Introduction and Overview of .Net Framework

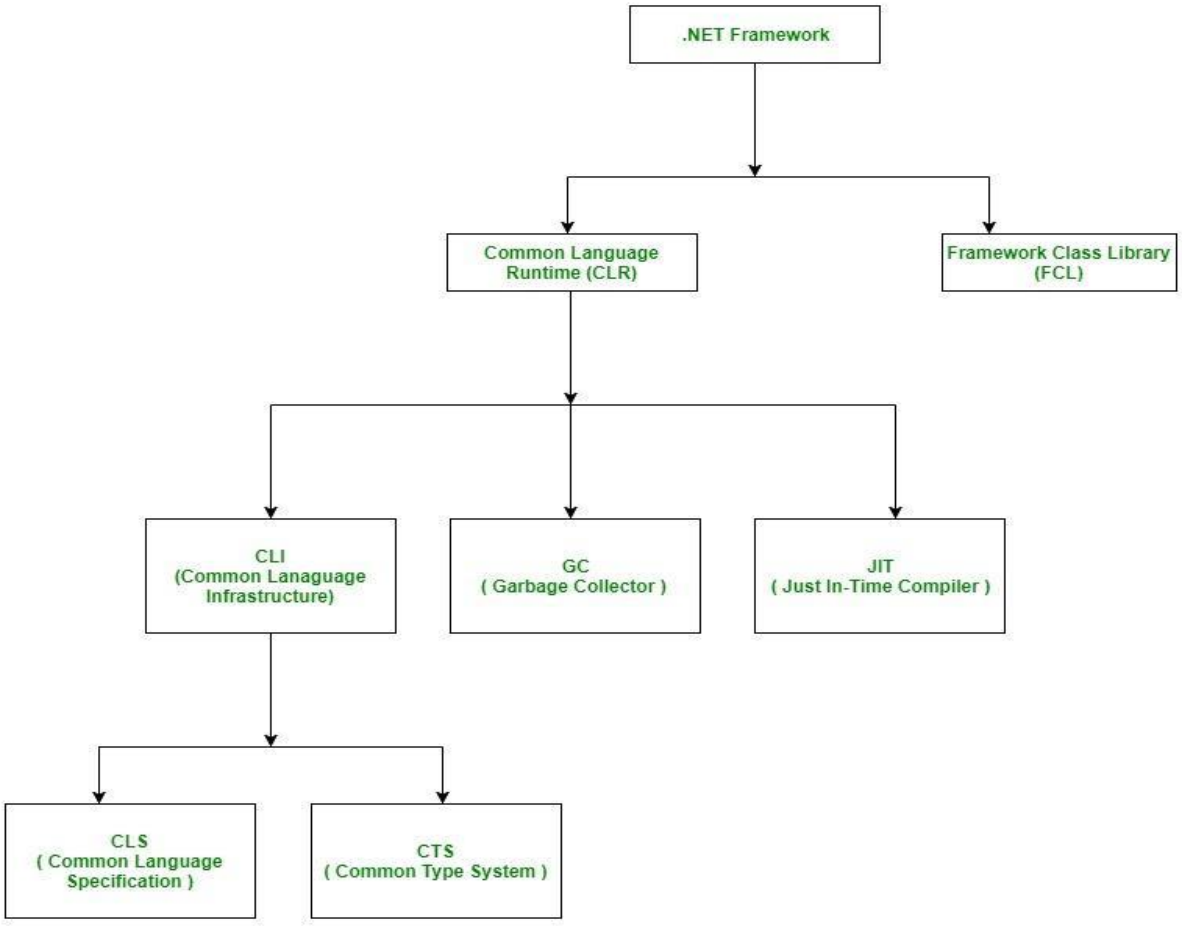
.NET is the framework for which we develop applications. It sits in between our application programs and operating system. .NET provides an object-oriented environment. It ensures safe execution of the code by performing required runtime validations.

For example, it is never possible to access an element of an array outside the boundary. Similarly, it is not possible to a program to write into another programs area, etc. The runtime validations performed by .NET makes the entire environment robust.

It is a programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services.

The .NET Framework consists of:

1. Common Language Runtime (CLR)
2. Framework Class Library (FCL)



Common Language Runtime (CLR)

CLR is the basic and Virtual Machine component of the .NET Framework. It is the run-time environment in the .NET Framework that runs the codes and helps in making the development process easier by providing the various services such as remoting, thread management, type-safety, memory management, robustness etc.

Basically, it is responsible for managing the execution of .NET programs regardless of any .NET programming language. It also helps in the management of code, as code that targets the runtime is known as the Managed Code and code doesn't target to runtime is known as Unmanaged code.

Features of Common Language Runtime:

1. The runtime enforces code access security. For example, users can trust that an executable embedded in a Web page can play an animation on screen or sing a song, but cannot access their personal data, file system, or network.
2. The runtime also enforces code robustness by implementing a strict type-and-code verification infrastructure called the C*ommon Type System (CTS)*. The CTS ensures that all managed code is self-describing.
3. The runtime also accelerates developer productivity. For example, programmers can write applications in their development language of choice, yet take full advantage of the runtime, the class library, and components written in other languages by other developers. Any compiler vendor who chooses to target the runtime can do so. Language compilers that target the .NET Framework make the features of the .NET Framework available to existing code written in that language, greatly easing the migration process for existing applications.
4. While the runtime is designed for the software of the future, it also supports software of today and yesterday.
5. The runtime is designed to enhance performance.
6. The runtime can be hosted by high-performance, server-side applications, such as Microsoft SQL Server and Internet Information Services (IIS).

Framework Class Library (FCL)

The class library is a comprehensive, object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

It is the collection of reusable, object-oriented class libraries and methods etc. that can be integrated with CLR. Also called the Assemblies. It is just like the header files in C/C++ and

packages in the java. Installing .NET framework basically is the installation of CLR and FCL into the system.

The core libraries are sometimes collectively called the Base Class Library (BCL). The entire framework is called the Framework Class Library (FCL).

Features of Framework Class Library

1. An object-oriented class library, the .NET Framework types enable you to accomplish a range of common programming tasks, including tasks such as string management, data collection, database connectivity, and file access.
2. In addition to these common tasks, the class library includes types that support a variety of specialized development scenarios.
   For example, you can use the .NET Framework to develop the following types of applications and services:
   - Console applications
   - Windows GUI applications (Windows Forms).
   - Windows Presentation Foundation (WPF) applications.
   - ASP.NET applications.
   - Windows services.
   - Service-oriented applications using Windows Communication Foundation (WCF).
   - Workflow-enabled applications using Windows Workflow Foundation (WF).

What is New in .NET Framework 4.6

- The Garbage Collector (GC) offers more control over when (not) to collect via new methods on the GC class. There are also more fine-tuning options when calling *GC.collect*.
- There is a brand-new faster 64-bit JIT compiler.
- The *System.Numerics* namespace now includes hardware-accelerated matrix, vector types, *BigInteger* and *Complex*.
- There is a new *System.AppContext* class, designed to give library authors a consistent mechanism for letting consumers switch new API features in or out.
- Tasks now pick up the current thread's culture and UI culture when created.
- More collection types now implement *IReadOnlyCollection<T>*
- WPF has further improvements, including better touch and high-DPI handling
- ASP.NET now supports HTTP/2 and the Token Binding Protocol in Windows 10.

What is New in .NET Framework 4.7

Framework 4.7 is more of a maintenance release than a new-feature release, with numerous bug fixes and minor improvements. Additionally:

- The *System.ValueTuple* struct is a part of Framework 4.7, so you can use tuples in C#7 without referencing the System.ValueTuple.dll assembly.
- WPF has better touch support
- Windows Form has better support for high-DPI monitors

The following table shows the history of compatibility between each version of C#, the CLR and the .NET Framework

C# Version	CLR Version	.NET Framework Versions
1.0	1.0	1.0
1.2	1.1	1.1
2.0	2.0	2.0, 3.0
3.0	2.0 (SP2)	3.5
4.0	4.0	4.0
5.0	4.5 (Patched CLR 4.0)	4.5
6.0	4.6 (Patched CLR 4.0)	4.6
7.0	4.6/4.7 (Patched CLR 4.0)	4.6/4.7

Table: C#, CLR, and .NET Framework versions

Other Frameworks

The Microsoft .NET Framework is the ***most expansive and mature framework***, but runs only on Microsoft Windows (desktop/server). Over the years, other frameworks have emerged to support other platforms. There are currently three major players besides the .NET Framework, all of which are currently owned by Microsoft

1. **Universal Windows Platform (UWP)**

   For writing Windows 10 Store Apps and for targeting Windows 10 devices (mobile, XBox, Surface Hub, Hololens). Your app runs in a sandbox to lessen the threat of malware, prohibiting operations such as reading or writing arbitrary files.

2. **.NET Core with ASP.NET Core**

   An open source framework (originally based on a cut-down version of the .NET Framework) for writing easily deployable Internet apps and micro services that run on

Windows, macOS, and Linux. Unlike the .NET Framework, .NET Core can be packaged with the web application and xcopy deployed (self-contained deployment).

3. **Xamarin**

   For writing mobile apps that target iOS, Android, and Windows Mobile. Microsoft purchased the Xamarin Company in 2016.

The following table compares the current platform support for each of the major frameworks.

Operating System	.NET Framework	UWP	.NET Core	Xamarin
Windows 7/8	Yes		Yes	
Windows 10 Desktop/Server	Yes	Yes	Yes	
Windows 10 Devices		Yes		Yes
Linux			Yes	
macOS			Yes	
iOS (iPhone)				Yes
Android				Yes

Table: Platform Support for the popular frameworks

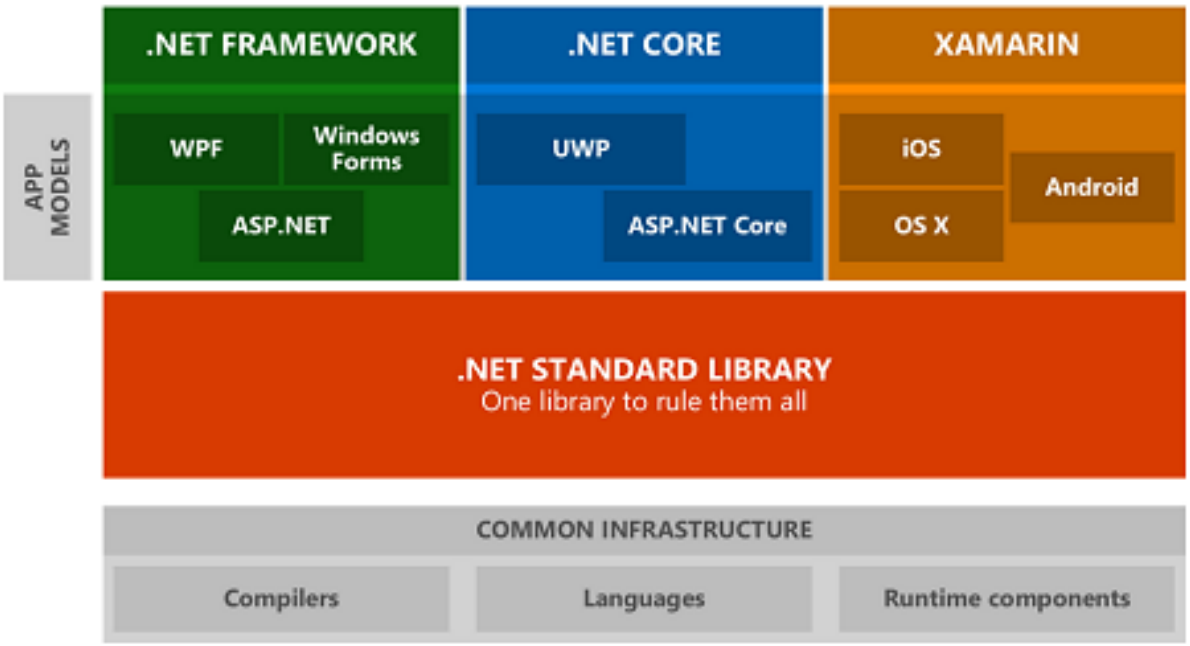
.NET Standard 2.0

.Net Standard is a specification, which dictates what the Base Class Libraries of different .Net platforms should implement to unify the Base Class Libraries of different .Net Platforms. Here, Platform means full .Net Framework, .Net Core, Xamarin, Silverlight, XBox etc.
This also enables code sharing between applications that runs on these different platforms. For example, a library or a component that is developed on top of a platform that implements specific .Net Standard version can be shared by all the applications that runs on any of the platforms that implements same .Net Standard version.

.NET Standard is not a Framework; it is merely a specification describing a minimum baseline of functionality (types and members), which guarantees compatibility with a certain set of frameworks. The concept is similar to C# interfaces: .NET Standard is like an interface that concrete types (frameworks) can implement

.Net Standard has solved all this in a different way; it provides an API specification, which all platforms should implement to remain .Net Standard complaint. This has unified the base

class libraries of different .Net platforms and has paved way to share libraries and also brought the BCL evolution centralized as seen below:



.NET Standard 2.0 is a new version that significantly increases the number of APIs compared to the previous version (1.6.1). In fact, the API surface has more than doubled with .NET Standard 2.0.

The .NET Standard 2.0 is supported by the following .NET implementations:

- .NET Core 2.0 or later
- .NET Framework 4.6.1 or later
- Mono 5.4 or later
- Xamarin. iOS 10.14 or later
- Xamarin. Mac 3.8 or later
- Xamarin. Android 8.0 or later
- Universal Windows Platform 10.0.16299 or later

What is new in the .NET Standard 2.0?

The .NET Standard 2.0 includes the following new features:

1. A vastly expanded set of APIs.
2. Support for .NET Framework libraries.
3. Support for Visual Basic.
4. Tooling support for .NET Standard libraries

Older .NET Standards

There are also older .NET Standards in use, most notably 1.1, 1.2, 1.3, and 1.6. A higher-numbered standard is always a strict superset of a lower-numbered standard. For instance, if you write a library that targets .NET Standard 1.6, you will support not only recent versions of the four major frameworks, but also .NET Core 1.0. In addition, if you target .NET Standard 1.3, you support everything we have already mentioned plus.NET Framework 4.6.0 (see table below)

If you target...	You also support...
Standard 1.6	.NET Core 1.0
Standard 1.3	Above plus .NET 4.6.0
Standard 1.2	Above plus .NET 4.5.1, Windows Phone 8.1, WinRT for Windows 8.1
Standard 1.1	Above plus .NET 4.5.0, Windows Phone 8.0, WinRT for Windows 8.0

Table: Older .NET Standards

The 1.x standards lack thousands of APIs that are present in 2.0. This can make targeting a 1.x standard significantly more challenging, especially if you need to integrate existing code or libraries.

If you need to support older frameworks but do not need cross platform compatibility, a better option is to target an older version of a specific framework. In the case of Windows, a good choice is .NET Framework 4.5 because it is widely deployed (pre-installed on all machines running Windows 8 and later), and it contains most of what is in .NET Frame- work 4.7.
You can also think of .NET Standard as a lowest common denominator. In the case of .NET Standard 2.0, the four frameworks that implement it have a similar Base Class Library, so the lowest common denominator is big and useful. However, if you also want compatibility with .NET Core 1.0 (with its significantly cut-down BCL), the lowest common denominator—.NET Standard 1.x—becomes much smaller and less useful.

The CLR and Core Framework System

System Types

The most fundamental types live directly in the System namespace. These include C#'s built-in types, the Exception base class, the Enum, Array, and Delegate base classes, and Nullable, Type, DateTime, TimeSpan, and Guid. The System namespace also includes types for performing mathematical functions (Math), generating random numbers (Random), and converting between various types (Convert and Bit Converter).

Text Processing

The *System.Text* namespace contains the StringBuilder class (the editable or mutable cousin of string), and the types for working with text encodings, such as UTF-8 (Encoding and its subtypes).

The *System.Text.RegularExpressions* namespace contains types that perform advanced pattern-based search-and-replace operations.

Collections

The .NET Framework offers a variety of classes for managing collections of items. These include both list- and dictionary-based structures, and work in conjunction with a set of standard interfaces that unify their common characteristics. All collection types are defined in the following namespaces:

```
System.Collections                  // Nongeneric collections
System.Collections.Generic          // Generic collections
System.Collections.Specialized      // strongly typed collections
System.Collections.ObjectModel      // Bases for your own collections
System.Collections.Concurrent       // Thread-safe collection
```

Queries

Language Integrated Query (LINQ) was added in Framework 3.5. LINQ allows you to perform type-safe queries over local and remote collections (e.g., SQL Server tables). A big advantage of LINQ is that it presents a consistent querying API across a variety of domains. The essential types reside in the following namespaces, and are part of .NET Standard 2.0:

```
System.Linq                 // LINQ to Objects and PLINQ
System.Linq.Expressions     // For building expressions manually
System.Xml.Linq             // LINQ to XML
```

The full .NET Framework also includes the following,

System.Data.Linq *// LINQ to SQL*
System.Data.Entity *// LINQ to Entities (Entity Framework)*

XML

XML is used widely within the .NET Framework, and so is supported extensively. The XML namespaces are:

System.Xml *// XmlReader, XmlWriter + the old W3C DOM*
System.Xml.Linq *// The LINQ to XML DOM*
System.Xml.Schema *// Support for XSD*
System.Xml.Serialization *// Declarative XML serialization for .NET types*
System.Xml.XPath *// XPath query language*
System.Xml.Xsl *// Stylesheet support*

Diagnostics

Diagnostics refers to .NET's logging and assertion facilities and describe how to interact with other processes, write to the Windows event log, and use performance counters for monitoring. The types for this are defined in and under *System.Diagnostics*. Windows-specific features are not part of .NET Standard, and are avail- able only in the .NET Framework.

Concurrency and Asynchronous

Many modern applications need to deal with more than one thing happening at a time. Since C# 5.0, this has become easier through asynchronous functions and high-level constructs such as tasks and task combinators. Types for working with threads and asynchronous operations are in the *System.Threading* and *System.Threading.Tasks* namespaces.

Streams and I/O

The Framework provides a stream-based model for low-level input/output. Streams are typically used to read and write directly to files and network connections, and can be chained or wrapped in decorator streams to add compression or encryption functionality. The .NET Stream and I/O types are defined in and under the *System.IO* namespace, and the WinRT types for file I/O are in and under *Windows.Storage*.

Networking

You can directly access standard network protocols such as HTTP, FTP, TCP/IP, and SMTP via the types in *System.Net*.

System.Net

System.Net.Http *// HttpClient*
System.Net.Mail *// For sending mail via SMTP*
System.Net.Sockets *// TCP, UDP and IP*

The latter two namespaces are unavailable to Windows Store applications if you're targeting Windows 8/8.1 (WinRT), but are available to Windows 10 Store apps (UWP) as part of the .NET Standard 2.0 contract. For WinRT apps, use third-party librariesfor sending mail, and the WinRT types in Windows.Networking.Sockets for working with sockets.

Serialization

The Framework provides several systems for saving and restoring objects to a binary or text representation. Such systems are required for distributed application technologies, such as WCF, Web Services, and Remoting and also to save and restore objects to a file. The types for serialization reside in the following namespaces:
System.Runtime.Serialization
System.Xml.Serialization

Assemblies, Reflection, and Attributes

The assemblies into which C# programs compile comprise executable instructions (stored as intermediate language or IL) and metadata, which describes the pro- gram's types, members, and attributes. Through reflection, you can inspect this metadata at runtime, and do such things as dynamically invoke methods. With *Reflection.Emit*, you can construct new code on the fly.
System
System.Reflection
System.Reflection.Emit *// .NET Framework only*

Dynamic Programming

Dynamic Language Runtime, which has been a part of the CLR since Framework 4.0. The types for dynamic programming are in *System.Dynamic*.

Security

Code access, role, and identity security, and the transparency model introduced in CLR 4.0. Cryptography can be done in the Framework, covering encryption, hashing, and data protection. The types for this are defined in:
System.Security
System.Security.Permissions

System.Security.Policy
System.Security.Cryptography

Advanced Threading

C#'s asynchronous functions make concurrent programming significantly easier because they lessen the need for lower-level techniques. However, there are still times when you need signaling constructs, thread-local storage, reader/writer locks, and so on. Threading types are in the *System.Threading* namespace.