Getting Started with EF Core

09/17/2019 • 3 minutes to read •  +8

In this article

In this tutorial, you create a .NET Core console app that performs data access against a SQLite database using Entity Framework Core.

You can follow the tutorial by using Visual Studio on Windows, or by using the .NET Core CLI on Windows, macOS, or Linux.

View this article's sample on GitHub.

Prerequisites

Install the following software:

.NET Core CLI | Visual Studio

- .NET Core SDK.

Create a new project

.NET Core CLI | Visual Studio

.NET Core CLI Copy

```
dotnet new console -o EFGetStarted
cd EFGetStarted
```

Install Entity Framework Core

To install EF Core, you install the package for the EF Core database provider(s) you want to target. This tutorial uses SQLite because it runs on all platforms that .NET Core supports. For a list of available providers, see Database Providers.

.NET Core CLI | Visual Studio

.NET Core CLI — Copy

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
```

Create the model

Define a context class and entity classes that make up the model.

.NET Core CLI | Visual Studio

- In the project directory, create **Model.cs** with the following code

C# — Copy

```
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;

namespace EFGetStarted
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder options)
            => options.UseSqlite("Data Source=blogging.db");
    }
```

```
    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public List<Post> Posts { get; } = new List<Post>();
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

EF Core can also reverse engineer a model from an existing database.

Tip: In a real app, you put each class in a separate file and put the connection string in a configuration file or environment variable. To keep the tutorial simple, everything is contained in one file.

Create the database

The following steps use migrations to create a database.

.NET Core CLI | Visual Studio

- Run the following commands:

.NET Core CLI Copy

```
dotnet tool install --global dotnet-ef
dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet ef migrations add InitialCreate
dotnet ef database update
```

This installs dotnet ef and the design package which is required to run the command on a project. The `migrations` command scaffolds a migration to create

the initial set of tables for the model. The `database update` command creates the database and applies the new migration to it.

Create, read, update & delete

- Open *Program.cs* and replace the contents with the following code:

C# Copy

```
using System;
using System.Linq;

namespace EFGetStarted
{
    class Program
    {
        static void Main()
        {
            using (var db = new BloggingContext())
            {
                // Create
                Console.WriteLine("Inserting a new blog");
                db.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
                db.SaveChanges();

                // Read
                Console.WriteLine("Querying for a blog");
                var blog = db.Blogs
                    .OrderBy(b => b.BlogId)
                    .First();

                // Update
                Console.WriteLine("Updating the blog and adding a post");
                blog.Url = "https://devblogs.microsoft.com/dotnet";
                blog.Posts.Add(
                    new Post
                    {
                        Title = "Hello World",
                        Content = "I wrote an app using EF Core!"
                    });
                db.SaveChanges();

                // Delete
                Console.WriteLine("Delete the blog");
                db.Remove(blog);
                db.SaveChanges();
            }
```

```
            }
        }
    }
```

Run the app

.NET Core CLI | Visual Studio

.NET Core CLI | Copy

```
dotnet run
```

Next steps

- Follow the ASP.NET Core Tutorial to use EF Core in a web app
- Learn more about LINQ query expressions
- Configure your model to specify things like required and maximum length
- Use Migrations to update the database schema after changing your model

Is this page helpful?

Yes No