**Adding Web Host Builder in ASP.NET Core**

##### **Changes in Program.cs class:**

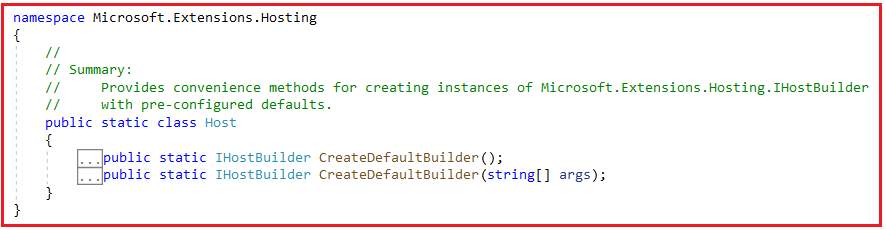
In order to make our console application a Web API Application, we also need to do some changes in our Program.cs class file which is the second step. We are going to do the following changes in our Program.cs class file

1. Adding the Web Host Builder
2. Configuring the Startup class.

Let us first understand what is Host Builder and then we will see how to add it.

##### **What is Host Builder?**

Host Builder is a static class that provides two methods and when we call these methods, they will add some features into ASP.NET Core Applications. The Host Builder Provides two convenient methods for creating instances of IHostBuilder with pre-configured defaults. If you go to the definition of Host class, then you will find two overloaded versions of the CreateDefaultBuilder method as shown in the below image.



**Note:** One method without parameter and the other method take string array as an argument. Let us discuss what pre-configured settings these two methods provide.

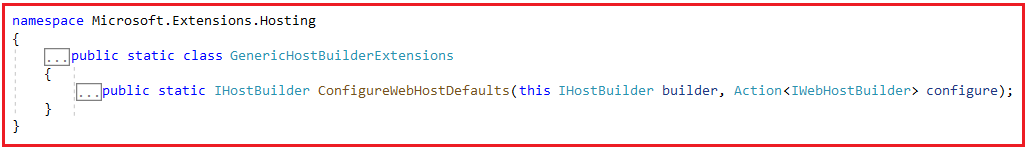
##### **CreateDefaultBuilder:**

The following defaults are applied to the returned Microsoft.Extensions.Hosting.HostBuilder:

1. By using the CreateDefaultBuilder method, we will get inbuilt support for Dependency Injection. This is one of the most popular design patterns used in almost all modern web applications and you get this by default when you add the CreateDefaultBuilder method. Also, enables scope validation on the dependency injection container when EnvironmentName is ‘Development’.
2. It load app Configurations from ‘appsettings.json’ and ‘appsettings.[EnvironmentName].json’. Configurations are basically the settings that our application uses.
3. The CreateDefaultBuilder method also set the ContentRootPath to the result of GetCurrentDirectory() method. The GetCurrentDirectory method belongs to the System.IO.Directory namespace.
4. The CreateDefaultBuilder method also configures the logging factory to log to the console, debug, and event source output. Nowadays, logging is mandatory for any kind of application. So, by using the CreateDefaultBuilder method, we also get logging support by default.
5. It also loads application Configuration from User Secrets when EnvironmentName is ‘Development’ using the entry assembly as well as load Configuration from environment variables.
6. It also loads host Configuration from supplied command line args. This is only applicable with the version which takes string array as an argument.

##### **ConfigureWebHostDefaults:**

Along with the CreateDefaultBuilder method, we also need to add the ConfigureWebHostDefaults method. This method Configures a HostBuilder with defaults for hosting a web app. Following is the signature of this method.



**Parameter**:

1. builder: The IHostBuilder instance to configure.
2. configure: The configure callback

**Returns**: A reference to the builder after the operation has completed.

The ConfigureWebHostDefaults method provides the following support.

1. Sets Kestrel Server as the Web Server and configure it using the application’s configuration providers. When we run the application using CLI, the kestrel web server is used.
2. Enables IIS Integration, etc.
3. It adds the Host Filtering middleware
4. Also adds the ForwardedHeaders middleware if ASPNETCORE\_FORWARDEDHEADERS\_ENABLED=true
5. It configures WebRootFileProvider to include static web assets from projects referenced by the entry assembly during the development

##### **Adding CreateHostBuilder Method in Program class:**

Before adding the CreateHostBuilder method, let us first add a class file with the name **Startup.cs** into our project. In our next article, we will discuss the use of Startup class. For now, simply create the Startup.cs class file. Once you add the Startup class, then modify the Program.cs class as shown in the below code. The IHostBuilder and Host class belong to Microsoft.Extensions.Hosting and the UseStartup class belongs to Microsoft.AspNetCore.Hosting namespace.

**using** *Microsoft.Extensions.Hosting;*

**using** *Microsoft.AspNetCore.Hosting;*

**using** *System;*

**namespace** *ConsoleToWebAPI*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

CreateHostBuilder**(**args**)**.Build**()**.Run**()**;

**}**

**public** **static** IHostBuilder CreateHostBuilder**(**string**[]** args**)** =**>**

Host.CreateDefaultBuilder**(**args**)**

.ConfigureWebHostDefaults**(**webBuilder =**>**

**{**

webBuilder.UseStartup**<**Startup**>()**;

**})**;

**}**

**}**

As you can see in the above code, we have created a method called CreateHostBuilder which takes string array as a parameter. Then we call the CreateDefaultBuilder method on the Host class bypassing the input array as an argument. Even you can also the other overloaded version of the CreateDefaultBuilder method which does not take any parameter. Then on the instance of the CreateDefaultBuilder method, we call the ConfigureWebHostDefaults method and also provide the Startup class name. In the Main method, first, we call the CreateHostBuilder method and then call the Build Method followed by the Run method.

**Build Method:**The Build Method runs the given actions to initialize the host. This can only be called once. It returns an initialized Microsoft.Extensions.Hosting.IHost object.

**Run Method:**It Runs the application and blocks the calling thread until the host shutdown.

Now if you compare the above code with the Program class auto-generated by ASP.NET Core Web API Application, then you will find everything as the same.

##### **Can I give some different names for the CreateHostBuilder method like MyCreateHostBuilder?**

If you are using some basic features of ASP.NET Core, then you will not face any problems. But if you use some features like Entity Framework core, then it will not work. The reason is if you are using EF Core, then EF Core will search for this CreateHostBuilder method to add some default configurations.

