Ye Goode Dotnet Core Cookbook

This document contains just a bunch of random general things that you need to do to add X to your **ASP.NET Core 2** project.

# Essential Project setup

## NuGet packages for ASP.Net Core 2

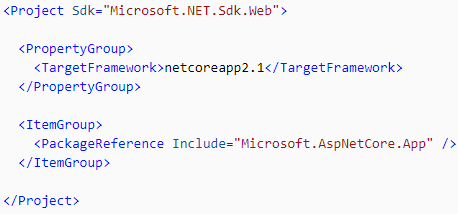
Add packages based on what you want:

1. **Microsoft.AspNetCore.App:** This is the essential starter meta-package. It pulls in all the basic stuff you’ll need in your project (this package supersedes Microsoft.AspNetCore.All. It includes the following:
   1. **Microsoft packages**
      1. All of ASP.NET Core – for serving web pages
      2. All of EF Core – for connecting to a data store
   2. Third-party packages:
      1. Json.NET
      2. Remotion.Linq
      3. IX-Async

The version number of the metapackage represents the versions of ASP.NET & EF Core that you’re using. So if you’re on 2.2 then you get ASP 2.2 and EF 2.2

Using the metapackage makes it so that you don’t have to include any of the ASP/EF shared libraries when you deploy your application.

Here is a sample minimalist .csproj config:



Note that we do not need to specify the version number. The version is implied by the Sdk attribute in the Project element, at the root.

This allows us to roll forward to the latest installed version of .NET Core.

## Program.cs

This is boilerplate, 100%

public class Program

{

    public static void Main(string[] args)

    {

        CreateWebHostBuilder(args).Build().Run();

    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>

        WebHost.CreateDefaultBuilder(args)

            .UseStartup<Startup>();

}

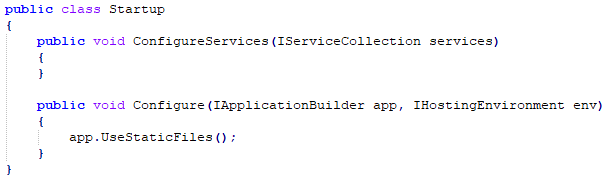
Don’t change this file.

It’s tempting to combine everything into Main(), but the tooling is going to look for a method called CreateWebHostBuilder. Function Main() is used so that you can run your ASP app from the command line.

The CreateDefaultBuilder() method contains all the boilerplate configuration to bootstrap your application. This is ASP’s way of making things super simple for you. You can override any of the default configuration in Startup.cs.

## Startup.cs

Here is the most minimal Startup.cs file you need:

****

The call to UseStaticFiles() gets you a wwwroot folder that you can serve static content from, but you get literally no other support.

If you want to get even more minimalist, you can put just this in your Configure() method:

app.Run(async (context) =>

{

    await context.Response.WriteAsync("Hello World!");

});

# Serving Static Content

**Static content can be anything:**

1. Images and stuff
2. HTML pages, CSS files, JavaScript, etc
3. A single page application (SPA), like React / Angular / Vue

It’s basically anything that isn’t generated server-side (not ASP, Razor, etc).

You can read about this more here:

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/static-files?view=aspnetcore-2.1>

## Configuration

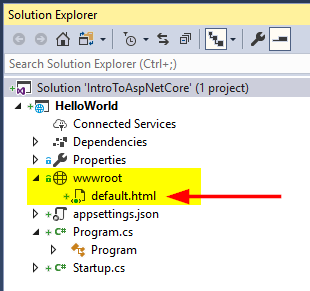
To enable static content, you need to add this line in your Startup.ConfigureServices() method:



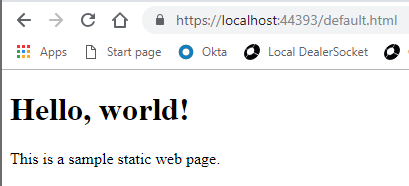
This will turn your wwwroot folder, but you don’t get any other behavior. For instance, it will not automatically look for a default HTML file if the client doesn’t specify a path.

Static files are stored within your project's web root directory. The default web root directory in your project is wwwroot. You can put any static content under this folder.

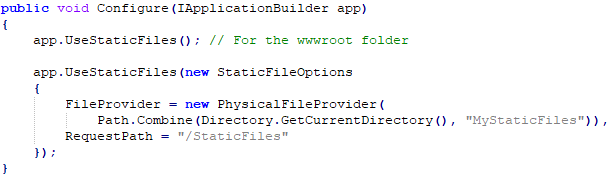
Here is a screenshot of your solution folder, for clarity:



Now you can start up your project and go to <http://localhost:port-number/default.html>.

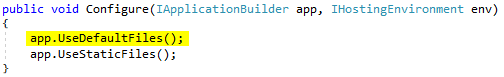


You can configure additional static folders by calling UseStaticFiles() as many times as you like, and each time passing in a StaticFileOptions object, like so:



## Setting a default start page

To enable a default start page, call the UseDefaultFiles() method before you call UseStaticFiles() method.

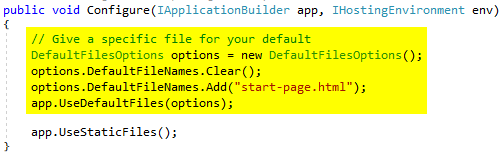


The UseDefaultFiles() method is actually a URL re-writer. It doesn’t tell ASP to do anything else and it doesn’t actually start ASP, so you still need to call UseStaticFiles() to get things going.

By default it will look for these four files in this order:

1. default.htm
2. default.html
3. index.htm
4. index.html

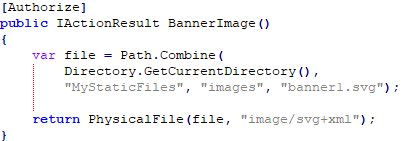
You can override this behavior and specify a start page of your own:



## Authorization

**WARNING: The static file middleware doesn’t provide any kind of authorization. Any files served up in your static folders are public to the entire world, literally.**

If you want to provide authorization then you need to use a MVC controller. Here is an example:



## Single page applications