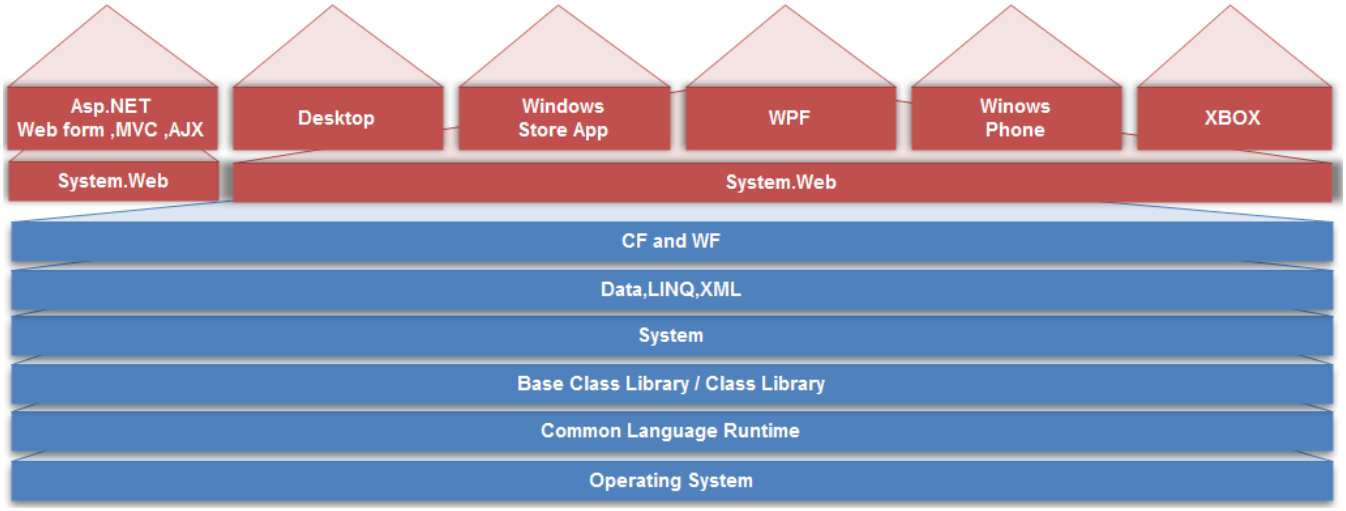
CLR, BCL, CIL and assemblies in .Net framework

Vahid Cheshmi
Jan 8 · 8 min read

This article is about about Base Class Library (BCL), Common Language Runtime (CLR), and Common Intermediate Language (CIL) and .NET assemblies.

Beforehand let's take a look at Framework Class Library (FCL) in which you can see so many other classes, functionalities and other services and this article covers Base Class Library and Common Language Runtime.



Framework Class Library(FCL)

Introducing the Building Blocks of the .NET Platform (the CLR, CTS, and CLS)

Now, you've understood some benefits provided by .NET, I'm going to describe three key topics that will help you to understand key benefits better,

CLR (Common Language Runtime)

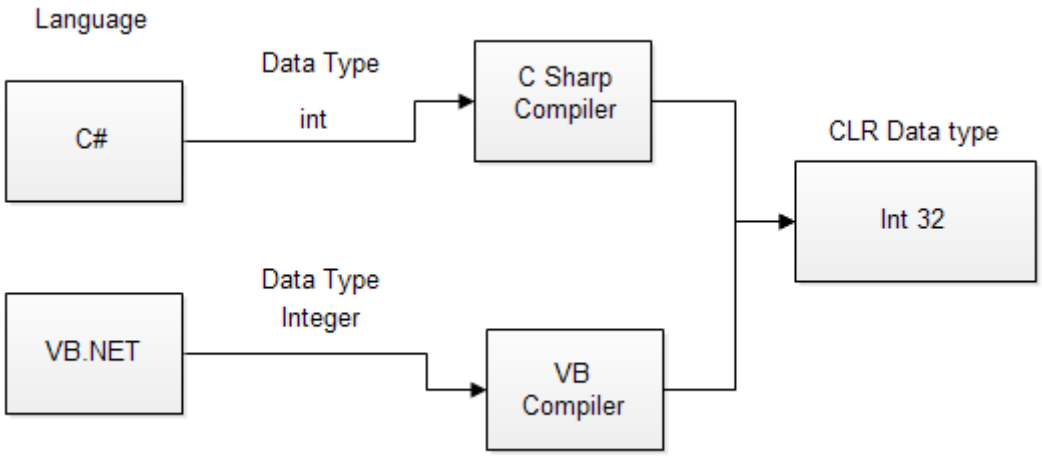
As a Programmer point of view, .NET could be a runtime environment and a comprehensive base class library.

The runtime is referred to as Common Language Runtime (CLR).the main goal of CLR is to load, Locate and manage .NET objects on programmer's behalf.it means that CLR always handle these operations and a programmer doesn't interfere to these processes. Also CLR take care memory management, application hosting, some security checks and coordinating threads.

CTS (Common Type System)

The CTS describes all data types and all related constructs which supported by runtime, details how they would be represented in the .NET metadata format and also it specifies how entities can interact with each other. In other words we have several languages and each language has own data type definition and cannot be understandable by other language but CLR can understand all the data types.

For example C# has int Data Type and VB.NET has Integer Data Type.so after compilation it uses the same structure as Int32 in CTS.



Common Language Runtime (CLR)

CLS (Common Language Specification)

CLS is a subset of CTS, It defines a set of rules and restrictions that every language must follow which runs under.NET framework.

For example imagine you develop and application by C# and VB.NET, in C# every statement must have to end with semicolon otherwise in VB.NET shouldn't be written like that.

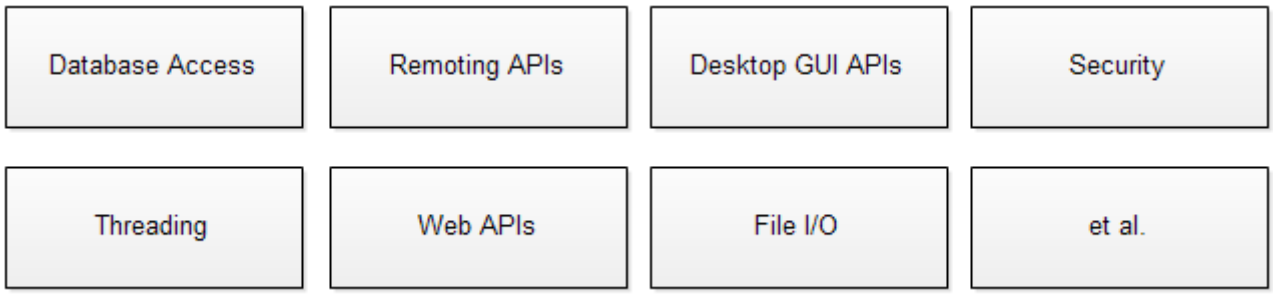
So these syntax rules are different from languages to languages but CLR can understand all languages syntax because in .NET each language is converted into MSIL code after compilation and MSIL code is language specification of CLR.

Base Class Library (BCL)

Stands for Base class library also known as Class library (CL). BCL is a subset of Framework class library (FCL). Class library is the collection of reusable types that are closely integrated with CLR. Base Class library provides classes and types that are helpful in performing day to day operation e.g. dealing with string and primitive types, database connection, IO operations.

BCL define types that can be used to build any type of software application.

You can use ASP.NET to build web sites and REST services, WCF to build distributed systems, WPF to build desktop GUI applications, and so on. As well, the base class libraries provide types to interact with the directory and file system on a given computer, communicate with relational databases (via ADO.NET), and so on.



Base Class Library (BCL)

Managed and Unmanaged Code

Now that you've found out the definition about BCL and CLR I'm going to talk about Managed and unmanaged codes.

C# language can be used only to build software that is hosted under .NET runtime. The term used to describe the code targeting the .NET runtime is managed code. There is a binary unit that contains the managed code which is called assembly.in contrast code that cannot be directly hosted by the .ET runtime is called unmanaged code.

As I mentioned in previous article the .NET platform can run on variety of operating systems.it is possible to build a C# application on a Windows operating system using

Visual Studio and run the program on a macOS machine using the .NET Core runtime. Also, you can build a C# application on Linux operating system using Xamarin Studio and run the program on Windows, macOS and so on. The latest release of Visual Studio, I mean Visual Studio 2017, you can also build .NET Core applications on a Mac to be run on Windows, macOS or Linux.it's possible to build, deploy, and run .NET programs on a on different operating systems.

An overview of .NET assemblies

Regardless of which .NET languages you chose to develop, after compiling your application there would be two types of assemblies , Executing files *.exe and Dynamic Link Library *.dll.

But when we compile our .NET program it's not converted into the executable binary code but to an intermediate code, called MSIL or IL, which is understandable by CLR. MSIL is an OS and H/W independent code. When the program needs to be executed, this MSIL or intermediate code is converted to binary executable code, called native code.

All the .NET assemblies contain the definition of types, versioning information for the type meta-data, and manifest and in this article I'm going to talk about it more.

If you have a class named Customer, the type metadata describes details such as Customer's base class, specifies which interfaces are implemented by Customer class and also give some information of each member in Customer class..NET metadata always present within an assembly and it's .NET duty to automatically generate it.

CIL and type metadata, assemblies themselves are also described using metadata, which is called*manifest*. The manifest contains information about the current version of the assembly, and a list of all externally referenced assemblies that are required for proper execution.so in the following article I'm going to use a tool to examine an assembly's type, metadata, and manifest.

The role of CIL in .NET framework

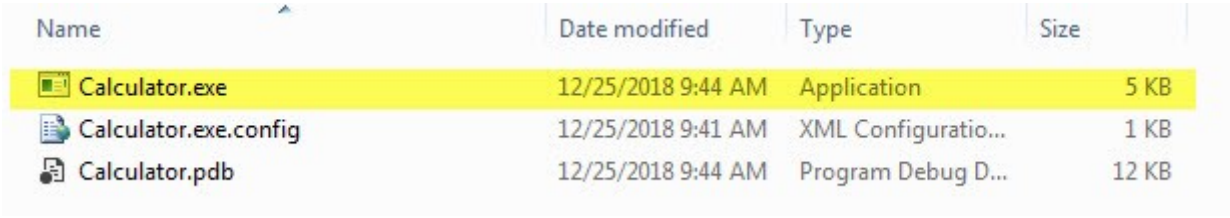
To understand the rule of CIL take a look at following C# code which models a calculator.

```
using System;

namespace Calculator
{

    class Program
    {
        static void Main(string[] args)
        {
            Calc cl=new Calc();
            Console.WriteLine("13+31={0}",cl.AddNumbers(13,31));
            Console.ReadLine();
        }
    }
    //Calculator class which has a method Addnumbers
    class Calc
    {
        public int AddNumbers(int x, int y)
        {
            return x + y;
        }
    }
}
```

As you can see we have two classes,

- Program: which is the app's entry point
- Calc:which has a method named Addnumbers and add two number and return the result of adding numbers.

After compiling the application using C# compiler (Calculator.exe), you end up with *.exe file that contains a manifest, metadata, CIL instructions.



Now let's use ildasm.exe to examine our Calculator.exe file.

You can find ildasm.exe file in C:\Program Files (x86)\Microsoft SDKs\Windows\[your windows version]\Bin

If you want to use this tool in Visual Studio just follow these steps and add it in Visual Studio tools,

- First find ildasm.exe location (usually it's located in C:\Program Files (x86)\Microsoft SDKs\Windows\[Windows Verwsion]\Bin\ildasm.exe
- Open up Visual Studio
- From Tools menu open External Tools and then you should see External Tools dialog,

ILDASM.exe

- Click on add button, choose a Title of your tool (my title is ILDASM) ,in Command text box enter the path of ildasm.exe and in Argument textbox just click on right-arrow button and choose $(TargetPath)
- Check Close and exit.
- That's it, it's done.

What ildasm.exe offers?

ildasm will show you three basic things,

- A list of all classes and methods in your assembly
- The contents of the assembly's manifest
- IL code for any method implemented in the assembly.

To open related assembly just go to tools menu and here we go,ILDASM is appear in this menu,just open it.

So,if you want to find some information about your application,open ildasm in your Tools menu,and you'll see ildasm.exe dialog.

As you see there are some information about assembly's type, manifest and metadata.

Assembly

If I expand Calculator.Calc there would be AddNumbers methods.by double clicking on the method I can see the information about this method in new dialog windows.

Pay attention to int32 data type, as you remember in early of this article I talked about CTS (Common Type System) in which all of data types after compilation would be understandable for CTS. Whatever int in C# or Integer in VB will be compiled as int32 in CTS.

So as you can see the AddNumbers method is represented using CIL as above picture.

Benefit of CIL

Now you've found out that what CIL exactly does, but why is it necessary to compile your code into CIL?do you remember as I said you can use variety of languages in .NET framework, here's a benefit of using CIL,no matter which language you use in your project maybe one or maybe two and so forth, at the end all of them will compile into CIL and all of languages will interact within a well-defined binary arena.

The Role of .NET Type Metadata

.NET assembly contains complete and accurate metadata, such as class, structure and also members of each type such as properties, methods and etc.so, how these information extract? By programmer or compiler? Hopefully all of these task done by compiler not the programmer.

To show you the format of the metadata, let's take a look at our project metadata generated by AddNumbers method.to do that,simply open ildasm in your Tools menu and after dialog popped up ,just hit ctl+m to illustrate the type of Metadata of your project.

If you hit ctl+m and then dialog box disappeares make sure to open proper version of ildasm.If your project is generated by .Net framework version 4.6.1 and you open ildasm with .NET framework version 4.5.1 your dialog box will disappear after using ctl+m.

as you see in above picture , we have a TypeDef #2 which provides some information about the project , for example TypeDefName refers to Calc class in Calculator project and also there are two methods in Calc class,

- AddNumbers
- which takes two parameters x and y

- .ctor which is default constructor of Calc class and generated automatically by .NET framework.

The Role of the Assembly Manifest

.Net assembly which contains metadata that describes the assembly itself,as I mentioned earlier it's called Manifest.

Manifest documents all external assemblies information such as the assembly's version number, copyright information and so on.so accordingly it is compiler duty to generate assembly's manifest rather than the programmer.

Manifest

resource: pro C# 7 by Andrew Troelsen,Philip Japikse

C Programming C Language Mvc Frameworks Programming Java