# DOT NET Framework

##### ****What is COM?****

COM stands for Component Object Model. The COM is one of Microsoft Technologies. Using this technology we can develop windows applications as well as web applications. In earlier COM, VB is the programming language that is used to implement windows applications and ASP is used to implement web applications.

##### ****What are the disadvantages of COM?****

The major two disadvantages of COM is

1. Incomplete object-oriented programming means it will not support all the features of OOPs.
2. Platform dependent means COM applications can run on only Windows OS.

To overcome the above problems, the DOT NET Framework comes into the picture.

##### ****What .NET Represents?****

NET stands for Network Enabled Technology. In .NET, dot (.) refers to object-oriented and NET refers to the internet. So the complete .NET means through object-oriented we can implement internet-based applications.

##### ****What is a Framework?****

A framework is a software. Or you can say a framework is a collection of many small technologies integrated together to develop applications that can be executed anywhere.

##### ****What does the DOTNET Framework provide?****

The DOTNET Framework provides two things are as follows

1. **BCL** (Base Class Libraries)
2. **CLR** (Common Language Runtime)

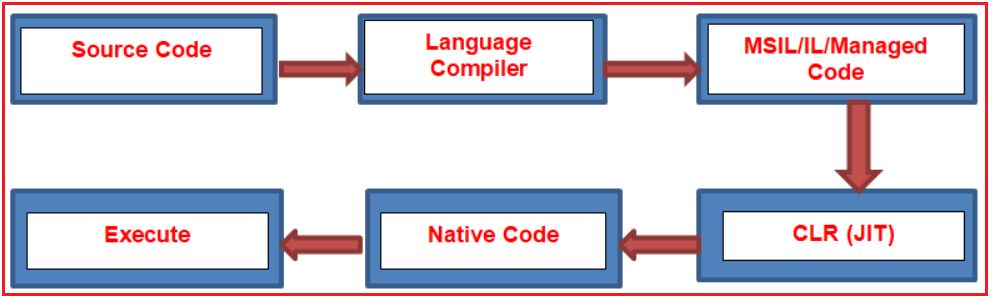
##### ****BCL****

Base Class Libraries (BCL) is designed by Microsoft. Without BCL we can’t write any code in .NET. So, BCL is also known as the Building block of .NET Programs. These are installed into the machine when we installed the .NET framework. BCL contains pre-defined classes and these classes are used for the purpose of application development.

The physical location of BCL is **C:\Windows\assembly**

###### **CLR**

CLR stands for Common Language Runtime and it is the core component under the .NET framework which is responsible for converting the MSIL (Microsoft Intermediate Language) code into native code. In our next article, we will discuss [**CLR**](https://dotnettutorials.net/lesson/common-language-runtime-dotnet/) in detail.



**In the .NET framework, the code is compiled twice.**

1. In the 1st compilation, the source code is compiled by the respective language compiler and generates the intermediate code which is known as **MSIL (Microsoft Intermediate Language)** or **IL (Intermediate language code)** Or **Managed Code**.
2. In the 2nd compilation, **MSIL** is converted into **Native code** (native code means code specific to the Operating system so that the code is executed by the Operating System ) and this is done by **CLR**.

Always 1st compilation is slow and 2nd compilation is fast.

##### ****What is JIT?****

JIT stands for the **Just-in-Time** compiler. It is the component of **CLR** that is responsible for converting **MSIL** code into **Native Code**. Native code is the code that is directly understandable by the operating system.

##### ****Different types of DOTNET Framework.****

The .net framework is available in three different flavors

1. **DOTNET Framework**: This is the general version required to run .NET applications on Windows OS only.
2. **.NET mono Framework:** This is required if we want to run DOT NET applications on other OS like Unix, Linux, MAC OS, etc.
3. **DOT NET Compact Framework**: This is required to run .NET applications on other devices like mobile phones and smartphones.

There is another company known as “**NOVEL**” designed a separate framework known as “**MONO Framework**”. Using this framework we can run **MSIL** on different **OS** Like **Linux, UNIX, Mac, BSD, OSX**, etc.

.NET is platform-dependent using the .NET framework but independent using the MONO framework.

##### ****What is not DOT NET?****

1. .NET is not an Operating system.
2. It is not an application or package.
3. .NET is not a database
4. It is not an ERP application.
5. .NET is not a Testing Tool.
6. It is not a programming language.

##### ****What is exactly DOTNET?****

.NET is a framework tool that supports many programming languages and many technologies. .NET support 60+ programming languages. In 60+ programming languages, 9 are designed by Microsoft and the remaining are designed by non-Microsoft.

Microsoft designed programming languages are as follows

1. VB.NET
2. C#.NET
3. VC++.NET
4. J#.NET
5. F#.NET
6. Jscript.NET
7. WindowsPowerShell
8. Iron phyton
9. Iron Ruby

Technologies supported by the .NET framework are as follows

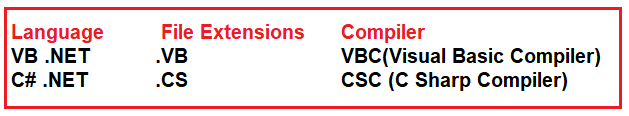
1. ASP.NET (Active Server Pages.NET)
2. ADO.NET (Active Data Object.NET)
3. WCF (Windows Communication Foundation)
4. WPF (Windows Presentation Foundation)
5. WWF (Windows Workflow Foundation)
6. AJAX (Asynchronous JavaScript and XML)
7. LINQ (Language Integrated Query)

##### ****What is a language and its need?****

1. Language acts as the mediator between the programmer and the system.
2. It offers some rules and regulations for writing the program.
3. The language also offers some libraries which are required for writing the program.

##### ****What are Technology and its needs?****

1. Technology is always designed for a particular purpose.
2. For example development of web-related applications in .NET using a technology ASP.NET.
3. But the technology does not offer any specific rules for writing the programs. That’s why technology can’t be implemented individually.
4. VB.NET, C#.NET both are programming languages. Using these two languages we can implement windows/desktop applications individually.
5. Every language is having its own compiler



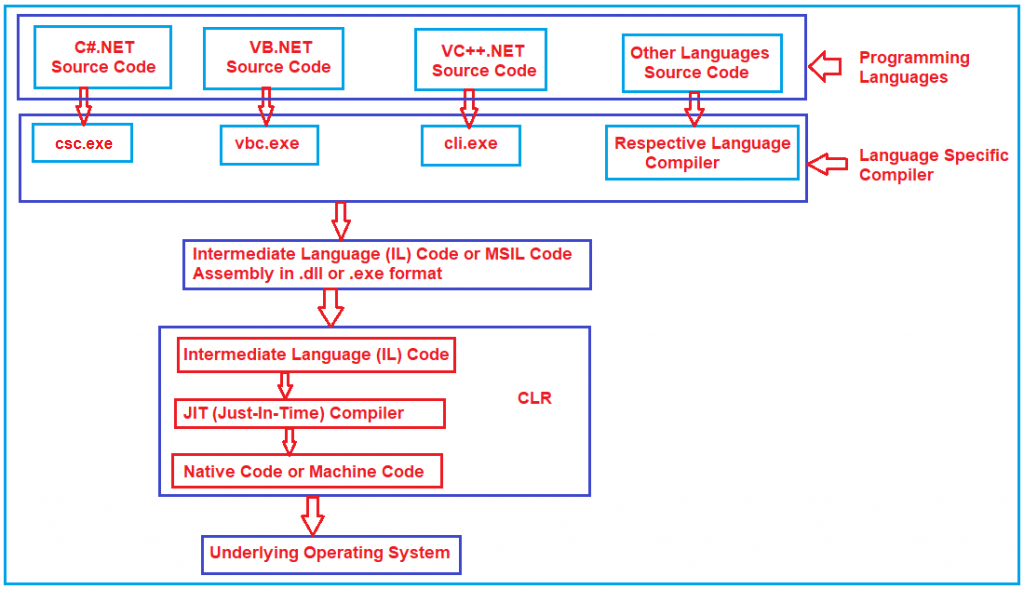
# Common Language Runtime in .NET

## ****Common Language Runtime (CLR) in C#.NET:****

In this article, I am going to discuss the **Common Language Runtime (CLR) in .NET Framework**. Please read our previous article before proceeding to this article where we gave a brief introduction to the [**DOT NET Framework**](https://dotnettutorials.net/lesson/dotnet-framework/). At the end of this article, you will understand all about CLR in C# with examples. But before understanding CLR in .NET, let us first understand how a .NET application is compiled and run.

##### ****How is a .NET application Compiled and Run?****

In order to understand how exactly a .NET Application is compiled and run, please have a look at the following image.



First, the developer has to write the code using any dot net supported programming languages such as C#, VB, J#, etc. Then the respective language compiler will compile the program code and generate something called **Microsoft Intermediate language (MSIL) or Intermediate language (IL)** code. For example, if the programming language is **C#**, then the compiler is **CSC** and if the programming language is **VB**, then the compiler will be **VBC**. This **Intermediate Language (IL)** code is half compiled code i.e. partially compiled code and cannot be executed directly by the Operating System. To execute this **Microsoft Intermediate language (MSIL) or Intermediate language (IL)** code on your machine, the .NET Framework provides something called **Common Language Runtime (CLR)** which takes the responsibility to execute your **Microsoft Intermediate language (MSIL) or Intermediate language (IL)**Code.

The CLR takes the IL (Intermediate Language) code and gives it to something called JIT (Just-in-Time) Compiler. The JIT compiler reads each and every line of the IL code and converts it to machine-specific instructions (i.e. into binary format) which can be executed by the underlying Operating System.

##### ****What is Intermediate Language (IL) Code in .NET Framework?****

The Intermediate Language or [**IL code**](https://dotnettutorials.net/lesson/intermediate-language/) in .NET Framework is a half compiled or partially compiled or CPU-independent partially compiled code and this code can not be executed by Operating System.

##### ****Why Partial Compiled Code or why not fully compiled Code?****

As a developer, you may be thinking about why the respective language compiler generates partially compiled code or why not fully compiled code i.e. machine code or binary code in .NET Framework. The reason is very simple. We don’t know in what kind of environment .NET Code is going to be run (for example, Windows XP, Windows 7, Windows 10, Windows Server, etc.). In other words, we don’t know what operating system is going to run our application; we also don’t know the CPU configuration, Machine Configuration, Security Configuration, etc. So, the Microsoft Intermediate language (MSIL) or Intermediate language (IL) code is partially compiled, and at runtime, this Microsoft Intermediate language (MSIL) or Intermediate language (IL) code is compiled to machine-specific instructions or you can say binary code using environmental properties such as Operating System, CPU, Machine Configuration, etc. by the CLR in .NET Framework.

##### ****Common Language Runtime (CLR) in .NET Framework:****

CLR is the heart of the .NET Framework and it contains the following components.

1. Security Manager
2. JIT Compiler
3. Memory Manager
4. [**Garbage Collector**](https://dotnettutorials.net/lesson/garbage-collector/)
5. [**Exception Manager**](https://dotnettutorials.net/lesson/exception-handling-csharp/)
6. [**Common Language Specification (CLS)**](https://dotnettutorials.net/lesson/common-language-specification/)
7. [**Common Type System (CTS)**](https://dotnettutorials.net/lesson/common-type-system/)

Let us discuss what each of these components does in detail.

##### ****Security Manager:****

There are basically two components to manage security. They are as follows:

1. **CAS (Code Access Security)**
2. **CV (Code Verification)**

These two components are basically used to check the privileges of the current user that the user is allowed to access the assembly or not.  The Security Manager also checks what kind of rights or authorities this code has and whether it is safe to be executed by the Operating System. So, basically, these types of checks are maintained by the Security Manager in .NET Application.

##### ****JIT Compiler:****

The JIT (Just-In-Time) Compiler is responsible for Converting the MSIL code into native code (Machine Code or Binary code) that is executed by the Operating System. The native code (Machine Code or Binary code) is directly understandable by the system hardware. JIT compiles the code just before the execution and then saves this translation in memory.

##### ****Memory Manager:****

The memory manager component of CLR in the .NET Framework allocates the necessary memory for the variables and objects that are to be used by the application.

##### ****Garbage Collector:****

When a dot net application runs, lots of objects are created. At a given point in time, it is possible that some of those objects are not used by the application. So, [**Garbage Collector in .NET Framework**](https://dotnettutorials.net/lesson/garbage-collector/) is nothing but is a **Small Routine** or you can say it’s a **Background Process Thread** that runs periodically and try to identify what objects are not being used currently by the application and de-allocates the memory of those objects.

##### ****Exception Manager:****

The [**Exception Manager**](https://dotnettutorials.net/lesson/exception-handling-csharp/) component of CLR in the .NET Framework redirects the control to execute the catch or finally blocks whenever an exception has occurred at runtime.

##### ****Common Type System (CTS) in .NET Framework:****

The .NET Framework supports many programming languages such as C#, VB.NET, J#, etc. Every programming language has its own data type. One programming language data type cannot be understood by other programming languages. But, there can be situations where we need to communicate between two different programming languages. For example, we need to write code in the VB.NET language and that code may be called from C# language. In order to ensure smooth communication between these languages, the most important thing is that they should have a [**Common Type System (CTS**](https://dotnettutorials.net/lesson/common-type-system/)) which ensures that data types defined in two different languages get compiled to a common data type.

CLR in .NET Framework will execute all programming language’s data types. This is possible because CLR having its own data types which are common to all programming languages. At the time of compilation, all language-specific data types are converted into CLR’s data type. This data type system of CLR is common to all .NET Supported Programming languages and this is known as the [**Common Type System**](https://dotnettutorials.net/lesson/common-type-system/)(CTS).

##### ****CLS (Common Language Specification) in .NET Framework:****

[**CLS (Common Language Specification)**](https://dotnettutorials.net/lesson/common-language-specification/)is a part of CLR in the .NET Framework. The .NET Framework supports many programming languages such as C#, VB.NET, J#, etc. Every programming language has its own syntactical rules for writing the code which is known as language specification. One programming language syntactical rules (language specification) cannot be understood by other programming languages. But, there can be situations where we need to communicate between two different programming languages. In order to ensure smooth communication between different .NET Supported Programming Languages, the most important thing is that they should have **Common Language Specifications**which ensures that language specifications defined in two different languages get compiled to a Common Language Specification.

CLR in .NET Framework will execute all programming language’s code. This is possible because CLR having its own language specification (syntactical rules) which are common to all .NET Supported Programming Languages. At the time of compilation, every language compiler should follow this language specification of CLR and generate the MSIL code. This language specification of CLR is common for all programming languages and this is known as [**Common Language Specifications (CLS)**](https://dotnettutorials.net/lesson/common-language-specification/)**.**

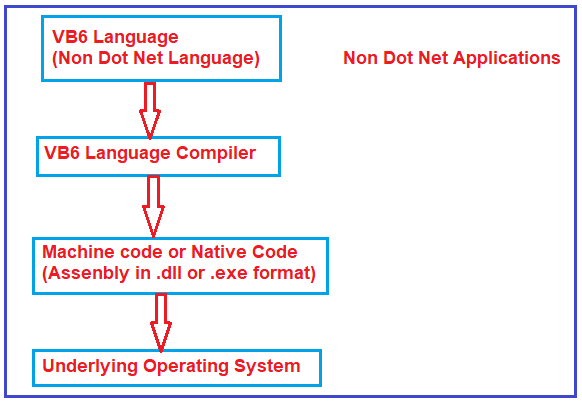
# .NET Program Execution Process

## ****.NET Program Execution Process Flow:****

In this article, I am going to discuss the **.NET Program Execution Process Flow**in Detail. Please read our previous article where we discussed the [**Common Language Runtime (CLR)**](https://dotnettutorials.net/lesson/common-language-runtime-dotnet/) architecture in detail. As .NET Developers, we should know when we create an application, how the application is compiled, and how the application is executed by the .NET Framework. But before understanding the **.NET Program Execution** process, let us first understand how non-dot net applications such as C, VB6, and C++ programs are executed.

##### ****Non-DOT NET Program Execution**** ****Process****:

We know that computers only understand machine-level code. The Machine-level code is also known as native code or binary code. So when we compile a C, VB6, or C++ program the respective language compiler compiles the respective language source code and generates the native machine code (also called binary code) which can be understood by the underlying operating system and the system hardware. The above process is shown in the below image.



The Native code or machine code that is generated by the respective language compiler is specific to the operating system on which it is generated. If we take this compiled native code and try to run it on another operating system, then it will fail. So the problem with this style of program execution is that it is not portable from one platform to another platform. That means it is platform-dependent.

##### ****.NET Program Execution Process:****

Let us now understand the .NET Program Execution Process in detail. Using .NET we can create different types of applications such as Console, Windows, Web, and Mobile Applications. Irrespective of the type of application when we execute any .NET application the following things are happening in order

The .NET application Source Code gets compiled into Microsoft Intermediate language (MSIL) which is also called Intermediate language (IL) or Common Intermediate language (CIL) code. Both .NET and Non-DOTNET applications generate an assembly when we compile the application. Generally, the assemblies have an extension of. DLL or .EXE based on the type of application we compiled. For example, if we compile a Window or Console application in .NET, we get an assembly of type .EXE whereas when we compile a Web or Class Library Project in .NET, we get an assembly of type .DLL.

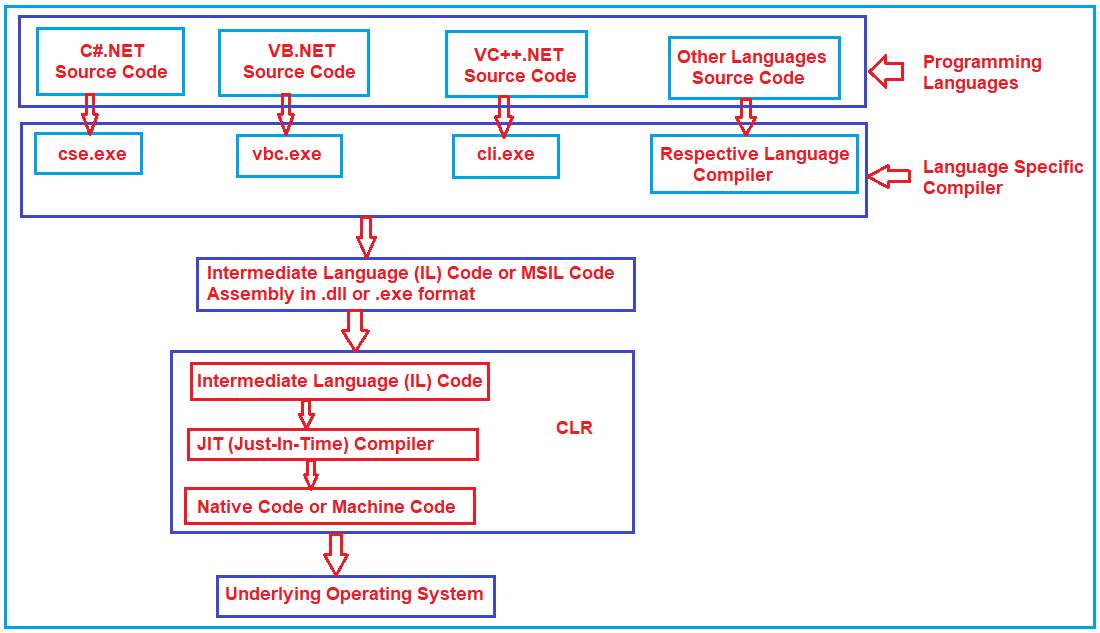
The difference between a .NET and NON-DOTNET assembly is that .NET Assembly is an Intermediate Language format whereas NON-.NET assembly is in native code format.

The NON .NET applications can run directly on top of the operating system as the NON-DOTNET assembly contains the native code whereas .NET applications run on top of a virtual environment called **Common Language Runtime (CLR)**. CLR contains a component called **Just-In-Time Compiler (JIT)** which will convert the Intermediate language into native code which can be understood by the underlying operating system.

##### ****.NET Program Execution Steps:****

In .NET, the application execution consists of 2 steps. They are as follows:

In step1 the respective language compiler compiles the Source Code into Intermediate Language (IL) and in the 2nd step, the JIT compiler of CLR will convert the Intermediate Language (IL) code into native code (Machine Code or Binary Code) which can then be executed by the underlying operating system. The above process is shown in the image below.



As the .NET assembly is in Intermediate Language (IL) format and not in native code or machine code, the .NET assemblies are portable to any platform as long as the target platform has the **Common Language Runtime (CLR)**. The target platform’s CLR converts the Intermediate Language code into native code or machine code that the underlying operating system can understand.

Intermediate Language code is also called managed code. This is because CLR manages the code that runs inside it. For example, in a VB6 program, the developer is responsible for de-allocating the memory consumed by an object. If a programmer forgets to de-allocate memory, then it may get out of memory exceptions. On the other hand, a .NET programmer needs not worry about de-allocating the memory consumed by an object. Automatic memory management is also known as garbage collection is provided by CLR. Apart from garbage collection, there are several other benefits provided by the CLR which we will discuss in a later session. Since CLR is managing and executing the Intermediate Language it (IL) is also called the managed code.

.NET supports different programming languages like C#, VB, J#, and C++. C#, VB, and J# can only generate managed code (IL) whereas C++ can generate both managed code (IL) and unmanaged code (Native code).

The native code is not stored permanently anywhere after we close the program the native code is thrown away. When we execute the program again the native code gets generated again.

The .NET program is similar to java program execution. In Java, we have bytecodes and JVM (Java Virtual Machine) whereas in .NET we have Intermediate Language and CLR (Common Language Runtime).

# Intermediate Language (ILDASM & ILASM) Code in C#

## ****Intermediate Language (ILDASM & ILASM) in C#.NET****

In this article, I am going to discuss **Intermediate Language (ILDASM & ILASM) Code in C#**with Examples. Please read our previous article, where we discussed the [**.NET Program Execution Process Flow**](https://dotnettutorials.net/lesson/dotnet-program-execution-process/) in detail. **ILDASM** stands for Intermediate Language disassembler and **ILASM** stands for Intermediate language assembler. As part of this article, we are going to discuss the following pointers and at the end of this article, you will understand all about Intermediate Language (IL Code) in C#.

1. **What happens when we compile a .NET Application?**
2. **Understanding the Intermediate Language (IL Code) in C#?**
3. **What are ILDASM and ILASM?**
4. **How to view the Intermediate Language code in C#?**
5. **What is Manifest?**
6. **How to export the Intermediate Language code to a text file?**
7. **How to rebuild an assembly from a text file which contains manifest and IL?**

##### ****What happens when we compile a .NET Application?****

When we compile any .NET application. it will generate an assembly with the extension of either a .DLL or an .EXE. For example, if you compile a Windows or Console application, then you will get an .EXE, whereas if you compile a Web or Class library project, then you will get a .DLL. Irrespective of whether it is a .DLL or .EXE, an assembly consists of two things i.e. **Manifest and Intermediate language**. Let us understand how the Intermediate Language and Manifest look like in .NET Framework with an example.

##### ****Understanding Intermediate Language (ILDASM and ILASM) Code in C#:****

In order to understand Intermediate Language Code (ILDASM and ILASM) in C#, let us create a simple console application. Once you create the console application, please modify the Program class as shown below.

**using** *System;*

**namespace** *ILDASMDemo*

**{**

**class** Program

**{**

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

**{**

Console.WriteLine**(**"Understanding ILDASM and ILASM"**)**;

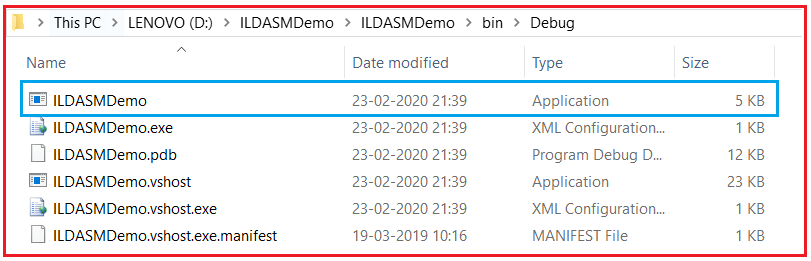
Console.Read**()**;

**}**

**}**

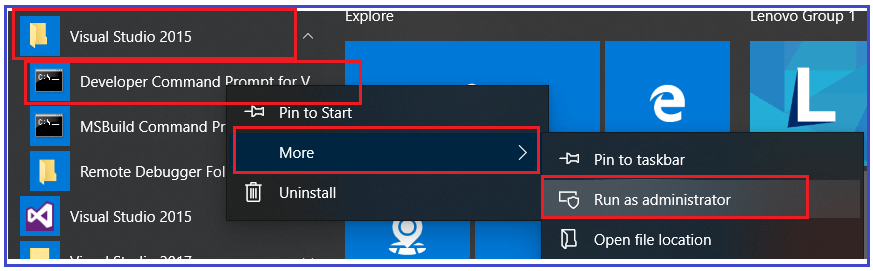
**}**

Now, build the application. Once you build the application, the above source code is compiled and intermediate language code generated and packaged into an assembly. In order to see the assembly, just right-click on the Project and select **Open Folder in File Explorer** option and then go to the **bin => Debug** folder and you should see an assembly with .exe extension as shown in the below image as it is a console application.

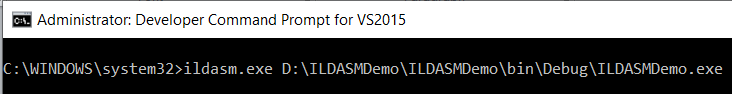


##### ****How to view the Intermediate Language Code in .NET Framework?****

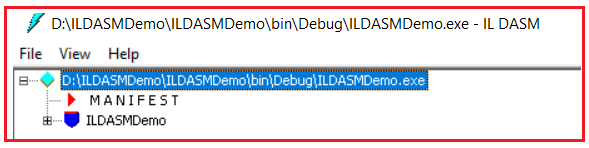
The .NET framework provides a nice tool called **ILDASM (Intermediate Language DisAssember)** to view the code of the intermediate language in C#.NET. In order to use the ILDASM tool, you need to follow the below steps. Open Visual Studio Command Prompt in Administrator mode as shown in the below image.



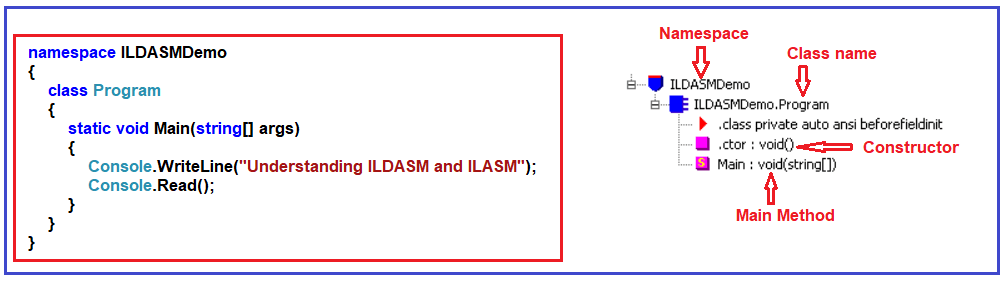
Once you open the visual studio command prompt in administrative mode, then type the “**Ildasm.exe C:\YourDirectoryPath\YourAssembly.exe**” command and press enter. Here, you need to provide the exe path where your exe is generated. My exe is generated in the path “**D:\ILDASMDemo\ILDASMDemo\bin\Debug\ILDASMDemo.exe**”, so I execute the following code in the command prompt:



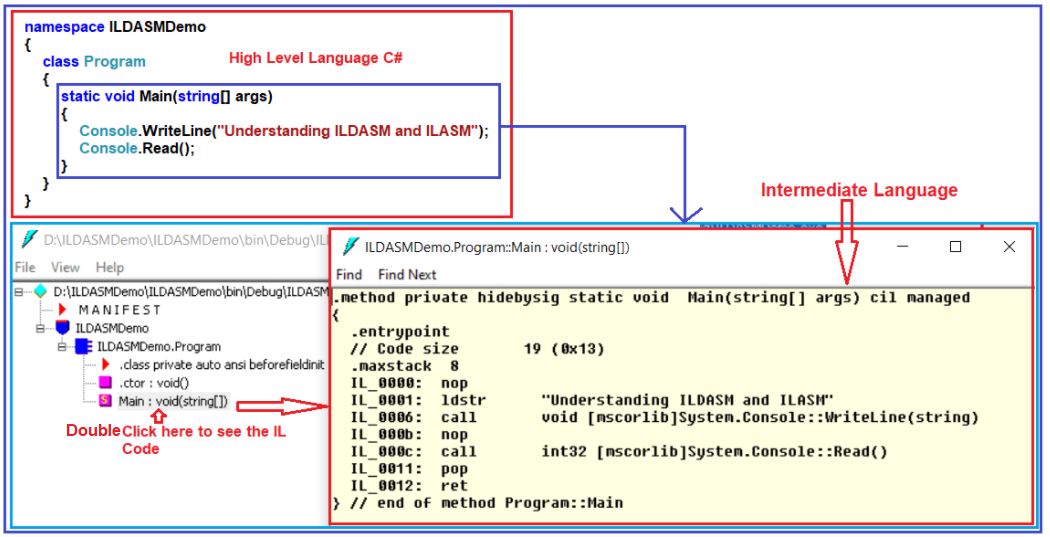
Once you type the above command and press enter, it should open the following ILDASM window.



As you can see, the assembly is consists of two things (**Manifest and Intermediate language**). Let us first discuss the intermediate language code and then we will discuss what Manifest is. Now, let us expand the ILDASMDemo and compare it with our code. For better understanding, please have a look at the below image. There is a constructor in ILDASM and this is because by default the .NET Framework provides a default constructor when there is no constructor in your class. You can also the Main method in the intermediate language code

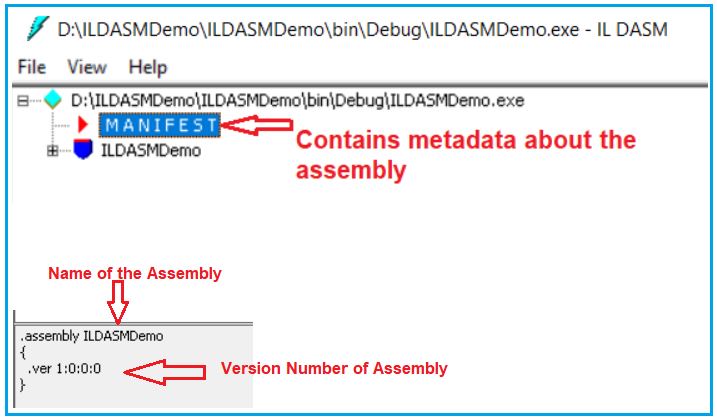


Now, double click on the Main method on the ILDASM window to see the intermediate language generated for the Main method as shown below.



##### ****What is Manifest?****

Manifest contains metadata about the assembly like the name of the assembly, the version number of the assembly, culture, and strong name information as shown in the below image.

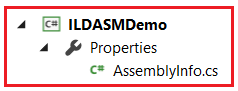


Metadata also contains information about the referenced assemblies. Each reference includes the dependent assembly’s name, assembly metadata (version, culture, operating system, and so on), and public key, if the assembly is strongly named.

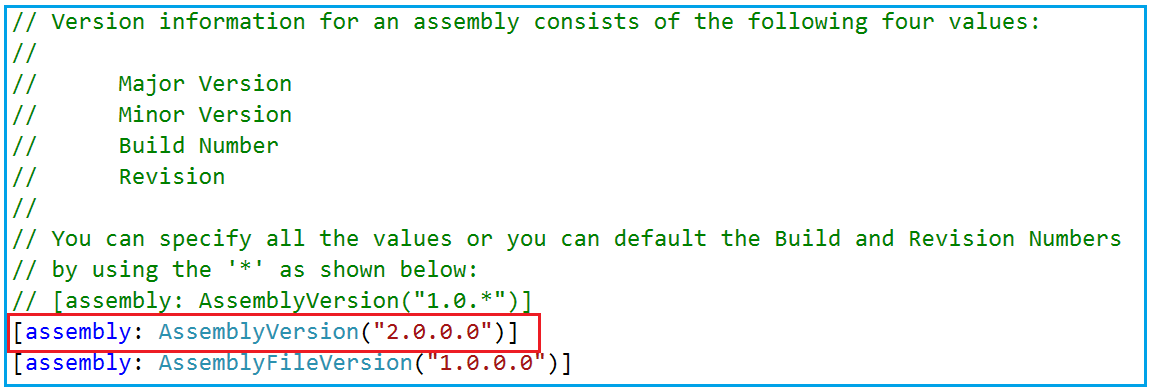
##### ****How to change Assembly info?****

It is also possible to change or modify some of the information in the assembly manifest using attributes. For example, if you want to modify the version number, then you need to follow the below steps.

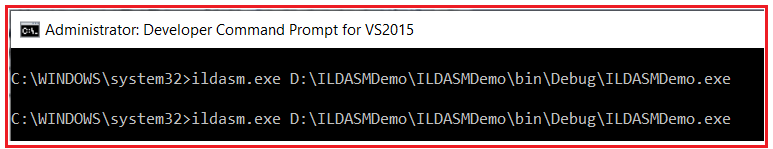
Open **AssemblyInfo.cs** class file which is present under the **Properties** folder as shown below. Every project in .NET has a properties folder.



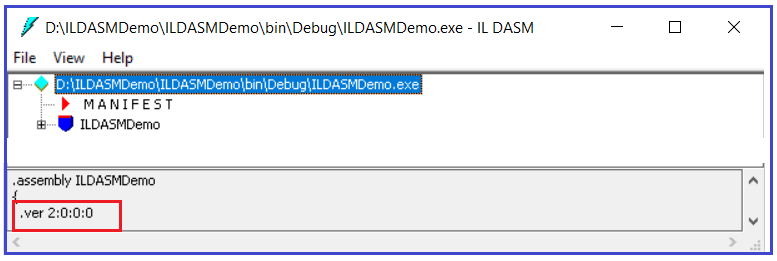
In this file, you will find one attribute called the **AssemblyVersion** attribute, which is by default set to 1.0.0.0. Now, change this value to 2.0.0.0 as shown below.



Now, rebuild the solution. But before that close the ILDASM window otherwise you will get an error. Once you rebuild the solution then open the assembly using the same **ILDASM.exe** in the command prompt as shown below.



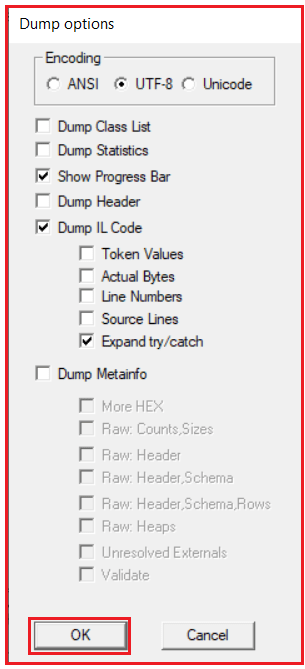
Once you execute the above command, it should open the assembly. At the bottom, you can find the updated version number of the assembly as expected as shown below.



##### ****How to export the Intermediate Language code to a text file?****

If you want to export or save the Intermediate Language code into a text file, then you need to follow the below steps.

Select **File Menu** Option from the **ILDASM tool** and then select **Dump** and you will see “**Dump Options Window**” and click on the **OK** button on the “Dump Options Window” as shown below.



Now you need to enter the file name as per your choice. I am entering the file name as **MyFile** and save it to the **D:** drive. Now navigate to D: drive in windows explorer and you should see MyFile.il Now, open MyFile.il with notepad and you should see assembly metadata and IL code.

##### ****How to rebuild an assembly from a text file which contains manifest and IL?****

If you want to rebuild an assembly from IL code then you need to use a tool called ILASM.exe. So, let’s go and create an assembly from the file (MyFile.il) that we just save. In order to rebuild an assembly, please follow the below steps.

Type the following command in “Visual Studio Command Prompt” and press enter  
         **ILASM.exe D:\MyFile.il**  
Now navigate to D: drive in windows explorer and you should see MyFile.exe. So, in short, we use **ILASM.exe** (Intermediate Language Assembler) to reconstruct an assembly from a text file that contains manifest and IL.

# Common Type System in .NET Framework

## ****Common Type System (CTS) in .NET Framework****

In this article, I am going to discuss the **Common Type System in .NET Framework**. Please read our previous article, where we discussed the [**Intermediate Language in .NET Framework**](https://dotnettutorials.net/lesson/intermediate-language/) with Examples. At the end of this article, you will understand what is Common Type System (CTS) in C# and how CTS in .NET works?

##### ****What is**** ****the Common Type System in .NET Framework?****

The .NET Framework supports many programming languages such as C#, VB.NET, J#, etc. Every programming language has its own data type. One programming language data type cannot be understood by other programming languages. But, there can be situations where we need to communicate between two different programming languages. For example, we need to write code in the VB.NET language and that code may be called from C# language. In order to ensure smooth communication between these languages, the most important thing is that they should have a Common Type System (CTS) which ensures that data types defined in two different languages get compiled to a common data type.

CLR in .NET Framework will execute all programming language’s data types. This is possible because CLR having its own data types which are common to all programming languages. At the time of compilation, all language-specific data types are converted into CLR’s data type. This data type system of CLR is common to all .NET Supported Programming languages and this is known as the Common Type System (CTS).

##### ****Example: Common Type System in .NET Framework****

Let us understand Common Type System (CTS) in .NET Framework with an example. What we are going to do is, we will create two applications. One Application using C# Language and the other one is using VB.NET Language. And then we will try to see the IL code of both of these applications and then we will try to see how the CTS looks like.

##### ****Understanding CTS in .NET:****

Here we are going to create two class library projects. One class library project using C# language and the other class library project using the VB language.

##### ****Creating C# Class Library Project:****

Create a class library project with the name **CsharpClassLibrary** and using the **C#** programming language. Once you create the C# class library project then add a class called **Calculator** and then copy and paste the following code in it.

**namespace** *CsharpClassLibrary*

**{**

**public** **class** Calculator

**{**

**public** **int** Calculate**()**

**{**

**int** a = 10, b = 20;

**int** c = a + b;

**return** c;

**}**

**}**

**}**

##### ****Creating VB Class Library Project:****

To the same CsharpClassLibrary solution, let us add another class library project with the name as **VBClassLibrary** but using **VB** as the programming language. Once you created the VB Class library project then add a class called **Calculator** to this project and copy and paste the following code in it.

Public Class Calculator

Public Function Calculate**()** As Integer

Dim a As Integer = 10

Dim b As Integer = 10

Dim c As Integer

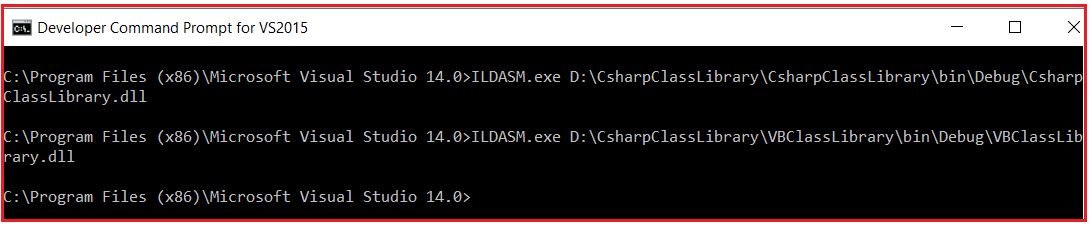
c = a + b

Return c

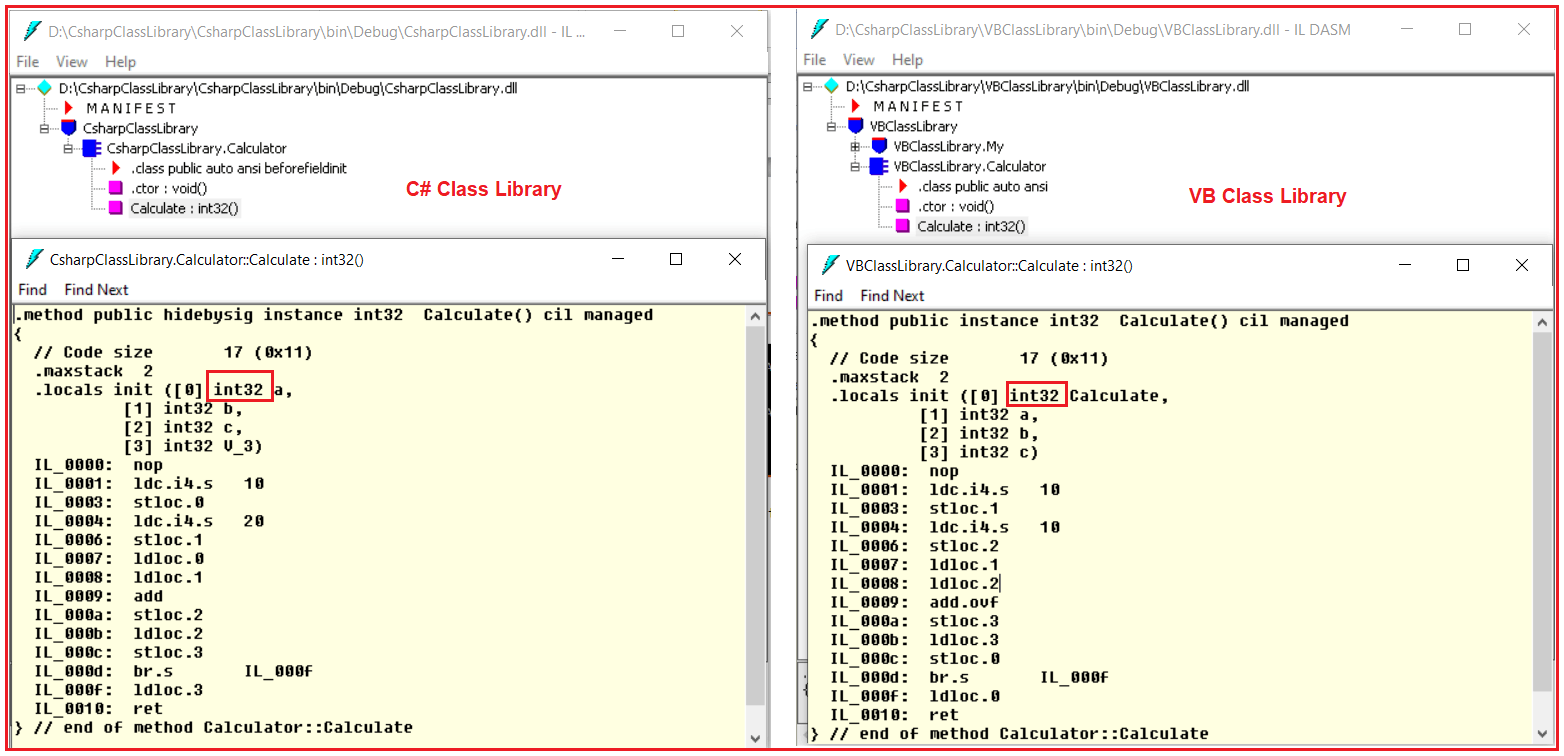
End Function

End Class

Now, build the solution which should generate the respected .dlls. In our previous article, we discussed [**how to use the ILDASM tool to see the IL code**](https://dotnettutorials.net/lesson/intermediate-language/) in detail. So, let us open Visual Studio Command Prompt in Administrative mode and run two instances of the **ILDASM.exe** command i.e. one for VB dot net DLL file and the other one for the C# DLL file as shown in the below image.



Now, let us open the IL code of the Calculate method of both the class library project as shown in the below image. Please have a look at the integer variable in the IL code which is int32 in this case. In the C# class library project, we use int as the data type to declare variables a, b and c whereas in the VB class library project we use Integer as the data type to declare the variables a, b, and c. At the end of the day, both the data types are compiled to a common data type i.e. int32.



Whether you write the code in VB.NET or in C#.NET, if it following the dot net rules or specifications, at the end of the day it is compiled into a Common Type System (CTS) in .NET Framework which is common for all .NET Supported Programming Languages.

# Common Language Specification in .NET Framework

## ****Common Language Specification (CLS) in .NET Framework****

In this article, I am going to discuss **Common Language Specification (CLS) in .NET Framework**. The Common Language Specification is also called as CLS in .NET Framework**.** Please read our previous article, where we discussed the [**Common Type System (CTS) in .NET Framework**](https://dotnettutorials.net/lesson/common-type-system/). At the end of this article, you will understand what is Common Language Specification (CLS) in C# and how CLS works in .NET Framework?

##### ****What is Common Language Specification (CLS) in .NET Framework?****

CLS (Common Language Specification) is a part of CLR in the .NET Framework. The .NET Framework supports many programming languages such as C#, VB.NET, J#, etc. Every programming language has its own syntactical rules for writing the code which is known as a language specification. One programming language syntactical rules (language specification) cannot be understood by other programming languages. But, there can be situations where we need to communicate between two different programming languages. In order to ensure smooth communication between different .NET Supported Programming Languages, the most important thing is that they should have Common Language Specifications which ensures that language specifications defined in two different languages get compiled to a Common Language Specification.

CLR in .NET Framework will execute all programming language’s code. This is possible because CLR having its own language specification (syntactical rules) which are common to all .NET Supported Programming Languages. At the time of compilation, every language compiler should follow this language specification of CLR and generate the MSIL code. This language specification of CLR is common for all programming languages and this is known as Common Language Specifications (CLS).

In order to understand this concept, what we will do here is, we will violate the common language specifications and then we will see what happens.

##### ****Example to understand CLS in .NET Framework:****

As we know C# is case sensitive whereas VB is not case sensitive. That means in C#, you can declare the same member name multiple times with case differences but it is not possible in VB. So, let us create two class library projects. One using C# and the other one is using VB Programming Language. Then we will try to consume the C# class library project in the VB class library project.

##### ****Creating a C# Class Library Project:****

Create a class library project with the name **CsharpClassLibrary** using the C# programming language. Once you create the C# class library project then add a class called **Calculator** and then copy and paste the following code in it. As you can see we have created two methods with the same name but with case differences. One method starts with capital C while the other one starts with a small c.

**namespace** *CsharpClassLibrary*

**{**

**public** **class** Calculator

**{**

**public** **int** Calculate**()**

**{**

**int** a = 10, b = 20;

**int** c = a + b;

**return** c;

**}**

**public** **int** calculate**()**

**{**

**int** a = 10, b = 20;

**int** c = a + b;

**return** c;

**}**

**}**

**}**

##### ****Creating VB Class Library Project:****

To the same CsharpClassLibrary solution, let us add another class library project with the name as VBClassLibrary but using VB as the programming language. Here, in this project we want to use the C# class library project, so first add a reference to the **CsharpClassLibrary** project. Then create a class with the name TestClass and copy-paste the following code in it.

Imports CsharpClassLibrary

Public Class TestClass

Public Sub TestMethod**()**

Dim obj As New Calculator**()**

obj.Calculate**()**

End Sub

End Class

Now, when you try to build the VB Class Library project, you will get the below error. This is because VB is not case sensitive and it found two methods with the same name. That means we are violating the Common Language Specifications in the C# code.

**‘Calculate’ is ambiguous because multiple kinds of members with this name exist in class ‘Calculator’.**

Now, let us change the second method name to Calculate2 as shown below.

**namespace** *CsharpClassLibrary*

**{**

**public** **class** Calculator

**{**

**public** **int** Calculate**()**

**{**

**int** a = 10, b = 20;

**int** c = a + b;

**return** c;

**}**

**public** **int** Calculate2**()**

**{**

**int** a = 10, b = 20;

**int** c = a + b;

**return** c;

**}**

**}**

**}**

With the above changes in place, now, build the VB class library project and the build should succeed as expected. Now, you may have one question, how to check whether the code is CLSCompliant or not.

##### ****How to check the code is CLS Compliant or not?****

In order to check whether your code is following the Common Language Specifications or not, first, you have to enable CLS Compliant in **AssemblyInfo.cs** file. So, go to the C# Class Library Project and open the **AssemblyInfo.cs** file which is present inside the **Properties** folder. Once you open the **AssemblyInfo.cs** class file, then follow the below 2 steps.

**Step1: Import the System namespace as**  
**using System;**

**Step2: Add the following CLSCompliant attribute at the bottom of this file and set its value to true**  
**[assembly: CLSCompliant(true)]**

With the above changes in place in the **AssemblyInfo.cs** file, now modify the Calculator class as shown below.

**namespace** *CsharpClassLibrary*

**{**

**public** **class** Calculator

**{**

**public** **int** Calculate**()**

**{**

**int** a = 10, b = 20;

**int** c = a + b;

**return** c;

**}**

**public** **int** calculate**()**

**{**

**int** a = 10, b = 20;

**int** c = a + b;

**return** c;

**}**

**}**

**}**

Now, when you build the C# Class Library Project you will get the following warning.

Common Language Specification (CLS) in C#.NET Framework

# Managed and Unmanaged Code in .NET

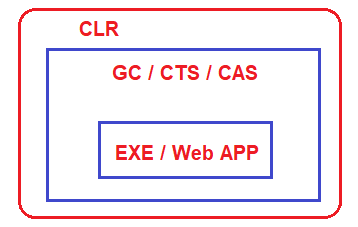
## ****Managed and Unmanaged Code in .NET****

In this article, I am going to discuss the **Managed and Unmanaged Code in C#.NET**. Please read our previous article where we discussed [**Common Language Specification (CLS)**](https://dotnettutorials.net/lesson/common-language-specification/) in detail. At the end of this article, you will understand what are Managed Code and Unmanaged code in C# and how they are executed in .NET Application in detail.

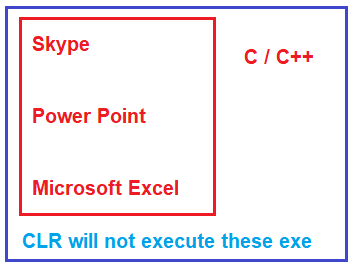
##### ****Understanding the Managed and Unmanaged Code in C#.NET Application:****

Whenever you create any exe (i.e. console application, windows application, class library project, etc.) or web application (i.e. ASP.NET MVC, Web API, ASP.NET, etc.) in .NET Framework using visual studio and using any .NET supported programming language such as C#, VB, etc., then these applications are run completely under the control of CLR (Common Language Runtime).

That means, if your applications having unused objects, then CLR will clean those objects using Garbage Collector. If your application wants to communicate with other applications, then it will make sure that CTS (Common Type System) and CLS are available. CLR uses CAS (Code Access Security) if your application has the proper rights to execute. The CLR will load your application and unload your application, etc. So, for better understanding, please have a look at the following image.



Now, let say, you have also used other third-party exe in your .NET application like Skype, PowerPoint, Microsoft Excel, etc. These “exe” are not made in dot net, they are made using other programming languages such as C, C++.



When you use these “exe” in your application, then these are not run by CLR. Even though you are running these “exe” in dot net applications, they are going to run under their own environment. For example, if one exe is developed using C or C++, then that exe will run under the C or C++ runtime environment. In the same line, if the exe is created using VB6, then it is going to run under the VB6 runtime environment.

##### ****What exactly is the managed and unmanaged code in .NET?****

The codes which run under the complete control of CLR are called Managed Code in .NET. These kinds of code (Managed code in C#) are run by dot net runtime environment. If the dot net framework is not installed or if dot net runtime is not available, then these kinds of codes are not going to be executed. CLR will provide all the facilities and features of .NET to the managed code execution like Language Interoperability, Automatic memory management, Exception handling mechanism, code access security, etc.

On the other hand, Skype, PowerPoint, Microsoft Excel does not require dot net runtime, they run under their own environment. So, in short, the code (exe, web app) which not run under the control of CLR is called unmanaged code in .NET. CLR will not provide any facilities and features of .NET to the unmanaged code in C# execution like Language Interoperability, Automatic memory management, Exception handling mechanism, code access security, etc.

# Garbage Collector in .NET Framework

## ****Garbage Collector in .NET Framework****

In this article, I am going to discuss the **Garbage Collector in .NET** Framework with Examples. Please read our previous article where we discussed [**Managed and Unmanaged Code in .NET**](https://dotnettutorials.net/lesson/managed-and-unmanaged-code/) Application. At the end of this article, you will understand what is Garbage Collector in .NET Framework and how does it work? As part of this article, we are going to discuss the following pointers in detail.

1. **What is Garbage Collector in .NET?**
2. **What are the different Generations of Garbage collectors?**
3. **How using a destructor in a class we end up in a double garbage collector loop?**
4. **How we can solve the double loop problems using finalize dispose patterns?**

##### ****What is Garbage Collector in .NET Application?****

When a dot net application runs, lots of objects are created. At a given point in time, it is possible that some of those objects are not used by the application. Garbage Collector in .NET Framework is nothing but is a Small Routine or you can say it’s a Background Process Thread that runs periodically and try to identify what objects are not being used currently by the application and de-allocates the memory of those objects.

So, Garbage Collector is nothing but, it is a feature provided by CLR which helps us to clean or destroy unused managed objects. By cleaning or destroying those unused managed objects, basically reclaims the memory.

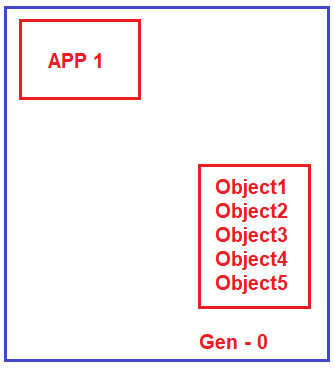
**Note:** The Garbage Collector will destroy only the unused managed objects. It does not clean unmanaged objects. If you want to learn what exactly is managed and unmanaged objects, please read our previous article.

##### ****Garbage Collector Generations in .NET:****

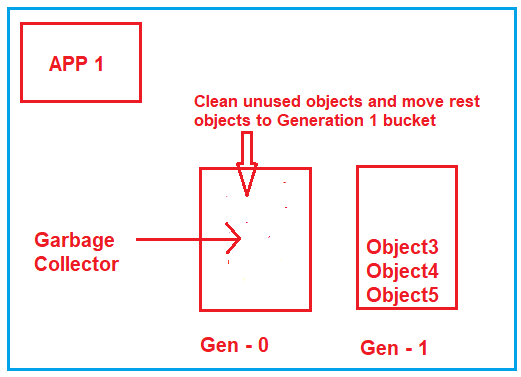
Let us understand what Garbage Collector Generations are and how does it affect Garbage Collector performance? There are three generations. They are Generation 0, Generation 1, and Generation 2.

##### ****Understanding Generation 0, 1, and 2:****

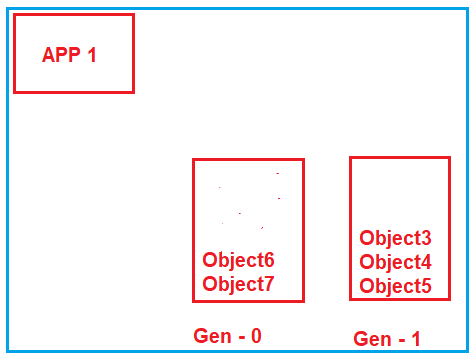
Let say you have a simple application called App1. As soon as the application started it creates 5 managed objects. Whenever any new objects (fresh objects) are created, they are moved into a bucket called Generation 0. For better understanding please have a look at the following image.



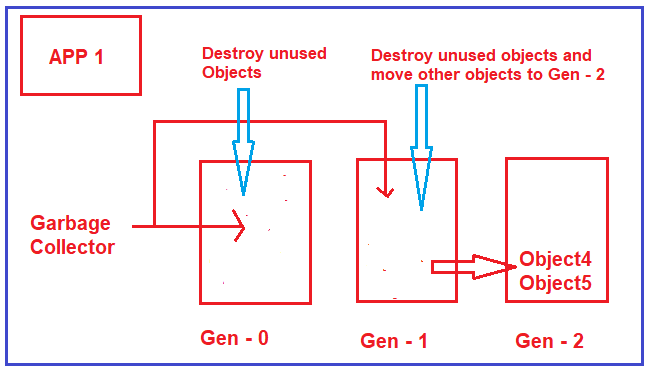
We know our hero Mr. Garbage Collector runs continuously as a background process thread to check whether there are any unused managed objects so that it reclaims the memory by cleaning those objects. Now, let say two objects (Object1 and Object2) are not needed by the application. So, Garbage Collector will destroy these two objects (Object1 and Object2) and reclaims the memory from Generation 0 bucket. But the remaining three objects (Object3, Object4, and Object5) are still needed by the application. So, the Garbage collector will not clean those three objects. What Garbage Collector will do is, he will move those three managed objects (Object3, Object4, and Object5) to Generation 1 bucket as shown in the below image.



Now, let say your application creates two more fresh objects (Object6 and Object7). As fresh objects, they should be created in Generation 0 bucket as shown in the below image.



Now, again Garbage Collector runs and it comes to Generation 0 bucket and checks which objects are used. Let say both objects (Object6 and Object7) are unused by the application, so it will remove both the objects and reclaims the memory. Now, it goes to the Generation 1 bucket, and checks which object are unused. Let say Object4 and Object5 are still needed by the application while object3 is not needed. So, what Garbage Collector will do is, it will destroy Object3 and reclaims the memory as well as it will move Objec4 and Object5 to Generation 3 bucket which is shown in the below image.



##### ****What are Generations?****

Generations are nothing but, will define how long the objects are staying in the memory. Now the question that should come to your mind is why do we need Generations?

##### ****Why do we need Generations?****

Normally, when we are working with big applications, they can create thousands of objects. So, for each of these objects, if the garbage collector goes and checks if they are needed or not, it’s really pain or it’s a bulky process. By creating such generations what it means if an object in Generation 2 buckets it means the Garbage Collector will do fewer visits to this bucket. The reason is, if an object move to Generation 2, it means it will stay more time in the memory. It’s no point going and checking them again and again.

So, in simple words, we can say that Generations 0, 1, and 2 will helps to increase the performance of the Garbage Collector. The more the objects in Gen 0, the better the performance and the more the memory will be utilized in an optimal manner.

##### ****How using a destructor in a class we end up in a double garbage collector loop?****

As we already discussed garbage collectors will only clean up the managed code. In other words, for any kind of unmanaged code, for those codes to clean up has to be provided by unmanaged code, the garbage collector does not have any control over them to clean up the memory.

For example, let say you have a class called MyClass in VB6, then you have to expose some function let say CLeanUp() and in that function, you have to write the logic to clean up the unmanaged code. From your dot net code, you simply need to call that method (CLeanUp()) to initiate the clean-up.

The point, or the section from where you would like to call the Clean-Up is the destructor of a class. This looks to be the best place to write the clean-up code. But, there is a big problem associated with it when you write clean-up in a destructor. Let us understand what the problem is?

When you define a destructor in your class, the Garbage Collector before destroying the object, will go and ask the question to the class, do you have a destructor, if you have a destructor, then move the object to the next generation bucket. In other words, it will not clean up the object having a destructor at that moment itself even though it is not used. So, it will wait for the destructor to run, and then it will go and clean up the object. Because of this, you will find more objects in generation 1 and Generation 2 as compared to Generation 0.

##### ****Example: Using Destructor****

Please create a console application and then copy and paste the following code in it in the Program class.

**using** *System;*

**namespace** *GCDemo*

**{**

**class** Program

**{**

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

**{**

**for(int** i = 0; i **<**= 1000000; i++**)**

**{**

MyClass obj = new MyClass**()**;

**}**

Console.Read**()**;

**}**

**}**

**public** **class** MyClass

**{**

~MyClass**()**

**{**

//Unmanaged code clean up

**}**

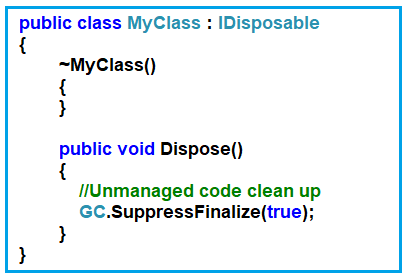
**}**

**}**

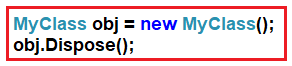
So, if you writing the clean-up code in your destructor, then you will end up creating more objects in Generation 1 and Generation 2 which means you are not utilizing the memory properly.

##### ****How to Overcome the above Problem?****

This problem can be overcome by using something called finalize dispose pattern. In order to implement this, your class should implement the IDisposable interface and provide the implementation for the Dispose method. Within the Dispose method, you need to write the clean-up code for unmanaged objects and in the end, you need to call GC.SuppressFinalize(true) method by passing true as the input value. This method tells suppress any kind of destructor and just go and clean up the objects. For better understanding, please have a look at the following image.



Once you have used to object, then you need to call the Dispose method so that the double garbage collector loop will not happen as shown below.



##### ****The complete code is given below.****

**using** *System;*

**namespace** *GCDemo*

**{**

**class** Program

**{**

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

**{**

**for(int** i = 0; i **<**= 1000000; i++**)**

**{**

MyClass obj = new MyClass**()**;

obj.Dispose**()**;

**}**

Console.Read**()**;

**}**

**}**

**public** **class** MyClass : IDisposable

**{**

~MyClass**()**

**{**

**}**

**public** **void** Dispose**()**

**{**

//Unmanaged code clean up

GC.SuppressFinalize**(true)**;

**}**

**}**

**}**

Now, the question that should come to your mind is why the destructor is there? The reason is as a developer you may forget to call the Dispose method once you use the object. In that case, the destructor will invoke and it will go and clean up the object.

# Assembly DLL EXE in .NET Framework

## ****Assembly DLL EXE in .NET Framework****

In this article, I am going to discuss **Assembly DLL and EXE** **in .NET Framework**with examples. Please read our previous article where we discussed the [**Garbage Collector in .NET**](https://dotnettutorials.net/lesson/garbage-collector/) Application. As part of this article, we are going to discuss the following pointers in detail.

1. **What is an Assembly in .NET?**
2. **Types of Assemblies of in .NET Framework.**
3. **Understanding DLL and EXE.**
4. **What is the difference between the DLL and the EXE in .NET Framework?**

##### ****What is an Assembly in .NET?****

According to MSDN, Assemblies are the building block of .NET Framework applications; they form the fundamental unit of deployment. In simple words, we can say that Assembly is nothing but a precompiled .NET Code that can be run by CLR (Common Language Runtime).

Let us understand the above definition with an example. In order to understand this, let us create a simple console application with the name MyConsoleApp. Once you created the console application then please modify the Program class as shown below.

**using** *System;*

**namespace** *MyConsoleApp*

**{**

**class** Program

**{**

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

**{**

Console.WriteLine**(**"This is From Console App"**)**;

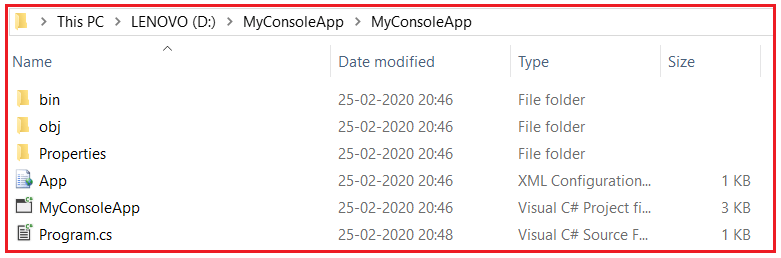
Console.ReadKey**()**;

**}**

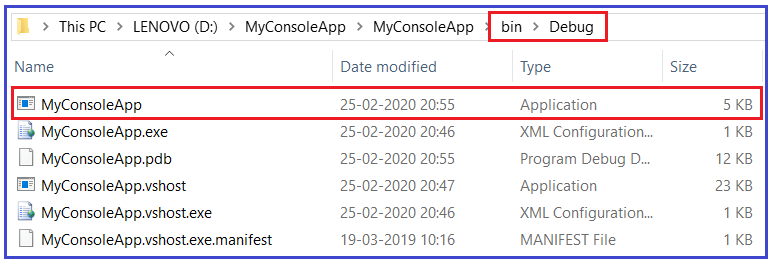
**}**

**}**

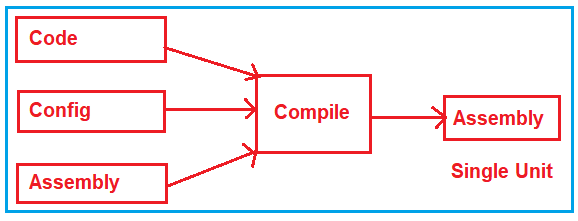
Now, if you right-click on your project and click on Open Folder in File Explorer, then you will find lots of things (Source code i.e. Program.cs class file, Configuration file i.e. App, Properties folder which contains AssemblyInfo.cs class file, etc.) as shown in the below image.



But when you build the application, then it will put the whole thing into a single EXE as shown in the below image. You can find this file under the **bin => Debug** folder. You can copy this single unit i.e. MyConsoleApp.exe and put it anywhere within your computer and from there you can run it.



So, an assembly is nothing but a single unit of deployment or it is a precompiled chunk of code that can be executed by CLR. For better understanding please have a look at the following diagram.



##### ****Types of Assemblies of in .NET Framework:****

In the .NET Framework, there are two types of assemblies. They are as follows:

1. **EXE (Executable)**
2. **DLL (Dynamic Link Library)**

In .NET Framework when we compile a Console Application or a Windows Application, it generates EXE, whereas when we compile a Class Library Project or ASP.NET web application, then it generates DLL. In.NET framework, both EXE, and DLL are called assemblies.

##### ****Understanding DLL and EXE in .NET Framework:****

We already created one console application and we already see that it creates an EXE. Let us see an example of DLL. In order to create a DLL, let us add a class library project to the same solution with the name as MyClassLibrary. Once you created the class library project, it will by default create a class file with the name Class1. Let us modify Class1 as shown below.

**namespace** *MyClassLibrary*

**{**

**public** **class** Class1

**{**

**public** string GetData**()**

**{**

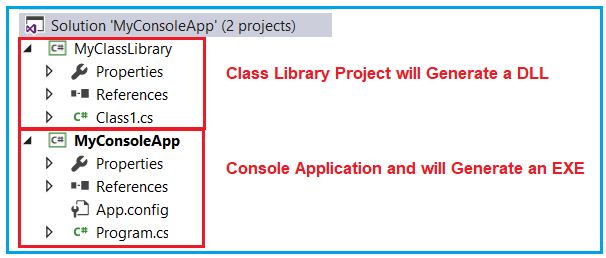
**return** "This is from Class Library";

**}**

**}**

**}**

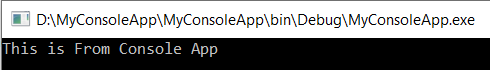
With this, now our solution contains two projects. One is a console application and the other one is a class library project as shown below.



Now, build the solution and you should get the respected assemblies as expected in their own bin => Debug folder. Now, the question that should come to your mind is what is the difference between the DLL and the EXE?

##### ****What is the difference between the DLL and the EXE in .NET Framework?****

The EXE is run in its own address space or in its own memory space. If you double click on the MyConsoleApp EXE then you will get the following output. Now, this program is running out of its own memory space.



Without closing this window, again if you double click on the MyConsoleApp EXE, again it will run and will display the same output. This is because now, both the EXE are running in their own memory space. The point that you need to remember is EXE is an executable file and can run by itself as an application.

Coming to DLL, it cannot be run by itself like EXE. That means **the MyClassLibrary.dll** cannot be invoked or run by himself. It needs a consumer who is going to invoke it. So, a DLL is run inside another memory space. The other memory space can be a console, or windows applications, or web applications that should have their own memory space.

For example, you can invoke the DLL from a console application. We have a console called MyConsoleApp and let’s see how to invoke the MyClassLibrary.dll from this console application. In order to use the MyClassLibrary.dll inside the MyConsoleApp, first, you need to make a reference to that DLL. Once you add a reference to MyClassLibrary DLL, and then please modify the Program class of Console Application as shown below.

**using** *System;*

**using** *MyClassLibrary;*

**namespace** *MyConsoleApp*

**{**

**class** Program

**{**

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

**{**

//Using MyClassLibrary DLL

Class1 obj = new Class1**()**;

Console.WriteLine**(**obj.GetData**())**;

Console.WriteLine**(**"This is From Console App"**)**;

Console.ReadKey**()**;

**}**

**}**

**}**

Now, run the application and you should see the following output. Here, the MyClassLibrary DLL is run inside the MyConsoleApp address space.

why do we need DLLs

So, in short, the difference between them is an EXE is an executable file and can run by itself as an application whereas DLL is usually consumed by an EXE or by another DLL and we cannot run or execute DLL directly.

Now, the question that should come to your mind why do we need DLLs as it is not invoked by itself. The reason behind the DLL is reusability. Suppose you want some class, or logic, or something else in many applications, then simply put those classes, logic inside a DLL, and refer that DLL wherever it is required.

# App Domain in .NET Framework

In this article, I am going to discuss **App Domain in .NET Framework** and in what scenarios we need them with examples. Please read our previous article where we discussed [**Assembly, DLL, and EXE**](https://dotnettutorials.net/lesson/assembly-dll-exe/) in detail. The App Domain (Application Domain) in the .NET Framework is a logically isolated container inside which the .NET Code runs. At the end of this article, you will understand what is App Domain and how to create a custom app domain in C# with examples.

##### ****Understanding App Domain in .NET:****

Let us understand App Domain in C# with an example. Please create a console application and then copy and paste the following code in Program.cs class file. This is a very simple application. Here we created two classes MyClass1 and MyClass2. Then we create objects of both these classes inside the Main method of the Program class and loading these two classes into the console application.

**using** *System;*

**namespace** *AppDomainDemo*

**{**

**class** Program

**{**

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

**{**

MyClass1 obj1 = new MyClass1**()**;

MyClass2 obj2 = new MyClass2**()**;

Console.Read**()**;

**}**

**}**

**public** **class** MyClass1

**{**

**}**

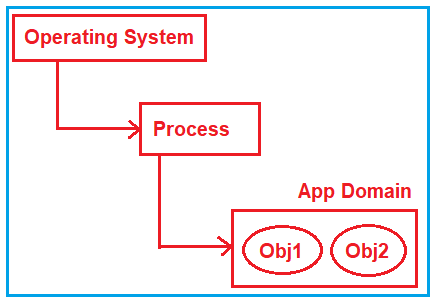
**public** **class** MyClass2

**{**

**}**

**}**

Now when you run the above application or EXE what will happen internally let us discuss. Here, the EXE runs as a Process inside the Operating System. Inside the Process, we have one App Domain by default loaded, and inside that App Domain both the objects (obj1 and obj2) are running as shown in the below image.

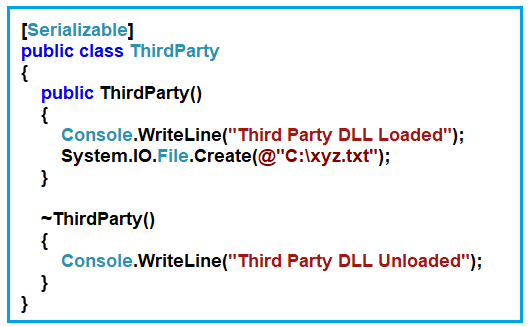


**Note:** By default always there is an App Domain under which our .NET Code runs.

##### ****Need for App Domain in .NET Application:****

Let us understand the need for Add Domain in .NET Application. Suppose you want to consume a Third Party DLL. That DLL you may get from the Internet or from any other third parties. Here, you have one doubt i.e. the Third Party DLL access your C:\ Drive. Suppose you want to use the Third DLL which you download from internet for reporting purpose, but there is some kind of virus which create a file in your C:/ Drive instead of working as a reporting tool.

Here, we are not downloading any DLL from the internet, instead, we will create a class as shown below which will act as the Third Party DLL.



Now, if you simply use the ThirdParty class with the default App Domain, then it can have access to your C:\ Drive. Let’s modify the Program.cs class file as shown below.

**using** *System;*

**namespace** *AppDomainDemo*

**{**

**class** Program

**{**

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

**{**

//Third Party DLL

ThirdParty Obj3 = new ThirdParty**()**;

//Own DLL

MyClass1 obj1 = new MyClass1**()**;

MyClass2 obj2 = new MyClass2**()**;

Console.Read**()**;

**}**

**}**

**[**Serializable**]**

**public** **class** ThirdParty

**{**

**public** ThirdParty**()**

**{**

Console.WriteLine**(**"Third Party DLL Loaded"**)**;

System.IO.File.Create**(**@"C:\xyz.txt"**)**;

**}**

~ThirdParty**()**

**{**

Console.WriteLine**(**"Third Party DLL Unloaded"**)**;

**}**

**}**

**public** **class** MyClass1

**{**

**}**

**public** **class** MyClass2

**{**

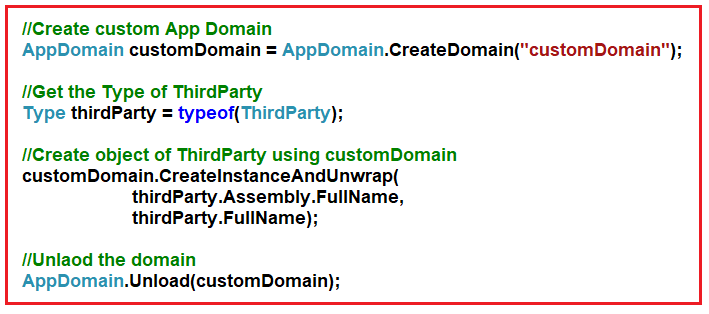
**}**

**}**

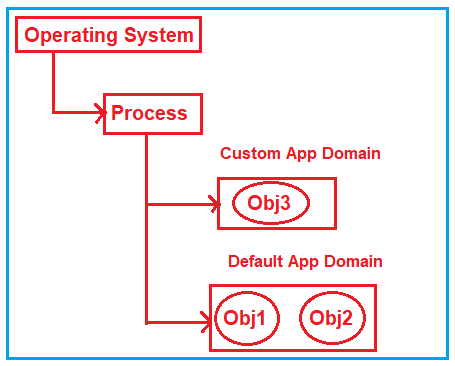
Now, when you execute the above code it will create the text file in the C Drive. But we want to restrict the third-party DLL to access our C drive. We can do this by creating a separate App Domain for the Third Party DLL and then we will provide settings to that App Domain so that, it will not access our C Drive.

##### ****How to Create a Custom App Domain in .NET?****

Let us see how to create our own App Domain and also see how we will run the Third Party DLL inside that App Domain. Then we will see how to provide permission to restrict access to the C Drive. Please have a look at the following image which shows how to create a custom app domain in C#. The code is self-explain, please go through the comment line.



Once you understood how to create a custom app domain in C#. Let us see what we want to do. We want to execute the Third Party DLLs using a custom App Domain while our classes we want to execute inside the default app domain which is shown in the below image.



The complete code to implement the above requirement is given below.

**using** *System;*

**namespace** *AppDomainDemo*

**{**

**class** Program

**{**

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

**{**

//Create custom App Domain

AppDomain customDomain = AppDomain.CreateDomain**(**"customDomain"**)**;

//Get the Type of ThirdParty

Type thirdParty = typeof**(**ThirdParty**)**;

//Create object of ThirdParty using customDomain

customDomain.CreateInstanceAndUnwrap**(**

thirdParty.Assembly.FullName,

thirdParty.FullName**)**;

//Unload the domain

AppDomain.Unload**(**customDomain**)**;

//Own DLL

MyClass1 obj1 = new MyClass1**()**;

MyClass2 obj2 = new MyClass2**()**;

Console.Read**()**;

**}**

**}**

**[**Serializable**]**

**public** **class** ThirdParty

**{**

**public** ThirdParty**()**

**{**

Console.WriteLine**(**"Third Party DLL Loaded"**)**;

System.IO.File.Create**(**@"C:\xyz.txt"**)**;

**}**

~ThirdParty**()**

**{**

Console.WriteLine**(**"Third Party DLL Unloaded"**)**;

**}**

**}**

**public** **class** MyClass1

**{**

**}**

**public** **class** MyClass2

**{**

**}**

**}**

Now if you execute, then it will also create the text file in the C Drive. This is because we have run the Third Party DLL using a custom app domain but till now we have not written any logic to restrict access to C Drive.

##### ****How to Restrict Access a Custom App Domain to C Drive?****

Let us see how to restrict the custom app domain to access our C Drive. In order to restrict the custom app domain to access C drive, we need to create a permission object and restrict No Access to C Drive and then create a setup for the app domain and finally, we need to use both permissions and set up while creating the custom app domain. The complete code is given below and the code is self-explain so, please go through the comment lines.

**using** *System;*

**using** *System.Security;*

**using** *System.Security.Permissions;*

**namespace** *AppDomainDemo*

**{**

**class** Program

**{**

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

**{**

//Create Permission object

var permission = new PermissionSet**(**PermissionState.None**)**;

permission.AddPermission**(**

new SecurityPermission**(**SecurityPermissionFlag.Execution**)**

**)**;

//Set No Access to C drive

permission.AddPermission**(**

new FileIOPermission**(**FileIOPermissionAccess.NoAccess, @"C:\")

);

//Create setup for App Domain

var setUp = new AppDomainSetup();

setUp.ApplicationBase = AppDomain.CurrentDomain.SetupInformation.ApplicationBase;

//Create custom App Domain using the setup and permission

AppDomain customDomain = AppDomain.CreateDomain("customDomain", null, setUp, permission);

//Get the Type of ThirdParty

Type thirdParty = typeof(ThirdParty);

//Create object of ThirdParty using customDomain

customDomain.CreateInstanceAndUnwrap(

thirdParty.Assembly.FullName,

thirdParty.FullName);

//Unload the domain

AppDomain.Unload(customDomain);

//Own DLL

MyClass1 obj1 = new MyClass1();

MyClass2 obj2 = new MyClass2();

Console.Read();

}

}

[Serializable]

public class ThirdParty

{

public ThirdParty()

{

Console.WriteLine("Third Party DLL Loaded");

System.IO.File.Create(@"C:\xyz.txt");

}

~ThirdParty()

{

Console.WriteLine("Third Party DLL Unloaded"**)**;

**}**

**}**

**public** **class** MyClass1

**{**

**}**

**public** **class** MyClass2

**{**

**}**

**}**

Now when you execute the above application, the line from which it will try to access and create a file in the C drive will through you an exception. But irrespective of the exception in the custom app domain, if you want to execute the default app domain, then you need to put the logic of the custom app domain inside the try-catch as shown in the below code.

**using** *System;*

**using** *System.Security;*

**using** *System.Security.Permissions;*

**namespace** *AppDomainDemo*

**{**

**class** Program

**{**

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

**{**

//Create Permission object

var permission = new PermissionSet**(**PermissionState.None**)**;

permission.AddPermission**(**

new SecurityPermission**(**SecurityPermissionFlag.Execution**)**

**)**;

//Set No Access to C drive

permission.AddPermission**(**

new FileIOPermission**(**FileIOPermissionAccess.NoAccess, @"C:\")

);

//Create setup for App Domain

var setUp = new AppDomainSetup();

setUp.ApplicationBase = AppDomain.CurrentDomain.SetupInformation.ApplicationBase;

//Create custom App Domain using the setup and permission

AppDomain customDomain = AppDomain.CreateDomain("customDomain", null, setUp, permission);

try

{

//Get the Type of ThirdParty

Type thirdParty = typeof(ThirdParty);

//Create object of ThirdParty using customDomain

customDomain.CreateInstanceAndUnwrap(

thirdParty.Assembly.FullName,

thirdParty.FullName);

}

catch(Exception ex)

{

//Unload the domain

AppDomain.Unload(customDomain);

}

//Own DLL

MyClass1 obj1 = new MyClass1();

MyClass2 obj2 = new MyClass2();

Console.Read();

}

}

[Serializable]

public class ThirdParty

{

public ThirdParty()

{

Console.WriteLine("Third Party DLL Loaded");

System.IO.File.Create(@"C:\xyz.txt");

}

~ThirdParty()

{

Console.WriteLine("Third Party DLL Unloaded"**)**;

**}**

**}**

**public** **class** MyClass1

**{**

**}**

**public** **class** MyClass2

**{**

**}**

**}**

##### ****Advantages of using App Domain in .NET Application:****

The App Domain (Application Domain) is a logically isolated container inside a process. In this logical isolation, you can load and run .NET Code in an isolated manner. The following are the advantages of using the App Domain.

1. You can load and unload DLL inside these logical containers without one container affecting the other. So, if there are issues in one application domain you can unload that application domain, and the other application domain work without issues.
2. If you have a Third Party DLL and for some reason, you don’t trust the third-party code. You can run that DLL in an isolated app domain with fewer privileges. For example, you can say that the DLL cannot access your “C:\” drive. And other DLLs that you trust you can run with full privilege in a different app domain.
3. You can run different versions of DLL in every application domain.

# Strong and Weak Assemblies in .NET

## ****Strong and Weak Assemblies in .NET Framework****

In this article, I am going to discuss **Strong and Weak Assemblies in .NET Framework** with examples. Please read our previous article where we discussed [**App Domain in .NET**](https://dotnettutorials.net/lesson/app-domain-dot-net-framework/) Application. In .NET Framework, the assemblies are broadly classified into 2 types. They are as follows:

1. **Weak Named Assemblies**
2. **Strong Named Assemblies**

Let us first understand what assembly is then we will discuss strong and week assembly and the difference between them.

##### ****Understanding Assembly in .NET Framework:****

Let us create a simple console application with the name **AssemblyDemo** and then modify the **Program** class as shown below. This is a very simple C# program, simply printing a message “**Hello world**” to the console. In order to print the message on the Console, here we are using the **Console** class. That **Console** class is coming from the **System** namespace. And the **System** namespace is present in the **System** Assembly. The System assembly is a .NET Framework assembly

**using** *System;*

**namespace** *AssemblyDemo*

**{**

**class** Program

**{**

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

**{**

Console.WriteLine**(**"Hello world"**)**;

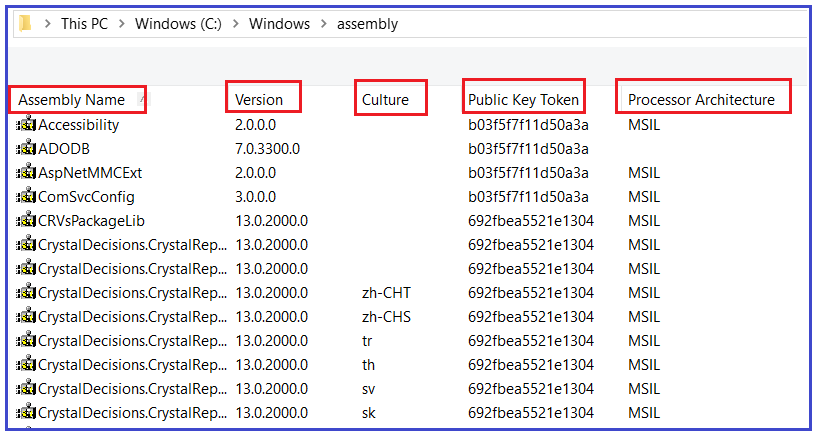
Console.ReadKey**()**;

**}**

**}**

**}**

When we installed .NET in the machine, two important components get installed. One is the .NET Framework Base Class Library (BCL) and the other is CLR which is nothing but the runtime environment. In .NET Framework Base Class Library, we have several assemblies. All the .NET Framework assemblies are installed in a special location called GAC (Global Assembly Cache). The location for GAC is “**C:\Windows\assembly**”. Once you go to this location, you will find all the .NET Framework assemblies as shown in the below image. We will discuss GAC in detail in our upcoming article.



All the assemblies present in GAC are strongly typed. Later part of this article we will discuss what exactly a strong type assembly is and the difference between a week and a strong type assembly in the .NET. In .NET, an assembly is consists of 4 Parts

1. Simple textual name (i.e. the Assembly name).
2. The Version number.
3. Culture information (If provided, otherwise the assembly is language-neutral)
4. Public key token

Let us discuss each part of an assembly in detail.

##### ****Assembly Name (Simple Textual Name):****

This is nothing but the project name. We just created one console application with the name **AssemblyDemo**. Now build the project and go to the **Bin => Debug** folder which you can find inside the project and you should find an assembly with the name **AssemblyDemo.**

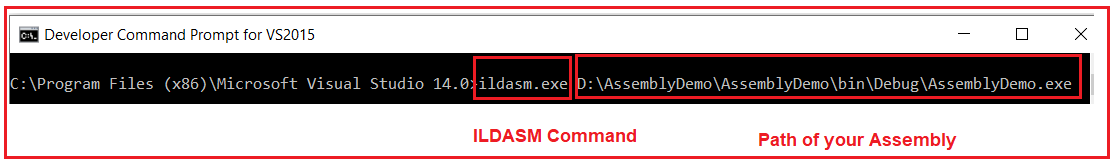
##### ****Version Number:****

The default format of the Version number is 1.0.0.0. That means the version number again consists of four parts as follows:

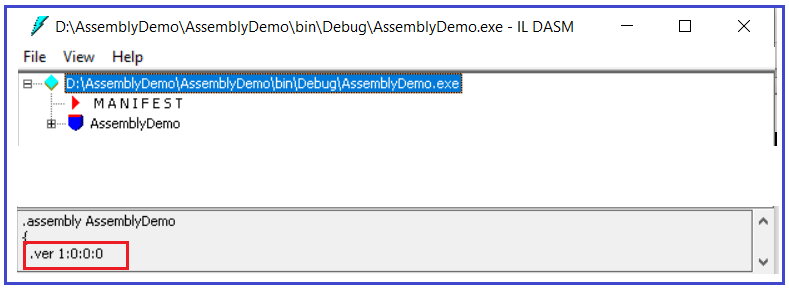
1. **Major Version**
2. **Minor Version**
3. **Build Number**
4. **Revision Number**

Typically, any software we develop will go under a change over a period of time. When we fix bugs or add new features, depending on the significance of the change, we either change the major number or the minor number. If the changes we are making to the application are huge, then probably we change the major number else we will change the minor number. Most of the time the build number and revision number have defaulted.

In order to see the Version Number of your assembly “**AssemblyDemo**”, Open Visual Studio Developer command prompt and use the **ILDASM** command to see the version number as shown below.



Once you use the ILDASM command followed by the physical path of your assembly and hit the enter key you will get the following ILDASM window and just look at the version number which you can find at the bottom of the window.



##### ****How to change the Version Number of an Assembly in .NET?****

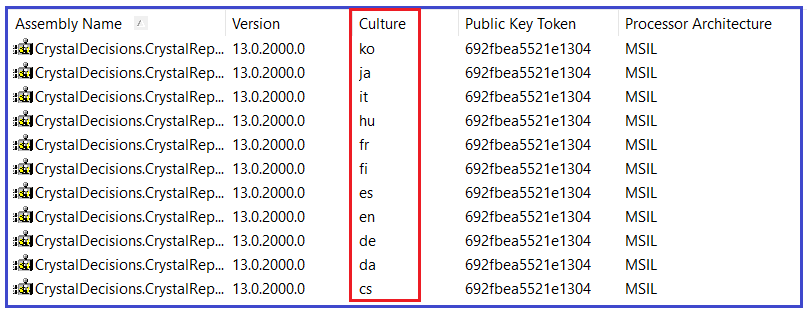
If you want to change the version number of your assembly, then you need to use the **AssemblyVersion** attribute within the **AssemblyInfo** class which is present inside the **Properties** folder of your project. You can specify all the values or you can default the Revision and Build Numbers by using the ‘\*’. Suppose, you want to change the major number to 3 and the minor number to 2, then you need to change the **AssemblyVersion** attribute as shown below in the **AssemblyInfo** class.

**[assembly: AssemblyVersion(“3.2.\*”)]**

With the above changes in place, now if you build the solution and check the version number using the ILDASM tool, then you should get the updated version number. Please read our [**ILDASM and ILASM articles**](https://dotnettutorials.net/lesson/intermediate-language/) to learn more about ILDASM and ILASM.

##### ****Assembly Culture:****

The AssemblyCulture attribute is used for specifying the culture of the assembly. By default in .NET assemblies are language-neutral which means the AssemblyCulture attribute contains an empty string. If you go to the GAC, then you will find most of the assemblies are culture neutral. But there could be some assemblies that are culture-specific. For a better understand, please have a look at the following image which you can also find in your GAC. The following assemblies are specific to the language specified in the Culture attribute.

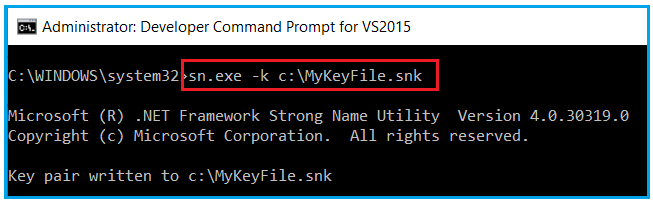


When you specify the culture, then that assembly becomes a satellite assembly. We will discuss satellite assemblies in detail in our upcoming article. If you want to specify the culture then you need to use the AssemblyCulture attribute with the AssemblyInfo.cs class file. For example, if you want to specify English as the culture then you need to use the AssemblyCulture attribute as shown below.

**[assembly: AssemblyCulture(“en”)]**

##### ****Public Key Token:****

If you go to the GAC, then you will see that every assembly is assigned with a public key token. In order to get the public key token, you need to sign your assembly with a private and public key pair. Now the question is how do I get the private-public key. In the .NET framework, we have a tool called a strong naming tool and you can use this tool to generate the key pair. Again in order to use this tool you need to use the Developer Command Prompt for Visual studio. So, open Developer Command Prompt for Visual Studio in administrator mode and then type **sn.exe -k c:\MyKeyFile.snk**and press enter as shown in the below image.



Once you type the required command and press enter, the key file with the name **MyKeyFile.snk** should be generated in the C: drive. In SN.exe, SN stands for Strong Name.

Once you generated the Key file, then you need to use the AssemblyKeyFile attribute of the AssemblyInfo class to sign the assembly with a strong name. To the constructor of the AssemblyKeyFile attribute, you need to pass the path of the key file that contains the private and public key as shown below.

**[assembly: AssemblyKeyFile(“C:\\MyKeyFile.snk”)]**

Once you add the above AssemblyKeyFile attribute, build the solution. Once you build the solution, now your assembly sign with a private-public key pair. Now, our assembly has all the four components such as Name, Version Number, Culture, and Public Key Token.

##### ****Strong Name Assembly in .NET Framework:****

An assembly is said to be strongly named assembly when it has the following properties

1. **The assembly name.**
2. **Version number.**
3. **The assembly should have been signed with the private/public key pair.**

##### ****What is the difference between Strong and Weak Assemblies in .NET Framework?****

If an assembly is not signed with the private/public key pair then the assembly is said to be a weak named assembly and it is not guaranteed to be unique and may cause the DLL hell problem. The Strong named assemblies are guaranteed to be unique and solve the DLL hell problem. Again, you cannot install an assembly into GAC unless the assembly is strongly named.