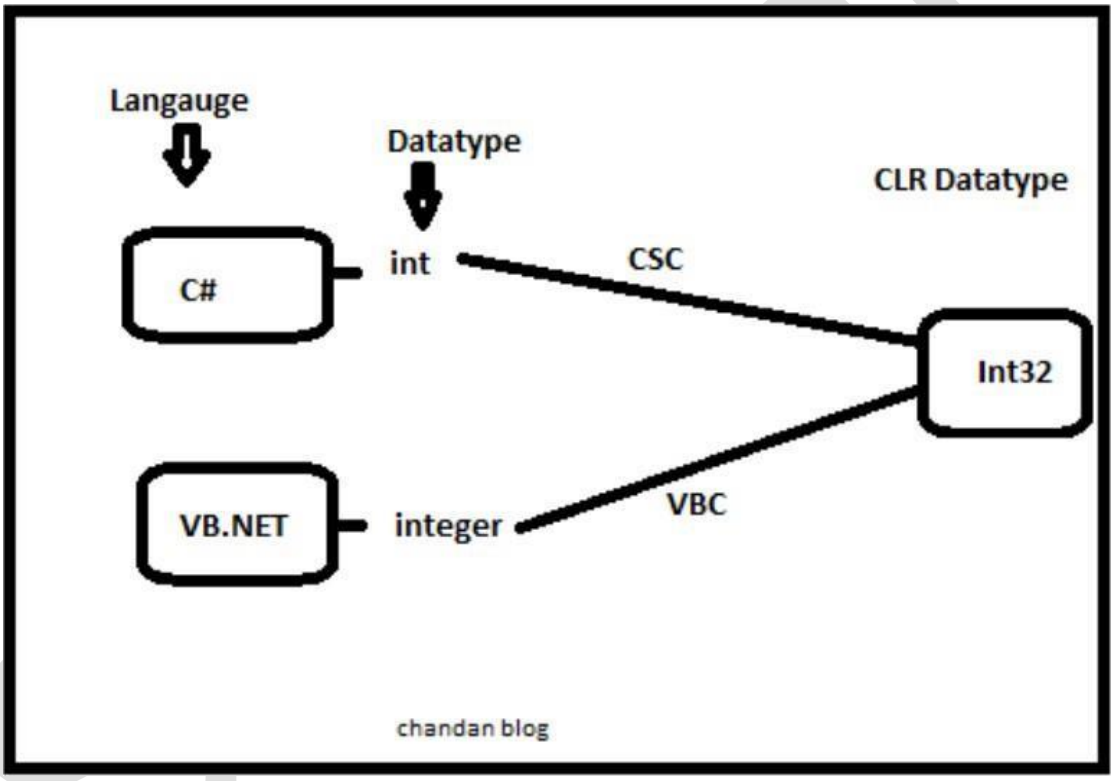
Framework Basics

CTS:-

Common Type System (CTS) describes the datatypes that can be used by managed code. CTS defines how these types are declared, used and managed in the runtime. It facilitates cross-language integration, type safety, and high-performance code execution. The rules defined in CTS can be used to define your own classes and values.

OR we can also understand like,

CTS deals with the data type. So here we have several languages and each and every language has its own data type and one language data type cannot be understandable by other languages but .NET Framework language can understand all the data types. C# has an **int** data type and VB.NET has **Integer** data type. Hence a variable declared as an int in C# and Integer in VB.NET, finally after compilation, uses the same structure Int32 from CTS.



Note

All the structures and classes available in CTS are common for all .NET Languages and the purpose of these is to support language independence in .NET. Hence it is called CTS.

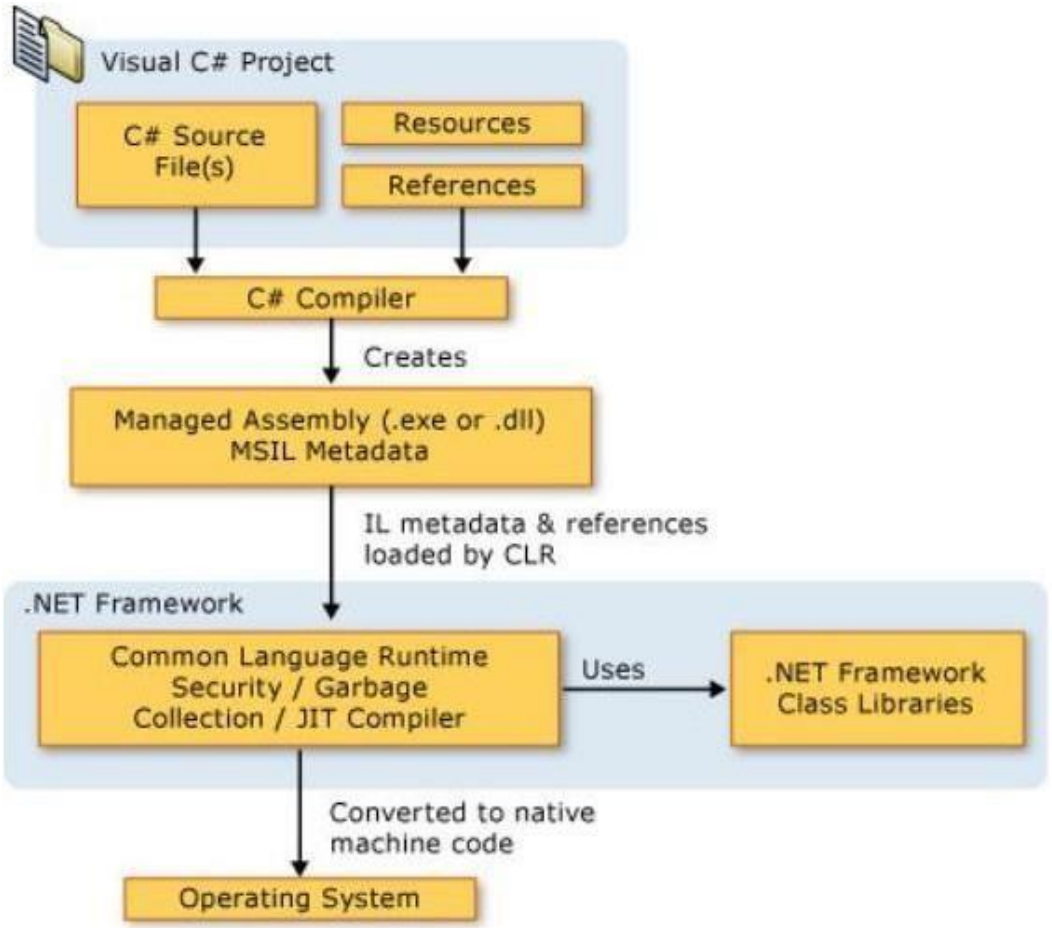
Common Language Runtime (CLR) :-

The Common Language Runtime (CLR) is a core component of .NET Framework that manages the execution and the lifecycle of all .NET applications (code). It provides various services, including automatic memory management, exception handling, security, and type safety. When a .NET application is compiled, it generates an intermediate language code called Common Intermediate Language (CIL). The CLR is responsible for translating this CIL into machine code and managing the execution of the resulting program. The CLR also provides a platform for interoperability between different programming languages that target the .NET Framework. This means that a program written in one .NET language can easily use libraries written in another .NET language. Overall, the CLR is an essential component of the .NET Framework that enables developers to create robust, secure, and interoperable applications.

Key attributes of .NET CLR :-

- As part of the Microsoft .NET Framework, the Common Language Runtime (CLR) is the programming (Virtual Machine component) that manages the execution of programs written in any language that uses the .NET Framework, for example, C#, VB.Net, F# and so on.
- Programmers write code in any language, including VB.Net, C#, and F#, when they compile their programs into an intermediate form of code called CLI in a portable execution file (PE) that can be managed and used by the CLR. Then the CLR converts it into machine code to be will be executed by the processor.
- The information about the environment, programming language, its version, and what class libraries will be used for this code are stored as metadata with the compiler that tells the CLR how to handle this code.
- The CLR allows an instance of a class written in one language to call a method of the class written in another language.

Sunbeam



As you can see from the above diagram, the CLR provides several services.

Functions of .NET CLR:-

- Convert code into CLI
- Exception handling
- Type safety
- Memory management (using the Garbage Collector)
- Security
- Improved performance
- Language independency
- Platform independency
- Architecture independency

Components of .NET CLR:-

- Performance improvements.
- The ability to easily use components developed in other languages.
- A class library provides extensible types.
- Language features such as inheritance, interfaces, and overloading for object-oriented programming.
- Support for explicit free threading that allows the creation of multithreaded, scalable applications.
- Support for structured exception handling.
- Support for custom attributes.
- Garbage collection.
- Use of delegates instead of function pointers for increased type safety and security.

IL Code:-

IL code stands for intermediate code.

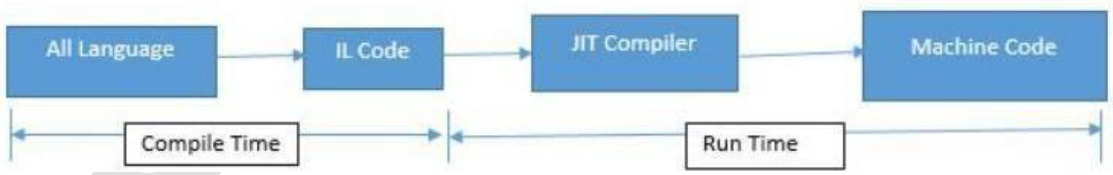
In Microsoft visual studio supports too many language. In same different user have different machine configuration and different operating system whose unknown to visual studio. That is the main problem. To avoid this problem Microsoft Creates a code that is called as IL Code.

It's called as IL code or half compiled code, it's creates at compile time.

Now why it is called as half compiled?

Visual studio does not know that what is system configuration of user, so in compile time it's compiled the language code and converted into IL code.

When this program will run at user system, this run time IL code is converted into machine code by JIT (just-in-time) compiler.



Summery

- IL code is half compiled code.
- IL Code is created at compile time.
- IL code is compiled by JIT Compiler.
- JIT Compiler converted IL Code into Machine Code.

What is a .Net Assembly?

The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file. All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. The designers of .NET have worked a lot on the component (assembly) resolution.

An assembly can be a single file or it may consist of the multiple files. In the case of multi-file, there is one master module containing the manifest while other assemblies exist as non-manifest modules. A module in .NET is a subpart of a multi-file .NET assembly. Assembly is one of the most interesting and extremely useful areas of .NET architecture along with reflections and attributes.

.NET supports three kinds of assemblies:

1. private
2. shared
3. satellite

Private Assembly

Private assembly requires us to copy separately in all application folders where we want to use that assembly's functionalities; without copying, we cannot access the private assembly features and power. Private assembly means every time we have one, we exclusively copy into the BIN folder of each application folder.

Public Assembly

Public assembly is not required to copy separately into all application folders. Public assembly is also called Shared Assembly. Only one copy is required in system level, there is no need to copy the assembly into the application folder.

Public assembly should install in GAC.

Shared assemblies (also called strong named assemblies) are copied to a single location (usually the Global assembly cache). For all calling assemblies within the same application, the same copy of the shared assembly is used from its original location. Hence, shared assemblies

Sunbeam

are not copied in the private folders of each calling assembly. Each shared assembly has a four-part name including its face name, version, public key token, and culture information. The public key token and version information makes it almost impossible for two different assemblies with the same name or for two similar assemblies with a different version to mix with each other.

GAC (Global Assembly Cache)

When the assembly is required for more than one project or application, we need to make the assembly with a strong name and keep it in GAC or in the Assembly folder by installing the assembly with the GACUtil command.

Satellite Assembly

Satellite assemblies are used for deploying language and culture-specific resources for an application.

What are the basic components of the .NET platform?

The basic components of .NET platform (framework) are:

.Net Applications *(Win Forms,Web Applications,Web Services)*
Data(ADO.Net) and XML Library
FrameWork Class Library(FCL) *(IO,Streams,Sockets,Security,Reflection,UI)*
Common Language Runtime(CLR) *(Debugger,Type Checker,JITer,GC)*
Operating System *(Windows,Linux,UNIX,Macintosh,etc.,)*

Common Language Runtime (CLR)

The most important part of the .NET Framework is the .Net Common Language Runtime (CLR) also called .Net Runtime in short. It is a framework layer that resides above the Operating System and handles/manages the execution of the .NET applications. Our .Net programs don't directly communicate with the Operating System but through CLR.

MSIL (Microsoft Intermediate Language) Code

When we compile our .Net Program using any .Net compliant language like (C#, VB.NET, C++.NET) it does not get converted into the executable binary code but to an intermediate code, called MSIL or IL in short, 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. The presence of IL makes it possible for the Cross-Language Relationship as all the .Net compliant languages produce the similar standard IL code.

Just In Time Compilers

When our IL compiled code needs to be executed, CLR invokes JIT compilers which compile the IL code to native executable code (.exe or .dll) for the specific machine and OS. JITers in many ways is different from traditional compilers as they, as their name suggests, compile the IL to native code only when desired e.g., when a function is called, IL of function's body is converted to native code; just in time of need. So, the part of code that is not used by a particular run is not converted to native code. If some IL code is converted to native code then the next time when it's needed to be used, the CLR uses the same copy without re-compiling. So, if a program runs for some time, then it won't have any just in a time performance penalty. As JITers are aware of processor and OS exactly at runtime, they can optimize the code extremely efficiently resulting in very robust applications. Also, since JITer knows the exact current state of executable code, they can also optimize the code by in-lining small function calls (like replacing body of small function when its called in a loop, saving the function call time). Although Microsoft stated that C# and .Net are not competing with languages like C++ in efficiency, speed of execution, JITers can make your code even faster than C++ code in some cases when the program is run over an extended period of time (like web-servers).

Framework Class Library (FCL)

.NET Framework provides a huge set of Framework (or Base) Class Library (FCL) for common, usual tasks. FCL contains thousands of classes to provide them access to Windows API and common functions like String Manipulation, Common Data Structures, IO, Streams, Threads, Security, Network Programming, Windows Programming, Web Programming, Data Access, etc. It is simply the largest standard library ever shipped with any development environment or programming language. The best part of this library is they follow extremely efficient OO design (design patterns) making their access and use very simple and predictable. You can use the classes in FCL in your program just as you use any other class and can even apply inheritance and polymorphism on these.

Common Language Specification (CLS)

Earlier we used the term '.NET Compliant Language' and stated that all the .NET compliant languages can make use of CLR and FCL. But what makes a language '.NET compliant language'? The answer is the Common Language Specification (CLS). Microsoft has released a small set of specifications that each language should meet to qualify as a .NET Compliant Language. As IL is a very rich language, it is not necessary for a language to implement all the IL functionality, rather it meets the small subset of it, CLS, to qualify as a .NET compliant language, which is the reason why so many languages (procedural and OO) are now running under .Net umbrella. CLS basically addresses to language design issues and lays certain standards like there should be no global function declaration, no pointers, no multiple

Sunbeam

inheritance and things like that. The important point to note here is that if you keep your code within the CLS boundary, your code is guaranteed to be usable in any other .Net language.

Common Type System (CTS)

.NET also defines a Common Type System (CTS). Like CLS, CTS is also a set of standards. CTS defines the basic data types that IL understands. Each .NET compliant language should map its data types to these standard data types. This makes it possible for the 2 languages to communicate with each other by passing/receiving parameters to/from each other. For example, CTS defines a type Int32, an integral data type of 32 bits (4 bytes) which is mapped by C# through int and VB.Net through its Integer data type.

Garbage Collector (GC)

CLR also contains Garbage Collector (GC) which runs in a low-priority thread and checks for un-referenced dynamically allocated memory space. If it finds some data that is no more referenced by any variable/reference, it re-claims it and returns the occupied memory back to the Operating System; so that it can be used by other programs as necessary. The presence of standard Garbage Collector frees the programmer from keeping track of dangling data.

Sunbeam