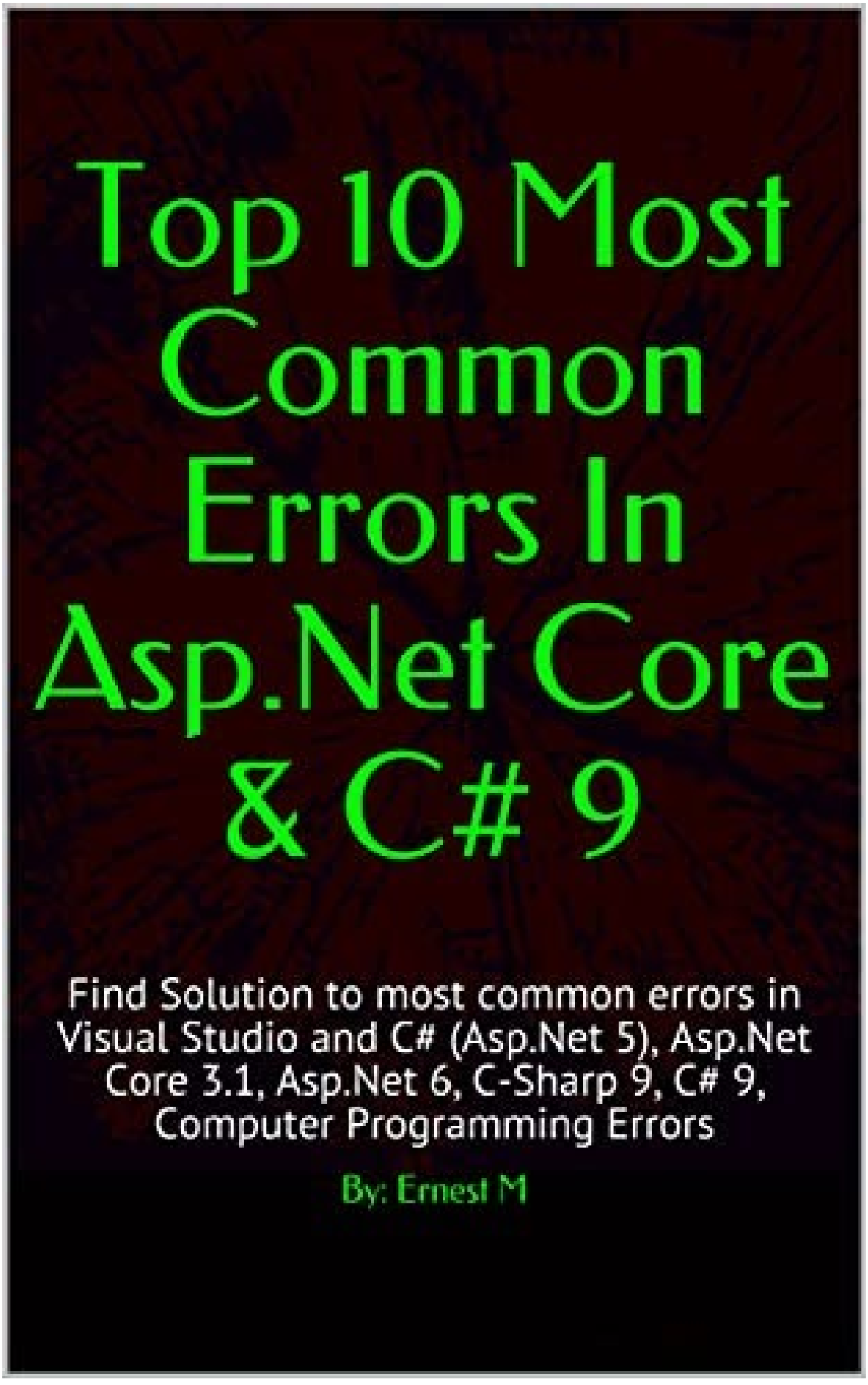
Top 10 Most Common Errors In Asp.Net Core & C# 9
Find Solution to most common errors in Visual Studio and C# (Asp.Net 5), Asp.Net Core 3.1, Asp.Net 6, C-Sharp 9, C# 9, Computer Programming Errors
By: Ernest M

Top 10 Most Common Errors In Asp.Net Core & C# 9

Find Solution to most common errors in Visual Studio and C# (Asp.Net 5)

1. MSB1009: Docker File Error when building Asp.Net 5

The most common Error when building a Docker File for Asp.Net Core 3.1 or .Net 5 is an MSB1009:

Error: Project file does not exist. Switch: ProjectName.csproj Error: Service 'ServiceName' Failed to build: The command '/bin/sh -c dotnet restore "ProjectName.csproj" returned a non-zero code: 1.

The error indicated above happens when you try to build a Docker Image for Asp.Net 5 or current version. The reason for this error is the constraiting files in one folder. This basically means that the appropriate files were not able to build.

Example of Dockerfile 1.1

```
1. FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build-env
2.WORKDIR /app

# Copy csproj and restore as distinct layers
3.COPY *.csproj ./
4.RUN dotnet restore

# Copy everything else and build
5.COPY . ./
6.RUN dotnet publish -c Release -o out

# Build runtime image
7.FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
8.WORKDIR /app
9.COPY --from=build-env /app/out .
10.ENTRYPOINT ["dotnet", "theNameOfYourAppComeInHere.dll"]
```

Reference: https://docs.docker.com/engine/examples/dotnetcore/

In the above Docker File, you can see that on line 1, we are pulling the Asp.Net Core 3.1 SDK and giving it an alias of "build-env", the alias is used to reference the SDK later on down the lines. It would be hard to type all of the text when in need to reference the line, instead the alias helps shorten the text.

[NB] *Keep in mind that in this case we are pulling down the SDK used for Development. Later on in the chapter I will show you how to speed up the process.*

On line 2, we are telling Docker to be in a directory called the /app, this is when Docker has pulled the SDK image and inside that Image itself. In the Docker File we are indicating that Docker has to be in the app directory in order to execute the next command which is **COPY *.csproj ./** on line 3.

Line 3 is coping the .csproj file and pasting it in the app/ directory, the reason it is indicted as ***.csproj** is because your Asp.Net Core 3.1 or Asp.Net 5 project file could be named anything. In this case the command looks for any file that has an extension of .**csproj** since in Asp.Net Framework a file with .csproj contains the project configuration, dependencies and project references.

Line number 4 defines a command that restores the project, basically it builds the project. Earlier, I mentioned that the .csproj file contains the project dependencies, when building the project the .csproj is used to restore all the dependencies defined in the .csproj file.

Line 5 copies everything from the current directory to another temp container at the root of the container. Then the application is published to the directory called “out” on line 6 using a command “release”. The app is published in a release mode, this is important if you are publishing the app for Production, the release flag makes sure that the app does not contain dependencies only used in development.

Line number 7 pulls down the Asp.Net Core 3.1 RunTime Image, the RunTime is used for running the application in Production. Most common errors will occur when you mistakenly pull the RunTime Image thinking it is an SDK, at this point you have to make sure the image the Docker File is pulling is the correct Image.

Line 8, instructs Docker Daemon to be in the directory called /app in order for the command on line 9 to execute. Line number 9 copies all files from the image referenced by the alias defined on line 1 to the directory called /app/out.

Finally line number 10 defines the Entry Point of an application, basically this line tells Docker to run a file called “theNameOfYourAppInHere.dll”.

The Docker File shown above requires the app to be in Development, however, In order to speed up the process and skip all the dependency pulling, the instruction below would help cut the building and publishing in half.

1. In Visual Studio, publish your application
2. Prepare your Dockerfile to look like below

```
1. FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
2.WORKDIR /app
5.COPY . ./
10.ENTRYPOINT ["dotnet", "theNameOfYourAppComeInHere.dll"]
```

3. After you prepare the Docker File, make sure you include the Docker File with the published application files.
4. Copy all the application files to the directory you want to build your Docker Image mainly on the Production Server.
5. Then navigate to the directory mentioned in step 4, and run the following command.
6. **Docker-compose build** , keep in mind you will have to install Docker Compose in order to have the **docker-compose build up** to run.

2. Error building Dot Net Core 2.2 Docker Image from a Docker File

MSBUILD: error MSB1009: Project file does not exist.

Switch: ProjectName.csproj

ERROR: Service 'ServiceName' failed to build: The command '/bin/sh -c dotnet restore "ProjectName.csproj"' returned a non-zero code: 1

Solution: **Make sure that your Dockerfile is in the root of your project as well as your docker-compose.yml file for the settings below in the code section** *(see Dockerfile Settings)* **to work.**

- You build the Docker Image by issuing the command **sudo docker-compose build** or in some cases if you want to build and run the container all at once then do: **sudo docker-compose up** in the folder where **docker-compose.yml** is located. Remember that docker-compose.yml should include the correct path to the Dockerfile.