

# **.NET Main Entry Point, Project and Solution Introduction.**

**Modern .NET/C# Course**

**.NET Academy**

# Content

- .NET Project and Solution Overview.
- Entry Point Method– Main (Program.cs).
- Restore, Build, Run Application and Pack Application.
- Application Exit Code.
- Runapplication using input arguments

# .NET Project and Solution Overview



Before talking about application development, we need to define main aspects of application code environment – project and solution. Let's figure out what does project and solution mean. When you create an app or website in your IDE, you start with a project. In a logical sense, a project contains all files that are compiled into an executable, library, or website. Those files can include source code, icons, images, data files, and more. A project also contains compiler settings and other configuration files that might be needed by various services or components that your program communicates with. Your IDE may use MSBuild to build each project in a solution, and each project contains an MSBuild project file. The file extension reflects the type of project, for example, a C# project (.csproj), a Visual Basic project (.vbproj), or a database project (.dbproj). The project file is an XML document that contains all the information and instructions that MSBuild needs to build your project. Such information and instructions include the content, platform requirements, versioning information, web server or database server settings, and the tasks to perform. Project files are based on the MSBuild XML schema. To look at the contents of newer, SDK-style project files in Visual Studio, right-click the project node in Solution Explorer and select Edit <projectname>. To look at the contents of .NET projects, first unload the project (right-click the project node in Solution Explorer and select Unload Project). Then, right-click on the project and choose Edit <projectname>.

## .NET Academy

# ● .NET Project and Solution Overview.

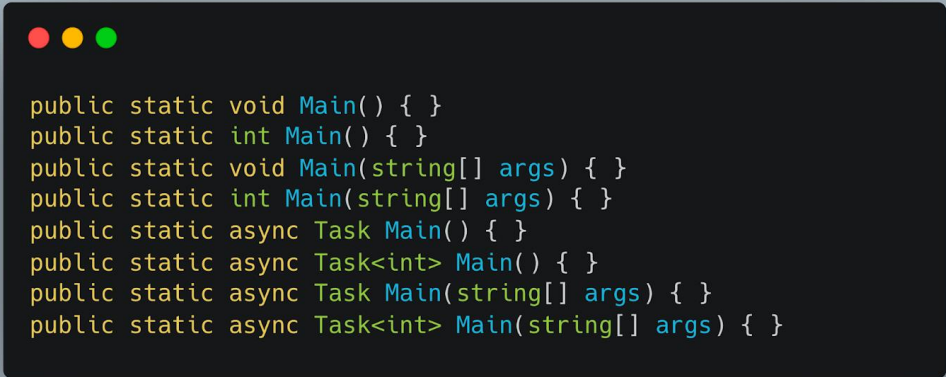
Great, the last thing that we need to figure out is Solution. A project is contained within a solution. Despite its name, a solution isn't an "answer". It's simply a container for one or more related projects, along with build information, Visual Studio window settings, and any miscellaneous files that aren't associated with a particular project.

# Entry Point Method – Main (Program.cs).

Okay, now we have one more question, what starts our application, where is the entry point? C# applications have an entry point called Main Method. It is the first method which gets invoked whenever an application started, and it is present in every C# executable file. The application may be a Console Application, ASP .NET Core or Windows Application. The most common entry point of a C# program is `static void Main()` or `static void Main(String []args)`.

# Entry Point Method – Main (Program.cs)

The following list shows valid Main signatures:



```
public static void Main() { }  
public static int Main() { }  
public static void Main(string[] args) { }  
public static int Main(string[] args) { }  
public static async Task Main() { }  
public static async Task<int> Main() { }  
public static async Task Main(string[] args) { }  
public static async Task<int> Main(string[] args) { }
```

# Restore, Build, Run Application and Pack Application

Then we have entry point for our application, you can run it:

- **dotnet run** - Builds and run our application. Use `dotnet run - - [arguments list]` to pass input arguments into application.
- **dotnet build** - Builds our application.
- **dotnet restore** - Restores all project dependencies.

# Application Exit Code

When your application ends execution, it returns an exit code. Exit code indicates does your application finished successfully or not. Exit code is 32-bit signed integer, the default value is 0 (zero), which means that the process completed successfully. Use a non-zero number to indicate an error. In your application, you can define your own error codes in an enumeration, and return the appropriate error code based on the scenario. For example, return a value of 1 to indicate that the required file is not present and a value of 2 to indicate that the file is in the wrong format.





# Application Exit Code

Let's see an examples to figure out how does we can use exit codes:

Use return keyword to set exit code:

```
using System;
using System.Numerics;

public class Example
{
    private const int ERROR_SUCCESS = 0;
    private const int ERROR_BAD_ARGUMENTS = 0xA0;
    private const int ERROR_ARITHMETIC_OVERFLOW = 0x216;
    private const int ERROR_INVALID_COMMAND_LINE = 0x667;

    public static int Main()
    {
        string[] args = Environment.GetCommandLineArgs();
        if (args.Length == 1) {
            return ERROR_INVALID_COMMAND_LINE;
        }
        else {
            BigInteger value = 0;
            if (BigInteger.TryParse(args[1], out value))
                if (value <= Int32.MinValue || value >= Int32.MaxValue)
                    return ERROR_ARITHMETIC_OVERFLOW;
                else
                    Console.WriteLine("Result: {0}", value * 2);

            else
                return ERROR_BAD_ARGUMENTS;
        }
        return ERROR_SUCCESS;
    }
}
```

# Application Exit Code

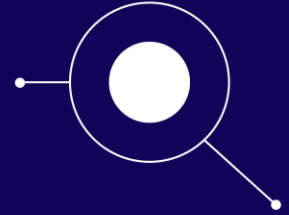
Use `Environment.ExitCode` Property to set exit code:

```
using System;
using System.Numerics;

public class Example
{
    private const int ERROR_BAD_ARGUMENTS = 0xA0;
    private const int ERROR_ARITHMETIC_OVERFLOW = 0x216;
    private const int ERROR_INVALID_COMMAND_LINE = 0x667;

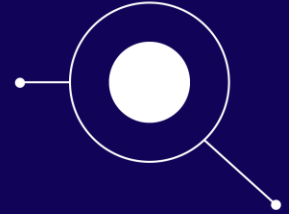
    public static void Main()
    {
        string[] args = Environment.GetCommandLineArgs();
        if (args.Length == 1) {
            Environment.ExitCode = ERROR_INVALID_COMMAND_LINE;
        }
        else {
            BigInteger value = 0;
            if (BigInteger.TryParse(args[1], out value))
                if (value <= Int32.MinValue || value >= Int32.MaxValue)
                    Environment.ExitCode = ERROR_ARITHMETIC_OVERFLOW;
                else
                    Console.WriteLine("Result: {0}", value * 2);
            else
                Environment.ExitCode = ERROR_BAD_ARGUMENTS;
        }
    }
}
```

# Questions for the study of the topic



1. What are the benefits of using solutions (.sln) to organise projects?
2. What files in a .csproj project can be customised, and how does this affect the project?
3. What is the role of the Main entry point in a .NET application?
4. Can a .NET application have multiple Main methods? If so, which one will be executed at startup?
5. What happens during the process of restoring project dependencies using the dotnet restore command?
6. Why is the dotnet build phase of .NET application development necessary and what result does it produce?

# Homework



1. What is the role of Main entry point in a .NET application?
2. How can I run a .NET application using the dotnet run command?
3. What is the purpose of Pack Application in .NET, and what code enables it?
4. What types of projects can be created in .NET, and what are they used for?
5. Create a console application in .NET and define a Main method.
  - a. In the Main method, write the code "Your first and last name".
  - b. Run the application and verify that it works correctly.

**Thank you for your attention.**

**.NET Academy**