Once you have your types, and a reference to AutoMapper, you can create a map for the two types.

Mapper.CreateMap<Order, OrderDto>();

The type on the left is the source type, and the type on the right is the destination type. To perform a mapping, use the Map method.

OrderDto dto = Mapper.Map<Order, OrderDto>(order);

To test your mappings, you need to create a test that does two things:

* Call your bootstrapper class to create all the mappings
* Call Mapper.AssertConfigurationIsValid

Here's an example:

AutoMapperConfiguration.Configure();

Mapper.AssertConfigurationIsValid();

**Projection:**

public class CalendarEvent

{

public DateTime EventDate { get; set; }

public string Title { get; set; }

}

public class CalendarEventForm

{

public DateTime EventDate { get; set; }

public int EventHour { get; set; }

public int EventMinute { get; set; }

public string Title { get; set; }

}

// Configure AutoMapper

Mapper.CreateMap<CalendarEvent, CalendarEventForm>()

.ForMember(dest => dest.EventDate, opt => opt.MapFrom(src => src.EventDate.Date))

.ForMember(dest => dest.EventHour, opt => opt.MapFrom(src => src.EventDate.Hour))

.ForMember(dest => dest.EventMinute, opt => opt.MapFrom(src => src.EventDate.Minute));

// Perform mapping

CalendarEventForm form = Mapper.Map<CalendarEvent, CalendarEventForm>(calendarEvent);

**Converter:**

public class Source

{

public string Value1 { get; set; }

public string Value2 { get; set; }

public string Value3 { get; set; }

}

public class Destination

{

public int Value1 { get; set; }

public DateTime Value2 { get; set; }

public Type Value3 { get; set; }

}

[Test]

public void Example()

{

Mapper.CreateMap<string, int>().ConvertUsing(Convert.ToInt32);

Mapper.CreateMap<string, DateTime>().ConvertUsing(new DateTimeTypeConverter());

Mapper.CreateMap<string, Type>().ConvertUsing<TypeTypeConverter>();

Mapper.CreateMap<Source, Destination>();

Mapper.AssertConfigurationIsValid();

var source = new Source

{

Value1 = "5",

Value2 = "01/01/2000",

Value3 = "AutoMapperSamples.GlobalTypeConverters.GlobalTypeConverters+Destination"

};

Destination result = Mapper.Map<Source, Destination>(source);

result.Value3.ShouldEqual(typeof (Destination));

}

public class DateTimeTypeConverter : ITypeConverter<string, DateTime>

{

public DateTime Convert(ResolutionContext context)

{

return System.Convert.ToDateTime(context.SourceValue);

}

}

public class TypeTypeConverter : ITypeConverter<string, Type>

{

public Type Convert(ResolutionContext context)

{

return context.SourceType;

}

}

**Resolver:**

public class Source

{

public int Value1 { get; set; }

public int Value2 { get; set; }

}

public class Destination

{

public int Total { get; set; }

}

public interface IValueResolver

{

ResolutionResult Resolve(ResolutionResult source);

}

public class CustomResolver : ValueResolver<Source, int>

{

protected override int ResolveCore(Source source)

{

return source.Value1 + source.Value2;

}

}

Mapper.CreateMap<Source, Destination>()

.ForMember(dest => dest.Total, opt => opt.ResolveUsing<CustomResolver>());

Mapper.AssertConfigurationIsValid();

var source = new Source

{

Value1 = 5,

Value2 = 7

};

var result = Mapper.Map<Source, Destination>(source);

result.Total.ShouldEqual(12);

**Inheritance:**

Mapper.CreateMap<Order, OrderDto>()

.Include<OnlineOrder, OnlineOrderDto>()

.Include<MailOrder, MailOrderDto>()

.ForMember(o=>o.Id, m=>m.MapFrom(s=>s.OrderId));

Mapper.CreateMap<OnlineOrder, OnlineOrderDto>();

Mapper.CreateMap<MailOrder, MailOrderDto>();