgnoreNoMap method while defining the Mapping as shown below.



**Below is the complete code.**

**using** *System;*

**using** *System.ComponentModel;*

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

var mapper = InitializeAutomapper**()**;

Employee employee = new Employee**()**

**{**

ID = 101,

Name = "James",

Address = "Mumbai"

**}**;

var empDTO = mapper.Map**<**Employee, EmployeeDTO**>(**employee**)**;

Console.WriteLine**(**"After Mapping : Employee"**)**;

Console.WriteLine**(**"ID : " + employee.ID + ", Name : " + employee.Name + ", Address : " + employee.Address + ", Email : " + employee.Email**)**;

Console.WriteLine**()**;

Console.WriteLine**(**"After Mapping : EmployeeDTO"**)**;

Console.WriteLine**(**"ID : " + empDTO.ID + ", Name : " + empDTO.Name + ", Address : " + empDTO.Address + ", Email : " + empDTO.Email**)**;

Console.ReadLine**()**;

**}**

**static** Mapper InitializeAutomapper**()**

**{**

var config = new MapperConfiguration**(**cfg =**>**

**{**

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

.IgnoreNoMap**()**; ;

**})**;

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**

**}**

**public** **class** NoMapAttribute : System.Attribute

**{**

**}**

**public** **static** **class** IgnoreNoMapExtensions

**{**

**public** **static** IMappingExpression**<**TSource, TDestination**>** IgnoreNoMap**<**TSource, TDestination**>(**

this IMappingExpression**<**TSource, TDestination**>** expression**)**

**{**

var sourceType = typeof**(**TSource**)**;

**foreach** **(**var property in sourceType.GetProperties**())**

**{**

PropertyDescriptor descriptor = TypeDescriptor.GetProperties**(**sourceType**)[**property.Name**]**;

NoMapAttribute attribute = **(**NoMapAttribute**)**descriptor.Attributes**[**typeof**(**NoMapAttribute**)]**;

**if** **(**attribute != **null)**

expression.ForMember**(**property.Name, opt =**>** opt.Ignore**())**;

**}**

**return** expression;

**}**

**}**

**public** **class** Employee

**{**

**public** **int** ID **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**[**NoMap**]**

**public** string Address **{** **get**; **set**; **}**

**[**NoMap**]**

**public** string Email **{** **get**; **set**; **}**

**}**

**public** **class** EmployeeDTO

**{**

**public** **int** ID **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

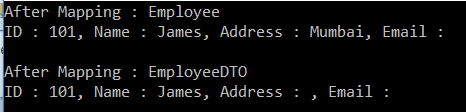
**public** string Address **{** **get**; **set**; **}**

**public** string Email **{** **get**; **set**; **}**

**}**

**}**

**When we run the application, it will display the following result**



In the above example, we have created a custom attribute with the name **NoMap** by inheriting from the **Attribute** class. Then we decorated the properties which need to be ignored from the source type with the newly created **NoMap** attribute. Within the extensions method, we just checked whether a property has this **NoMap** attribute or not. If the property is decorated with the **NoMap** attribute then we added that property to the ignored list.

**UseValue ResolveUsing and Null Substitution in AutoMapper**

**UseValue ResolveUsing and Null Substitution in AutoMapper**

In this article, I am going to discuss the **UseValue ResolveUsing and Null Substitution in AutoMapper** with examples. Please read our previous article where we discussed the [**AutoMapper Ignore Property in C#**](https://dotnettutorials.net/lesson/ignore-using-automapper-in-csharp/) with some examples.

**Understanding the AutoMapper** **UseValue() and ResolveUsing() methods**

The AutoMapper UseValue() method is used to retrieve a value on the first-run and stores it in the mapping whereas the ResolveUsing() method resolves at run-time.

**Let us understand the above AutoMapper UseValue() and ResolveUsing() methods with an example**

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

InitializeAutomapper**()**;

A aObj = new A**()**

**{**

Name = "Pranaya",

AAddress = "Mumbai"

**}**;

var bObj = Mapper.Map**<**A, B**>(**aObj**)**;

Console.WriteLine**(**"After Mapping : "**)**;

//Here FixedValue and DOJ will be empty for aObj

Console.WriteLine**(**"aObj.Member : " + aObj.Name + ", aObj.FixedValue : " + aObj.FixedValue + ", aObj.DOJ : " + aObj.DOJ + ", aObj.AAddress : " + aObj.AAddress**)**;

Console.WriteLine**(**"bObj.Member : " + bObj.Name + ", bObj.FixedValue : " + bObj.FixedValue + ", bObj.DOJ : " + bObj.DOJ + ", bObj.BAddress : " + bObj.BAddress**)**;

bObj.Name = "Rout";

bObj.BAddress = "Delhi";

Mapper.Map**(**bObj, aObj**)**;

Console.WriteLine**(**"After ReverseMap : "**)**;

Console.WriteLine**(**"aObj.Member : " + aObj.Name + ", aObj.FixedValue : " + aObj.FixedValue + ", aObj.DOJ : " + aObj.DOJ + ", aObj.AAddress : " + aObj.AAddress**)**;

Console.WriteLine**(**"bObj.Member : " + bObj.Name + ", bObj.FixedValue : " + bObj.FixedValue + ", bObj.DOJ : " + bObj.DOJ + ", bObj.BAddress : " + bObj.BAddress**)**;

Console.ReadLine**()**;

**}**

**static** **void** InitializeAutomapper**()**

**{**

Mapper.Initialize**(**config =**>**

**{**

config.CreateMap**<**A, B**>()**

.ForMember**(**dest =**>** dest.BAddress, act =**>** act.MapFrom**(**src =**>** src.AAddress**))**

//To Store Static Value use the UseValue() method

.ForMember**(**dest =**>** dest.FixedValue, act =**>** act.UseValue**(**"Hello"**))**

//To Store DateTime value use ResolveUsing() method

.ForMember**(**dest =**>** dest.DOJ, act =**>** act.ResolveUsing**(**src =**>**

**{**

**return** DateTime.Now;

**}))**

.ReverseMap**()**;

**})**;

**}**

**}**

**public** **class** A

**{**

**public** string Name **{** **get**; **set**; **}**

**public** string AAddress **{** **get**; **set**; **}**

**public** string FixedValue **{** **get**; **set**; **}**

**public** DateTime DOJ **{** **get**; **set**; **}**

**}**

**public** **class** B

**{**

**public** string Name **{** **get**; **set**; **}**

**public** string BAddress **{** **get**; **set**; **}**

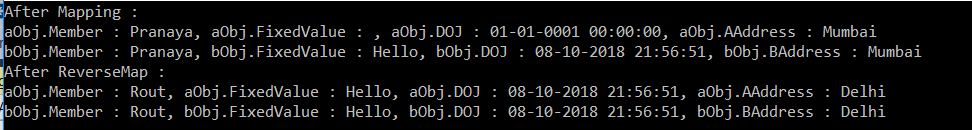
**public** string FixedValue **{** **get**; **set**; **}**

**public** DateTime DOJ **{** **get**; **set**; **}**

**}**

**}**

When you run the application, it gives you the below output.



**Note:** Use the **MapFrom(s => s.MemberName)** when you return the actual source object member whereas you need to use the **UseValue()** method if you want to map a static value in the mapping, and you know the value itself won’t change.

Use **ResolveUsing(s => {})** when you want to resolve a destination field from a derived value. This should be used for any DateTime mappings and any more complicated mapping functions.

**Null Substitution in Automapper:**

The Null substitution allows us to supply an alternate value for a destination member if the source value is null. That means instead of mapping the null value from the source object, it will map from the value we supply. We need to use the **NullSubstitute()** method to substitute the null value using AutoMapper.

Let us understand how to use Null Substitution using AutoMapper with an example

**using** *AutoMapper;*

**namespace** *AutoMapperDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

InitializeAutomapper**()**;

A aObj = new A**()**

**{**

Name = "Pranaya",

AAddress = **null**

**}**;

var bObj = Mapper.Map**<**A, B**>(**aObj**)**;

Console.WriteLine**(**"After Mapping : "**)**;

//Here FixedValue and DOJ will be empty for aObj

Console.WriteLine**(**"aObj.Member : " + aObj.Name + ", aObj.FixedValue : " + aObj.FixedValue + ", aObj.AAddress : " + aObj.AAddress**)**;

Console.WriteLine**(**"bObj.Member : " + bObj.Name + ", bObj.FixedValue : " + bObj.FixedValue + ", bObj.BAddress : " + bObj.BAddress**)**;

Console.ReadLine**()**;

**}**

**static** **void** InitializeAutomapper**()**

**{**

Mapper.Initialize**(**config =**>**

**{**

config.CreateMap**<**A, B**>()**

.ForMember**(**dest =**>** dest.BAddress, act =**>** act.MapFrom**(**src =**>** src.AAddress**))**

//You need to use NullSubstitute method to substitute null value

.ForMember**(**dest =**>** dest.FixedValue, act =**>** act.NullSubstitute**(**"Hello"**))**

.ForMember**(**dest =**>** dest.BAddress, act =**>** act.NullSubstitute**(**"N/A"**))**;

**})**;

**}**

**}**

**public** **class** A

**{**

**public** string Name **{** **get**; **set**; **}**

**public** string AAddress **{** **get**; **set**; **}**

**public** string FixedValue **{** **get**; **set**; **}**

**}**

**public** **class** B

**{**

**public** string Name **{** **get**; **set**; **}**

**public** string BAddress **{** **get**; **set**; **}**

**public** string FixedValue **{** **get**; **set**; **}**

**}**

**}**

**When we run the application, it will give us the below output.**

