Getting Started with EF Core on .NET Core Console App with a New database

04/05/2017 • 3 minutes to read • Contributors all

In this article

In this walkthrough, you will create a .NET Core console app that performs basic data access against a SQLite database using Entity Framework Core. You will use migrations to create the database from your model. See ASP.NET Core - New database for a Visual Studio version using ASP.NET Core MVC.

> **Note**
>
> The .NET Core SDK no longer supports `project.json` or Visual Studio 2015. We recommend you migrate from project.json to csproj. If you are using Visual Studio, we recommend you migrate to Visual Studio 2017.

> **Tip**
>
> You can view this article's sample on GitHub.

Prerequisites

The following prerequisites are needed to complete this walkthrough:

- An operating system that supports .NET Core.
- The .NET Core SDK 2.0 (although the instructions can be used to create an application with a previous version with very few modifications).

Create a new project

- Create a new `ConsoleApp.SQLite` folder for your project and use the `dotnet` command to populate it with a .NET Core app. 1

Console Copy

```
mkdir ConsoleApp.SQLite
cd ConsoleApp.SQLite/
dotnet new console
```

Install Entity Framework Core

To use EF Core, install the package for the database provider(s) you want to target. This walkthrough uses SQLite. For a list of available providers see Database Providers.

- Install Microsoft.EntityFrameworkCore.Sqlite and Microsoft.EntityFrameworkCore.Design

Console Copy

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

- Manually edit `ConsoleApp.SQLite.csproj` to add a DotNetCliToolReference to Microsoft.EntityFrameworkCore.Tools.DotNet: 2

XML Copy

```
<ItemGroup>
  <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Ver
</ItemGroup>
```

`ConsoleApp.SQLite.csproj` should now contain the following:

Copy

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>
```

```
  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="2.0.0" />
  </ItemGroup>
</Project>
```

Note: The version numbers used above were correct at the time of publishing. 1

- Run `dotnet restore` to install the new packages.

Create the model

Define a context and entity classes that make up your model.

- Create a new *Model.cs* file with the following contents.

C# Copy

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

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

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite("Data Source=blogging.db");
        }
    }

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

        public List<Post> Posts { get; set; }
    }

    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; }
    }
}
```

Tip: In a real application you would put each class in a separate file and put the connection string in a configuration file. To keep the tutorial simple, we are putting everything in one file.

Create the database

Once you have a model, you can use migrations to create a database.

- Run `dotnet ef migrations add InitialCreate` to scaffold a migration and create the initial set of tables for the model.
- Run `dotnet ef database update` to apply the new migration to the database. This command creates the database before applying migrations.

> **Note**
>
> When using relative paths with SQLite, the path will be relative to the application's main assembly. In this sample, the main binary is `bin/Debug/netcoreapp2.0/ConsoleApp.SQLite.dll`, so the SQLite database will be in `bin/Debug/netcoreapp2.0/blogging.db`.

Use your model

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

C# Copy

```
using System;

namespace ConsoleApp.SQLite
{
    public class Program
    {
        public static void Main()
        {
            using (var db = new BloggingContext())
            {
```

```
                db.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
                var count = db.SaveChanges();
                Console.WriteLine("{0} records saved to database", count);

                Console.WriteLine();
                Console.WriteLine("All blogs in database:");
                foreach (var blog in db.Blogs)
                {
                    Console.WriteLine(" - {0}", blog.Url);
                }
            }
        }
    }
}
```

- Test the app:

  `dotnet run`

  One blog is saved to the database and the details of all blogs are displayed in the console.

  Console | Copy

```
ConsoleApp.SQLite>dotnet run
1 records saved to database

All blogs in database:
 - http://blogs.msdn.com/adonet
```

Changing the model:

- If you make changes to your model, you can use the `dotnet ef migrations add` command to scaffold a new migration to make the corresponding schema changes to the database. Once you have checked the scaffolded code (and made any required changes), you can use the `dotnet ef database update` command to apply the changes to the database.
- EF uses a `__EFMigrationsHistory` table in the database to keep track of which migrations have already been applied to the database.
- SQLite does not support all migrations (schema changes) due to limitations in SQLite. See SQLite Limitations. For new development, consider dropping the database and creating a new one rather than using migrations when your model changes.

Additional Resources

- .NET Core - New database with SQLite - a cross-platform console EF tutorial. 1
- Introduction to ASP.NET Core MVC on Mac or Linux
- Introduction to ASP.NET Core MVC with Visual Studio
- Getting started with ASP.NET Core and Entity Framework Core using Visual Studio

Note

The feedback system for this content will be changing soon. Old comments will not be carried over. If content within a comment thread is important to you, please save a copy. For more information on the upcoming change, **we invite you to read our blog post**.