# Create Data Transfer Objects (DTOs)

06/16/2014 • 2 minutes to read • 👤👕👤👤👤 +2

by [Mike Wasson](#)

[Download Completed Project](#)

Right now, our web API exposes the database entities to the client. The client receives data that maps directly to your database tables. However, that's not always a good idea. Sometimes you want to change the shape of the data that you send to client. For example, you might want to:

- Remove circular references (see previous section).
- 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.
- Avoid "over-posting" vulnerabilities. (See [Model Validation](#) for a discussion of over-posting.)
- Decouple your service layer from your database layer.

To accomplish this, you can define a *data transfer object* (DTO). A DTO is an object that defines how the data will be sent over the network. Let's see how that works with the Book entity. In the Models folder, add two DTO classes:

| C# | 🗐 Copy |
|---|---|

```csharp
namespace BookService.Models
{
    public class BookDto
    {
        public int Id { get; set; }
        public string Title { get; set; }
```

```csharp
        public string AuthorName { get; set; }
    }
}

namespace BookService.Models
{
    public class BookDetailDto
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public int Year { get; set; }
        public decimal Price { get; set; }
        public string AuthorName { get; set; }
        public string Genre { get; set; }
    }
}
```

The `BookDetailDto` class includes all of the properties from the Book model, except that `AuthorName` is a string that will hold the author name. The `BookDto` class contains a subset of properties from `BookDetailDto`.

Next, replace the two GET methods in the `BooksController` class, with versions that return DTOs. We'll use the LINQ **Select** statement to convert from Book entities into DTOs.

| C# | Copy |
| --- | --- |

```csharp
// GET api/Books
public IQueryable<BookDto> GetBooks()
{
    var books = from b in db.Books
                select new BookDto()
                {
                    Id = b.Id,
                    Title = b.Title,
                    AuthorName = b.Author.Name
                };
```

```csharp
        return books;
    }

    // GET api/Books/5
    [ResponseType(typeof(BookDetailDto))]
    public async Task<IHttpActionResult> GetBook(int id)
    {
        var book = await db.Books.Include(b => b.Author).Select(b =>
            new BookDetailDto()
            {
                Id = b.Id,
                Title = b.Title,
                Year = b.Year,
                Price = b.Price,
                AuthorName = b.Author.Name,
                Genre = b.Genre
            }).SingleOrDefaultAsync(b => b.Id == id);
        if (book == null)
        {
            return NotFound();
        }

        return Ok(book);
    }
```

Here is the SQL generated by the new `GetBooks` method. You can see that EF translates the LINQ **Select** into a SQL SELECT statement.

```sql
SQL                                                                    📋 Copy

SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Title] AS [Title],
    [Extent2].[Name] AS [Name]
    FROM  [dbo].[Books] AS [Extent1]
    INNER JOIN [dbo].[Authors] AS [Extent2] ON [Extent1].[AuthorId] = [Extent2].[Id]
```

Finally, modify the `PostBook` method to return a DTO.

C#                                                                            🗐 Copy

```csharp
[ResponseType(typeof(BookDto))]
public async Task<IHttpActionResult> PostBook(Book book)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    db.Books.Add(book);
    await db.SaveChangesAsync();

    // New code:
    // Load author name
    db.Entry(book).Reference(x => x.Author).Load();

    var dto = new BookDto()
    {
        Id = book.Id,
        Title = book.Title,
        AuthorName = book.Author.Name
    };

    return CreatedAtRoute("DefaultApi", new { id = book.Id }, dto);
}
```

ⓘ **Note**

In this tutorial, we're converting to DTOs manually in code. Another option is to use a library like **AutoMapper** that handles the conversion automatically.

Previous     Next

---

## Is this page helpful?

👍 Yes   👎 No

---