# Simplifying EDM with OData

## Summary

In a previous article, I talked about how you can leverage the power of OData with your existing ASP.NET Core API to bring in more features to your API consumers.

But there are different ways you could enable OData on your existing API that are just as simple but offers more powerful features than overriding your existing routes and enabling dependency injection.

For instance, if you've tried to perform a count operation using our previous method you will notice it doesn't really return or perform anything, the same thing goes with many other features that we will talk about extensively in future articles.

In this article, however, I'm going to show you how you can enable OData on your existing ASP.NET Core API using EDM.

## What is EDM?

EDM is short for Entity Data Model, it plays the role of a mapper between whatever data source and format you have and the OData engine.

In other words, whether your source of data is SQL, Cosmos DB or just plain text files, and whether your format is XML, Json or raw text or any other type out there, What the entity data model does is to turn that raw data into entities that allow functionality like count, select, filter and expand to be performed seamlessly through your API.

## Setting Things up

Let's set our existing API up with OData using EDM.

First and foremost, add in Microsoft.AspNetCore.OData nuget package to your ASP.NET Core project.

Once the nuget package is installed, let’s setup the configuration to utilize that package.

In your *Startup.cs* file, in your *ConfigureServices* function, add in the following line:

```
    services.AddOData();
```

**Important Note:** This will work with ASP.NET Core 2.1, if you are trying to set this up with ASP.NET Core 2.2, then you must add another line of code as follows:

```
1
    services.AddMvcCore(action => action.EnableEndpointRouting = false);
```

OData doesn’t yet support .NET Core 3.0 – at the time of this article, .NET Core 3.0 is still in preview, OData support will extend to 3.0 once it’s production ready.

Once that part is done, let’s build a private method to do a handshake between your existing data models (Students in this case) and EDM, as follows:

```
1
    private IEdmModel GetEdmModel()
2
    {
3
        var builder = new ODataConventionModelBuilder();
4
        builder.EntitySet<Student>("Students");
5
        return builder.GetEdmModel();
6
    }
```

The student model we are using here is the same model we used in our previous article, as a reminder here’s how the model looks like:

```
1
    public class Student
```

```
2
    {
3
        public Guid Id { get; set; }
4
        public string Name { get; set; }
5
        public int Score { get; set; }
6
    }
```

Now that we have created our EDM method, now let’s do one last configuration change in the *Configure* method in our *Startup.cs* file as follows:

```
app.UseMvc(routeBuilder =>
    {
        routeBuilder.Select().Filter().OrderBy().Expand().Count().MaxTop(10);
        routeBuilder.MapODataServiceRoute("api", "api", GetEdmModel());
    });
```

Just like our last article, we enabled the functionality we needed such as select, filter and order by then we used the MapODataServiceRoute method to utilize our EDM method.

We used “api” instead of “odata” as our first and second parameters as a route name and a route prefix to continue to support our existing APIs endpoints, but there’s a catch to that.

Your contract in this case will change, if your API returns a list of students like this:

```
[
  {
    "id": "acc25b4f-c53d-4363-ad33-e0c860a83a1b",
    "name": "Hassan Habib",
    "score": 100
  },
  {
    "id": "d42daeb4-37d7-4a20-9e9b-7f7a60f27ff6",
```

```
"name": "Cody Allen",
    "score": 90
  },
  {
    "id": "db246814-d34e-40e4-aa00-b9192cec447b",
    "name": "Sandeep Pal",
    "score": 120
  },
  {
    "id": "c4e9efc9-40b7-4a85-b000-ce9c076fcd57",
    "name": "David Pullara",
    "score": 50
  }
]
```


With the EDM method, your contract will change a bit, your response will have some helpful metadata that we are going to talk about, and it will look like this:


```
{
  "@odata.context": "https://localhost:44374/api/$metadata#Students",
  "value": [
    {
      "Id": "9cef40f6-db31-4d4c-997d-8b802156dd4c",
      "Name": "Hassan Habib",
      "Score": 100
    },
    {
      "Id": "282be5ea-231b-4a59-8250-1247695f16c3",
      "Name": "Cody Allen",
      "Score": 90
    },
    {
      "Id": "b3b06596-729b-4c6f-b337-7ad11b01371b",
      "Name": "Sandeep Pal",
      "Score": 120
    },
    {
      "Id": "084bd81e-b8a2-471d-8396-ace675f73688",
```

```
"Name": "David Pullara",
      "Score": 50
    }
  ]
}
```

That extra metadata is going to help us perform more operations than the old method.

In that case if you have existing consumers for your API, I recommend introducing a new endpoint, version or informing them to change their contracts, otherwise this will be a breaking change.

The other option is to change your route name and route prefix parameters to say “odata” instead, which is the standard way to implement OData.

The last thing we need to do to make this work for us is removing the notations on top of your existing API controller class, in our case we will remove these two lines:

1
```
    [Route("api/[controller]")]
```

2
```
    [ApiController]
```

And don’t forget to add the enabling querying annotation on top of your API method:

1
```
    [EnableQuery()]
```

# Putting OData into Action

Once that’s done, now you can try to perform higher operations using OData like Count for instance, you can call your endpoint with */api/students?$count=true* and you should get:

As you can see here, you have a new property *@odata.count* that shows you the count of the items in your list.

# Final Notes

Now that you've learned about the simplest way (8 lines of code) to create a handshake between ASP.NET Core, OData and EDM here's few notes:

1. EDM doesn't have any dependency on the Entity Framework, in fact the whole purpose of creating an EDM is to link whatever data you have in any format it may be to the OData engine and serialize the results through an API endpoint.
2. EDM can only be useful if you need some specific OData features such as count and nextlink and so many other features that we will explore in future articles.
3. There's more to learn about EDM, I encourage you to check all about EDM in this extensive, comprehensive documentation.
4. OData is an open-source project, I encourage you as you benefit from it's amazing features to contribute to the project, suggest new features and participate with documentation and your experiences to keep the community active and useful for everyone.
5. You can clone the project I built and try things out from this github repo.

---

Revision #5

Created Wed, Apr 21, 2021 5:31 AM by Nelson

Updated Wed, Apr 21, 2021 5:38 AM by Nelson