**.NET Framework**

**.NET Framework:**  The .NET Framework is a platform for building, deploying, and running Services and applications. It is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library called Framework Class Library (FCL).

With .NET, we can use multiple languages, editors, and libraries to build for web, mobile, desktop, games, AI and IoT. The Framework provides an execution environment, simplified development , deployment, and integration with a variety of programming languages, including Visual Basic, F# and C#.

The two major components of .NET Framework are the **Common Language Runtime (CLR)** and the .NET **Framework Class Library (FCL)**. The CLR is the execution engine that handles running applications. The FCL provides a set of APIs and types for common functionality.It is built on top of the CLR and provides services needed by modern applications.

**Key components of .NET Framework**

Common Language Runtime (CLR)

.NET Framework Class Library (FCL)

Common Type System

Common Language Specification

Application Domains

Runtime Host

Metadata and Self-Describing Components

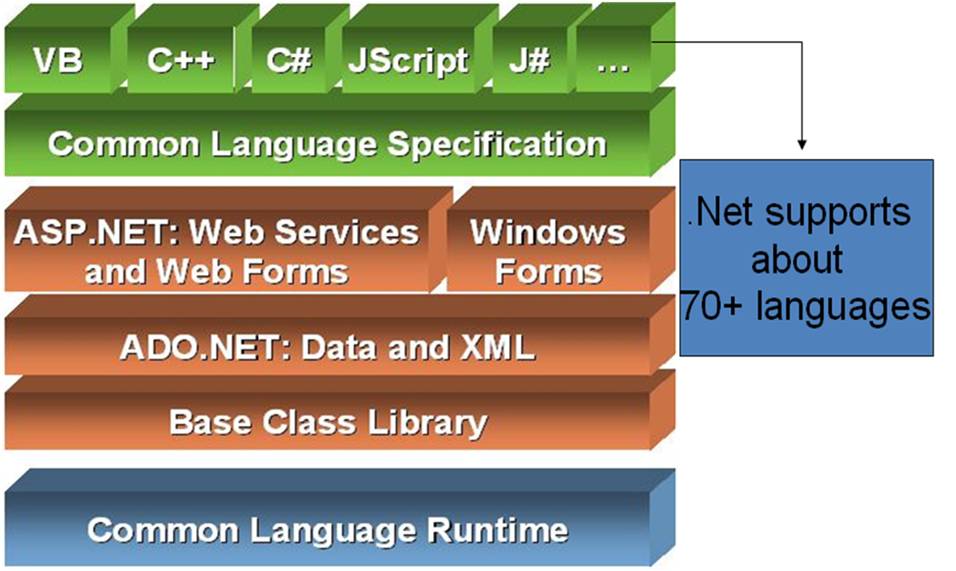
Cross-Language Interoperability

.NET Framework Security

Profiling

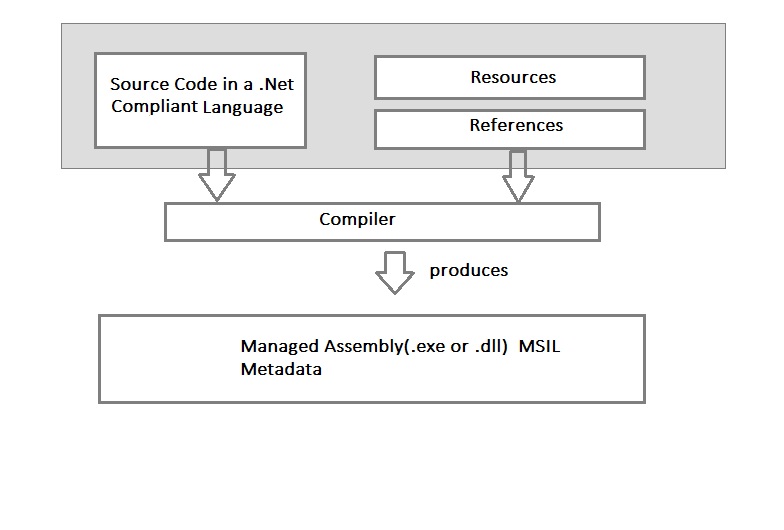
Side-by-Side Execution

**.Net Framework Component Stack**



**Compilation process**

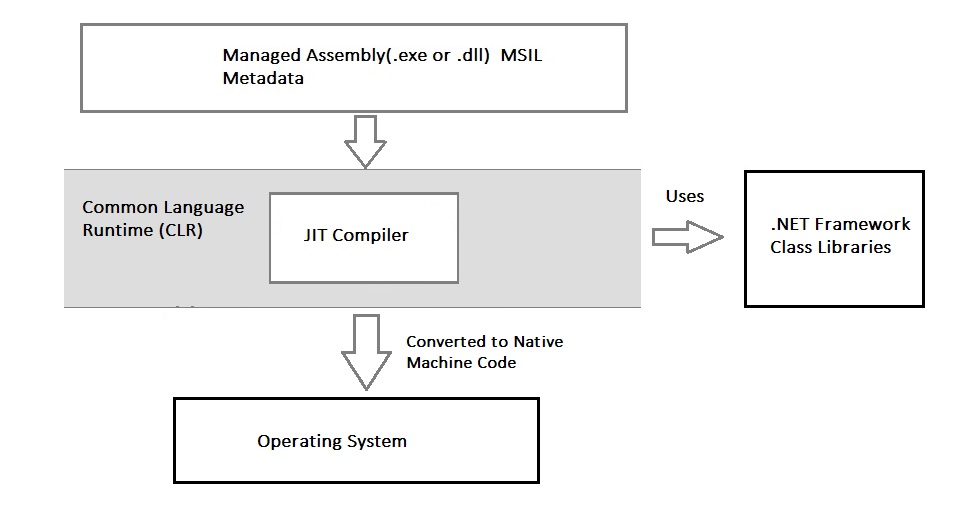
The compiler for a .NET language takes a source code file and produces an output file called an assembly. An assembly is either an executable or a DLL. The code in an assembly is not native machine code, but an intermediate language called the Common Intermediate Language (CIL) or IL.



**Execution Process**

The program’s CIL is not compiled to native machine code until it is called to run. At run time, the CLR performs the following steps:

* It checks the assembly’s security characteristics.
* It allocates space in memory.
* It sends the assembly’s executable code to the Just-in-Time (JIT) compiler, which compiles portions of it to native code.



While applications targeting the .NET Framework interact directly with the FCL, the CLR serves as the underlying engine. In order to understand the .NET Framework, one first must understand the role of the CLR.

**Common Language Runtime (CLR)**

The CLR is a modern runtime environment that manages the execution of user code, providing services such as JIT compilation, memory management, exception management, debugging and profiling support, and integrated security and permission management.

CLR provides an environment to execute .NET applications on target machines. CLR is also a common runtime environment for all .NET code irrespective of their programming language, as the compilers of respective language in .NET Framework convert every source code into a common language known as **MSIL or IL (Intermediate Language)**. CLR also provides various services to execute processes, such as memory management service and security services. CLR performs various tasks to manage the execution process of .NET applications.

**The responsibilities of CLR are listed as follows:**

* Automatic memory management
* Garbage Collection
* Code Access Security
* Code verification
* JIT compilation of .NET code
* Language independency
* Platform independency
* Architecture independency

## **Key attributes of .NET CLR**

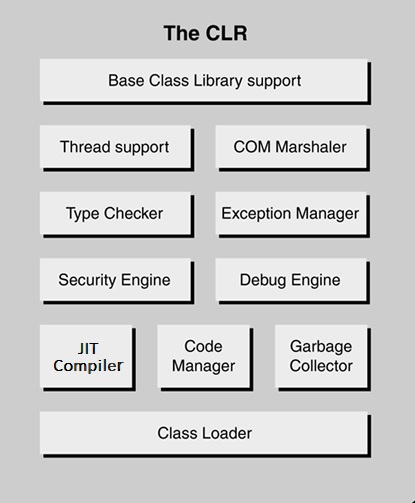
* As part of the Microsoft .NET Framework, the Common Language Runtime (CLR) is the programming (Virtual Machine component) that manages the execution of programs written in any language that uses the .NET Framework, for example C#, VB.Net, F# and so on.
* Programmers write code in any language, including VB.Net, C# and F# then 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 and then the CLR converts it into machine code to 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 in the form of 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.

## **Benefits of .NET CLR**

The runtime provides the following benefits:

* Performance improvements.
* The ability to easily use components developed in other languages.
* Extensible types provided by a class library.
* Language features such as inheritance, interfaces, and overloading for object-oriented programming.
* Support for explicit free threading that allows 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.

**Architecture of Common Language Runtime**

****

**Class Loader:**

Various classes, modules, resources, assemblies etc are loaded by the Class Loader. It loads the modules on demand if they are actually required so that the program initialization time is faster and the resources consumed are lesser.

**Garbage Collector:**

Automatic memory management is made possible using the garbage collector in CLR. The garbage collector automatically releases objects memory after it is no longer required so that it can be reallocated.

**Code Manager:**

The code manager in CLR manages the code at run time developed in the .NET framework i.e. the managed code.

**JIT Compiler:**

The JIT compiler is an important element of CLR, which loads MSIL on target machines for execution. The MSIL is stored in .NET assemblies after the developer has compiled the code written in any .NET-compliant programming language, such as Visual Basic and C#. JIT compiler translates the MSIL code of an assembly and uses the CPU architecture of the target machine to execute a .NET application. It also stores the resulting native code so that it is accessible for subsequent calls. If a code executing on a target machine calls a non-native method, the JIT compiler converts the MSIL of that method into native code. JIT compiler also enforces type-safety in runtime environment of .NET Framework. It checks for the values that are passed to parameters of any method.

**Debug Engine:**

An application can be debugged during the run-time using the debug engine. There are various ICorDebug interfaces that are used to track the managed code of the application that is being debugged.

**Security Engine:**

The security engine in the CLR handles the security permissions at various levels such as the code level, folder level, and machine level. This is done using the various tools that are provided in the .NET framework.

**Exception Manager:**

The exception manager in the CLR handles the exceptions regardless of the .NET Language that created them. For a particular application, the catch block of the exceptions are executed in case they occur and if there is no catch block then the application is terminated.

**Type Checker:**

Type safety is provided by the type checker by using the Common Type System (CTS) and the Common Language Specification (CLS) that are provided in the CLR to verify the types that are used in an application.

**COM Marshaller:**

Communication with the COM (Component Object Model) component in the .NET application is provided using the COM marshaller. This provides the COM interoperability support.

**Thread Support:**

The CLR provides thread support for managing the parallel execution of multiple threads. The System.Threading class is used as the base class for this.

**Base Class Library Support:**

The Common Language Runtime provides support for the base class library. The BCL contains multiple libraries that provide various features such as Collections, I/O, XML, DataType definitions, etc. for the multiple .NET programming languages.

**Managed and Unmanged Code**

Managed code is the code that is executed directly by the CLR instead of the operating system. The code compiler first compiles the managed code to intermediate language (IL) code, also called as MSIL code. This code doesn't depend on machine configurations and can be executed on different machines.

Unmanaged code is the code that is executed directly by the operating system outside the CLR environment. It is directly compiled to native machine code which depends on the machine configuration. In the managed code, since the execution of the code is governed by CLR, the runtime provides different services, such as garbage collection, type checking, exception handling, and security support. These services help provide uniformity in platform and language-independent behavior of managed code applications. In the unmanaged code, the allocation of memory, type safety, and security is required to be taken care of by the developer. If the unmanaged code is not properly handled, it may result in memory leak.

**Framework Class Library (FCL)**

The Framework Class Library (FCL) is a component of Microsoft's .NET Framework, it is a collection of reusable classes, interfaces and value types, and includes an implementation of the Base Class Library (BCL). The FCL provides a diverse array of higher-level software services, addressing the needs of modern applications. Conceptually, these can be grouped into several categories such as:

Support for core functionality, such as interacting with basic data types and collections;

console, **network** and file I/O; and interacting with other runtime-related facilities.

Support for interacting with **databases**; consuming and producing XML; and manipulating tabular and tree-structured data.

Support for building **web-based** (thin client) applications with a rich server-side event model.

Support for building **desktop-based** (thick client) applications with broad support for the Windows GUI.

Support for building **SOAP-based** XML web services.

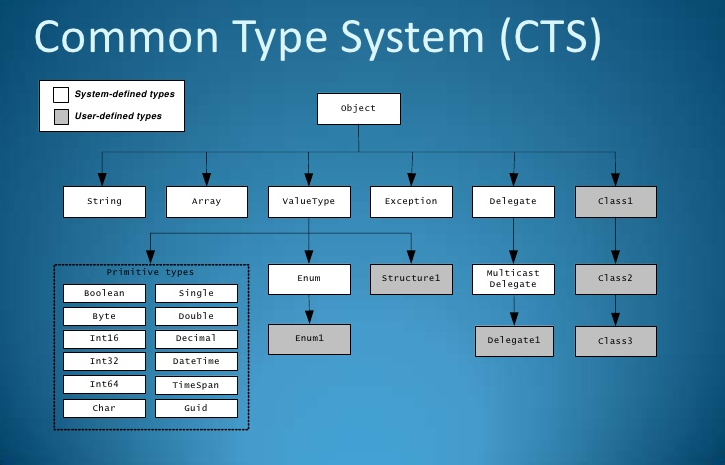
The FCL is vast, including more than 3,500 classes

**Common Type System (CTS)**

Ultimately, the CLR exists to safely execute managed code, regardless of source language. In order to provide for cross-language integration, to ensure type safety, and to provide managed execution services such as JIT compilation, garbage collection, exception management, etc., the CLR needs intimate knowledge of the managed code that it is executing. To meet this requirement, the CLR defines a shared type system called the Common Type System (CTS). The CTS defines the rules by which all types are declared, defined and managed, regardless of source language. The CTS is designed to be rich and flexible enough to support a wide variety of source languages, and is the basis for cross-language integration, type safety, and managed execution services.

Compilers for managed languages that wish to be first-class citizens in the world of the CLR are responsible for mapping source language constructs onto the CTS analogs. In cases in which there is no direct analog, the language designers may decide to either adapt the source language to better match the CTS (ensuring more seamless cross-language integration), or to provide additional plumbing that preserves the original semantics of the source language (possibly at the expense of cross-language integration capabilities).

Since all types are ultimately represented as CTS types, it now becomes possible to combine types authored in different languages in new and interesting ways. For example, since managed languages ultimately declare CTS types, and the CTS supports inheritance, it follows that the CLR supports cross-language inheritance.



**Common Language Specification**

Not all languages support the exact same set of constructs, and this can be a barrier to cross-language integration. Consider this example: Language A allows unsigned types (which are supported by the CTS), while Language B does not. How should code written in Language B call a method written in Language A, which takes an unsigned integer as a parameter?

The solution is the Common Language Specification (CLS). The CLS defines the reasonable subset of the CTS that should be sufficient to support cross-language integration, and specifically excludes problem areas such as unsigned integers, operator overloading, and more.

Each managed language decides how much of the CTS to support. Languages that can consume any CLS-compliant type are known as CLS Consumers. Languages which can extend any existing CLS-compliant type are known as CLS Extenders. Naturally, managed languages are free to support CTS features over and above the CLS, and most do. As an example, the C# language is both a CLS Consumer and a CLS Extender, and supports all of the important CTS features.

The combination of the rich and flexible CTS and the widely supported CLS has led to many languages being adapted to target the .NET platform. Microsoft is offering compilers for various managed languages (C#, VB.NET, JScript, Managed Extensions for C++, Microsoft IL, and J#), and a host of other commercial vendors and academics are offering managed versions of languages, such as COBOL, Eiffel, Haskell and more.

**Side-by-Side Execution in the .NET Framework**

Side-by-side execution is the ability to install multiple versions of component on the same computer so that an application can choose which version of the common language runtime or of a component it uses.

**ADO.NET:** It is a data access technology from the Microsoft .NET Framework that provides communication between relational and non-relational systems through a common set of components. ADO.NET is a set of computer software components that programmers can use to access data and data services from a database.

**WPF (Windows Presentation Foundation) :** Windows Presentation Foundation (WPF) is a graphical subsystem given by Microsoft which uses DirectX and is used in Windows-based applications for rendering UI (User Interface). WPF was initially released as part of .NET Framework 3.0 in 2006 and previously known as **“Avalon”**.

**WCF (Windows Communication Foundation) :** It is a framework for building connected and service-oriented applications used to transmit the data as asynchronous from one service endpoint to another service point. It was previously known as the **Indigo**.

**WF (Windows Workflow Foundation) :** It is a technology given by Microsoft which provides a platform for building workflows within .Net applications.

**Card Space :** It is a Microsoft .NET Framework software client which is designed to let users provide their digital identity to online services in a secure, simple and trusted way.

**LINQ (Language Integrated Query) :** It is introduced in .Net framework version 3.5. Basically, it is a query language used to make the query for data sources with VB or [C#](https://www.geeksforgeeks.org/introduction-to-c/) programming languages.

**Entity Framework :** It is open–source ORM (Object Relational Mapping) based framework which comes into .Net Framework version 3.5. It enables the .Net developer to work with databases using .Net objects. Before the entity framework, .Net developers have performed a lot of things related to databases. Like to open a connection to the database, developers have to create a Data Set to fetch or submit the data to the database, convert data from the Data Set to .NET objects or vice-versa. It creates the difficulties for developers and also it was the error-prone process, then **“Entity Framework”** comes to automate all these database related activities for the application. So, Entity Framework allows the developers to work at a higher level of abstraction.

**Note :** **REST (Representational State Transfer)** and **AJAX** were added in .Net Framework 3.5 as an extension and services of ASP.NET for enhancing web services of .NET Framework.

**Parallel LINQ (Language Integrated Query) :** It comes in .Net Framework version 4.0 and also termed as PLINQ. It provides a concurrent query execution engine for **LINQ**. It executes the **LINQ** in parallel such that it tries to use as much processing power as the system on which it is executing.

**TPL (Task Parallel Library) :** It is a set of public types and APIs. It allows the developers to be more productive by simplifying the process of adding concurrency and parallelism to .Net applications.

**.NET API(Application Programming Interface) For Store/UWP(Universal Windows Platform) Apps :** In 2012, Microsoft added some APIs for creating **UWP(Universal Windows Platform)** apps for Windows using [C#](https://www.geeksforgeeks.org/introduction-to-c/) or VB.

**Task-Based Asynchronous Model :** It is a model used to describe the asynchronous operations and tasks in .Net Framework.

**Types of Applications :** Mainly the applications which are built in .Net framework is divided into the following categories :

* **ASP.Net Web applications** are programs that used to run inside some web server to fulfill the user requests over the http. ASP.NET Web applications can range from simple Web sites that consist of HTML pages to advanced enterprise applications that run on local and remote networks. These enterprise applications also provide components for exchanging data using XML. This type includes dynamic and data driven browser based applications. (Ex: Hotmail and Google).
* **Web services** are “web callable” functionality available via industry standards like HTTP, XML and SOAP.
* **Windows applications** are form based standard Windows desktop applications for common day to day tasks. (Ex: Microsoft word). Run only under Windows environment. These applications consume the services provided by the Windows operating system.
* **Windows services** are long-running executable applications that run on the system as a background process. These applications do not interfere with the working of the other processes that run on the same computer. Windows services execute within separate Windows sessions created specifically for each Windows service. These services do not have a graphic user interface and are ideal for running on the server. Windows services were earlier called NT services.
* **Console applications** are light weight programs run inside the command prompt (DOS) window. They are commonly used for test applications.
* **Mobile applications** can run on multiple mobile devices, such as Pocket PCs, mobile phones, or personal digital assistants. These applications provide ubiquitous access to data from mobile devices. The .NET Framework automatically makes changes to these applications to enable them to run on multiple browsers, depending on the mobile device.
* **Class libraries** are components that you create once and reuse a number of times in multiple applications. Class libraries allow you to define several classes, along with their methods and interfaces, in one file. These libraries compile to .dll files and facilitate rapid development of new applications because of reusability of code. To access the functionality of the classes in a class library from your application, you need to include a reference to that library in your program.