# Laboratory -1: Learn Your Tool

Ref: Simone Chiaretta, Front-end Development with ASP.NET Core, Angular, and Bootstrap, John Wiley & Sons, Inc

## Objectives:

O[1]. To learn the execution procedure of the console and web applications.
O[2]. To learn the execution process of a program using .Net framework.

In this course, students are going to learn two different frameworks namely ConsoleApp (.Net Core/.Net Framework) and Web Application(ASP.NET Core/.Net Framework). In ConsoleApp students will practice exercises on C# programming language, and in Web Application, they will practice different interoperabilities between the web server and database server, connectivity of different web features namely form, API, and services, and authorization activities. Thus, the focus of today's laboratory is to learn your tool(Visual Studio 2019) properly to execute programs.

**Task1: How to run a program as ConsoleAPP?**

a) Create a simple program(SampleProgram.cs) using Console .Net Core.
b) Execute the program using Visual Stdio 2019 Editor.

The most important tool that comes with .NET Core is the dotnet host, which is used to launch .NET Core console applications, including the development tools, via the new .NET command-line interface (CLI). This CLI centralizes all the interactions with the framework and acts as the base layer that all other IDEs, like Visual Studio, use to build applications.
In order to try it out, just open the command prompt, create a new folder, move into this folder, and type dotnet new console. This command creates the skeleton of a new .NET Core console application, made of a Program.cs code file and the .csproj project definition file, named as the folder in which the command was launched. The new command can be executed using other arguments to specify the type of project to build: console (the one we used before), web, mvc, webapi, classlib, xunit (for unit testing), and etc. This is also the structure of all commands of the .NET CLI: dotnet followed by the command name, followed by its arguments.
Now that all the pieces are ready, the application can be executed by simply typing the command dotnet run. This first builds the application and then invokes it via the dotnet application host. In fact, this could be done manually as well, first by explicitly using the build command and then by launching the result of the build (which is a DLL with the same name of the folder where the application has been created) using the application host: dotnet bin\Debug\netcoreapp2.0\ consoleapplication.dll (consoleapplication is the name of the folder). In addition to building and running apps, the dotnet command can also deploy them and create packages for sharing libraries.

**SampleProgram.cs**

```
using System;
namespace ConsoleApplication
```

```
{
        public class Program
        {
                public static void Main(string[] args)
                {
                        Console.WriteLine("Hello World!");
                }
         }
}
```

c) Execute the program using MS Command Prompt.

- Atfirst, go to the specific directory, where your project is located, then run the following command:
  **dotnet new console --force --output ConsoleAppNetCore**
  **dotnet run --project ConsoleAppNetCore**
  //ConsoleAppNetCore is the project name.

Web Forms provided the abstractions to deal with HTTP and web server objects and introduced the concept of server-side events to hide the stateless nature of the web, using the ViewState. The result was a very successful, feature-rich web framework with a very approachable programming model.

### Task2: How to run a program in .Net Framework?

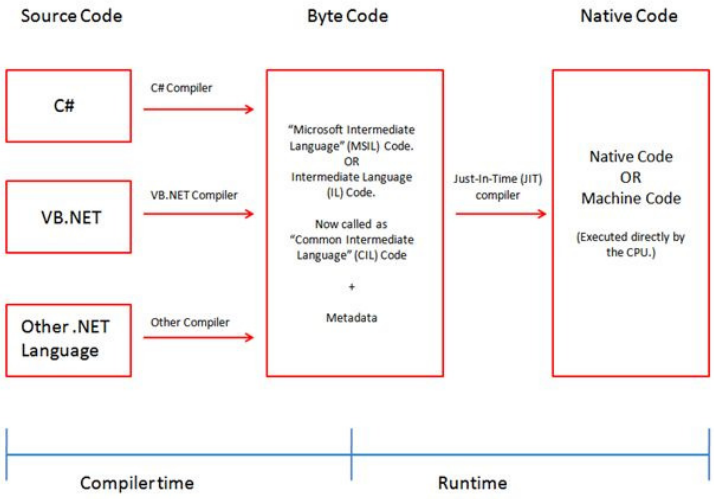


Fig 1: Compile and Runtime Environment of .Net Framework

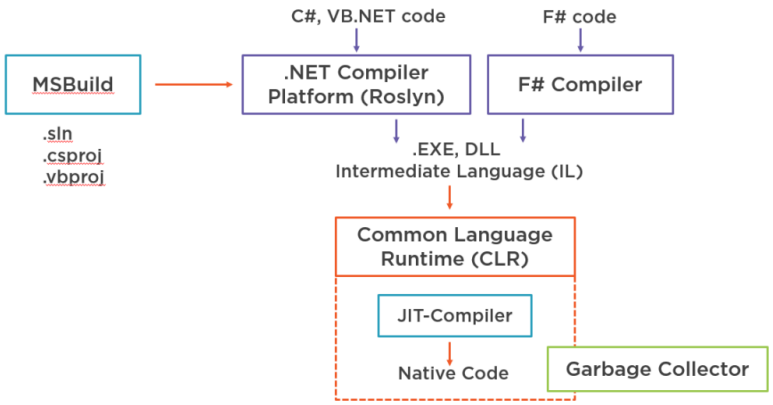


Fig 2: Detailed Runtime Environment of .Net Framework

.NET SDK is a set of libraries and tools for developing and running .NET applications. SDK download includes the following components:

- .NET CLI: Command-line tools that you can use for local development and continuous integration scripts.
- dotnet driver: A CLI command that runs framework-dependent apps.
- .NET runtime: Provides a type system, assembly loading, a garbage collector, native interop, and other basic services.
- Runtime libraries. Provides primitive data types and fundamental utilities.

**Task3: Cross-Language Interoperability.Net Framework?**

The .NET Framework is language-independent. We can develop many languages targeting the .NET Framework, such as C#, C++, F#, Visual Basic and Windows PowerShell. In simple words, a function, class or anything written in one language can be easily used in another language. (For example a function, class or anything written in C# can be easily used in VB.NET).

In our example we will use two(2) languages namely VB .Net and C#.

a) Create a simple WebApplication in .Net Framework using C#. Build and trace the path of the DLL in bin/debug folder.

```
namespace CrossPlatformApp
{
  public class Class1
  {
    public int Sum(int n1, int n2) {
      return n1 + n2;
    }

    public int Sum2(int n1, int n2) {
      return n1 + n2;
    }

    public int Sum3(int n1, int n2) {
```

```
            return n1 + n2;
        }
    }
}
```

b) Create a simple WebApplication in .Net Framework using VB.
Add refernce of the C# project then write the following codes

```
Imports System
Imports CrossPlatformApp.Class1

Module Program
        Dim obj As New CrossPlatformApp.Class1

        Sub Main(args As String())
            Console.WriteLine(obj.Sum(2, 3))
        End Sub
End Module
```

c) Execute the VB program with C# modules.