The .NET Framework is a technology that supports building and running the next generation of applications and XML Web services. The .NET Framework is designed to fulfill the following objectives:

* To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
* To provide a code-execution environment that minimizes software deployment and versioning conflicts.
* To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
* To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
* To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
* To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

Manage Code - Code which is executing in CLR

Managed Execution Environment – Choosing a compiler, Convert HLL to MSIL,IL to Machine code using JIT, Running Code

Dotnet Framework = FCL+CLR

The **Common Language Infrastructure** (**CLI**) is an open [specification](https://en.wikipedia.org/wiki/Specification) (technical standard) developed by [Microsoft](https://en.wikipedia.org/wiki/Microsoft) and standardized by [ISO](https://en.wikipedia.org/wiki/International_Organization_for_Standardization)[[1]](https://en.wikipedia.org/wiki/Common_Language_Infrastructure#cite_note-iso_iec_23271_2012-1) and [ECMA](https://en.wikipedia.org/wiki/Ecma_International)[[2]](https://en.wikipedia.org/wiki/Common_Language_Infrastructure#cite_note-ecma_355_2012-2) that describes executable code and a runtime environment that allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures.

The [.NET Framework](https://en.wikipedia.org/wiki/.NET_Framework) and the [free and open source](https://en.wikipedia.org/wiki/Free_and_open_source_software) [Mono](https://en.wikipedia.org/wiki/Mono_(software)) and [Portable.NET](https://en.wikipedia.org/wiki/Portable.NET) are implementations of the CLI.

CLI Consist of :

1. **The**[**Common Type System**](https://en.wikipedia.org/wiki/Common_Type_System)**(CTS)**

A set of [data types](https://en.wikipedia.org/wiki/Data_type) and operations that are shared by all CTS-compliant [programming languages](https://en.wikipedia.org/wiki/Programming_language).

1. **The**[**Metadata**](https://en.wikipedia.org/wiki/Metadata_(CLI))

Information about program structure is [language-agnostic](https://en.wikipedia.org/wiki/Language-independent_specification), so that it can be referenced between languages and tools, making it easy to work with code written in a language one's not using.

1. **The Common Language Specification (CLS)**

A set of base rules to which any language targeting the CLI should conform in order to interoperate with other CLS-compliant languages. The CLS rules define a subset of the Common Type System.

1. **The**[**Virtual Execution System**](https://en.wikipedia.org/wiki/Virtual_Execution_System)**(VES)**

The VES loads and executes CLI-compatible programs, using the metadata to combine separately generated pieces of code at runtime.

MetaData refers to certain data structures embedded within the [Common Intermediate Language](https://en.wikipedia.org/wiki/Common_Intermediate_Language) (CIL) code that describes the high-level structure of the code. Metadata describes all [classes](https://en.wikipedia.org/wiki/Class_(computer_science)) and class members that are defined in the assembly, and the classes and class [members](https://en.wikipedia.org/wiki/Method_(computer_science)) that the current assembly will call from another assembly. The metadata for a method contains the complete description of the method, including the class (and the assembly that contains the class), the [return type](https://en.wikipedia.org/wiki/Return_type) and all of the method [parameters](https://en.wikipedia.org/wiki/Parameter_(computer_science)).

The **Virtual Execution System** (**VES**) is a [run-time system](https://en.wikipedia.org/wiki/Run-time_system) of the [Common Language Infrastructure](https://en.wikipedia.org/wiki/Common_Language_Infrastructure) CLI which provides an environment for executing [managed code](https://en.wikipedia.org/wiki/Managed_code)

**Common Type System** (**CTS**) defines how types are declared, used, and managed in the common language runtime, and is also an important part of the runtime's support for cross-language integration. The common type system performs the following functions:

* Establishes a framework that helps enable cross-language integration, type safety, and high-performance code execution.
* Provides an object-oriented model that supports the complete implementation of many programming languages.
* Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.
* Provides a library that contains the primitive data types (such as [Boolean](https://msdn.microsoft.com/en-us/library/system.boolean(v=vs.100).aspx), [Byte](https://msdn.microsoft.com/en-us/library/system.byte(v=vs.100).aspx), [Char](https://msdn.microsoft.com/en-us/library/system.char(v=vs.100).aspx), [Int32](https://msdn.microsoft.com/en-us/library/system.int32(v=vs.100).aspx), and [UInt64](https://msdn.microsoft.com/en-us/library/system.uint64(v=vs.100).aspx)) used in application development.

**Common Language Specification** (CLS), which is a set of basic language features needed by many applications, has been defined. The CLS rules define a subset of the [Common Type System](https://msdn.microsoft.com/en-us/library/zcx1eb1e(v=vs.100).aspx); that is, all the rules that apply to the common type system apply to the CLS, except where stricter rules are defined in the CLS

The CLR 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.

**Dotnet Framework 4.6.2 Features**

1. Debugging improvements -The *unmanaged debugging API* has been enhanced in the .NET Framework 4.6.2 to perform additional analysis when a [NullReferenceException](https://msdn.microsoft.com/en-us/library/system.nullreferenceexception(v=vs.110).aspx) is thrown so that it is possible to determine which variable in a single line of source code is null.

### 2. Converting Windows Forms and WPF apps to UWP apps

Windows now offers capabilities to bring existing Windows desktop apps, including WPF and Windows Forms apps, to the Universal Windows Platform (UWP).

### 3. Character categories

Characters in the .NET Framework 4.6.2 are classified based on the [Unicode Standard, Version 8.0.0](http://www.unicode.org/versions/Unicode8.0.0/).

**4. Improved support for localized error messages in data annotation validators**  
Data annotation validators enable you to perform validation by adding one or more attributes to a class property. The attribute's [ValidationAttribute.ErrorMessage](https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.validationattribute.errormessage(v=vs.110).aspx) element defines the text of the error message if validation fails.

public class RatingInfo

{

[Required(ErrorMessage = "The rating must be between 1 and 10.")]

[Display(Name = "Your Rating")]

public int Rating { get; set; }

}

<https://docs.microsoft.com/en-us/dotnet/standard/collections/index>

<https://docs.microsoft.com/en-us/dotnet/standard/collections/selecting-a-collection-class>

<https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference>

Implementations of .NET include the .NET Framework, .NET Core, and Mono. There is an API specification common to all implementations of .NET that's called the .NET Standard.

The .NET Standard is also a [target framework](https://docs.microsoft.com/en-us/dotnet/standard/glossary#target-framework). If your code targets a version of the .NET Standard, it can run on any .NET implementation which supports that version of the .NET Standard.

Each implementation of .NET includes the following components:

* One or more runtimes. Examples: CLR for .NET Framework, CoreCLR and CoreRT for .NET Core.
* A class library that implements the .NET Standard and may implement additional APIs. Examples: .NET Framework Base Class Library, .NET Core Base Class Library.

### NET Core

.NET Core is a cross-platform implementation of .NET and designed to handle server and cloud workloads at scale. It runs on Windows, macOS and Linux. It implements the .NET Standard, so code that targets the .NET Standard can run on .NET Core. ASP.NET Core runs on .NET Core

### .NET Framework

The.NET Framework is the original .NET implementation that has existed since 2002. It's the same .NET Framework that existing .NET developers have always used. Versions 4.5 and later implement the .NET Standard, so code that targets the .NET Standard can run on those versions of the .NET Framework.

### Mono

Mono is a .NET implementation that is mainly used when a small runtime is required. It is the runtime that powers Xamarin applications on Android, Mac, iOS, tvOS and watchOS and is focused primarily on a small footprint.+

It supports all of the currently published .NET Standard versions.

## **.NET runtimes**

A runtime is the execution environment for a managed program. The OS is part of the runtime environment but is not part of the .NET runtime. Here are some examples of .NET runtimes:1

* Common Language Runtime (CLR) for the .NET Framework
* Core Common Language Runtime (CoreCLR) for .NET Core
* .NET Native for Universal Windows Platform
* The Mono runtime for Xamarin.iOS, Xamarin.Android, Xamarin.Mac, and the Mono desktop framework

Use .NET Core for your server application when:

* You have cross-platform needs.
* You are targeting microservices.
* You are using Docker containers.
* You need high-performance and scalable systems.
* You need side-by-side .NET versions per application.

+

Use .NET Framework for your server application when:

* Your app currently uses .NET Framework (recommendation is to extend instead of migrating).
* Your app uses third-party .NET libraries or NuGet packages not available for .NET Core.
* Your app uses .NET technologies that aren't available for .NET Core.
* Your app uses a platform that doesn’t support .NET Core.

The [.NET Standard](https://github.com/dotnet/standard) is a formal specification of .NET APIs that are intended to be available on all .NET implementations.

When choosing a .NET Standard version, you should consider this trade-off:

* The higher the version, the more APIs are available to you.
* The lower the version, the more platforms implement it.

The following characteristics best define .NET Core:

* **Flexible deployment:** Can be included in your app or installed side-by-side user- or machine-wide.
* **Cross-platform:** Runs on Windows, macOS and Linux; can be ported to other operating systems. The [supported Operating Systems (OS)](https://github.com/dotnet/core/blob/master/roadmap.md), CPUs and application scenarios will grow over time, provided by Microsoft, other companies, and individuals.
* **Command-line tools:** All product scenarios can be exercised at the command-line.
* **Compatible:** .NET Core is compatible with .NET Framework, Xamarin and Mono, via the [.NET Standard](https://docs.microsoft.com/en-us/dotnet/standard/net-standard).
* **Open source:** The .NET Core platform is open source, using MIT and Apache 2 licenses. Documentation is licensed under [CC-BY](https://creativecommons.org/licenses/by/4.0/). .NET Core is a [.NET Foundation](https://dotnetfoundation.org/) project.
* **Supported by Microsoft:** .NET Core is supported by Microsoft, per [.NET Core Support](https://www.microsoft.com/net/core/support/)

The major differences between .NET Core and the .NET Framework:

* **App-models** -- .NET Core does not support all the .NET Framework app-models, in part because many of them are built on Windows technologies, such as WPF (built on top of DirectX). The console and ASP.NET Core app-models are supported by both .NET Core and .NET Framework.
* **APIs** -- .NET Core contains many of the same, but fewer, APIs as the .NET Framework, and with a different factoring (assembly names are different; type shape differs in key cases). These differences currently typically require changes to port source to .NET Core. .NET Core implements the [.NET Standard](https://docs.microsoft.com/en-us/dotnet/standard/net-standard) API, which will grow to include more of the .NET Framework BCL API over time.
* **Subsystems** -- .NET Core implements a subset of the subsystems in the .NET Framework, with the goal of a simpler implementation and programming model. For example, Code Access Security (CAS) is not supported, while reflection is supported.
* **Platforms** -- The .NET Framework supports Windows and Windows Server while .NET Core also supports macOS and Linux.
* **Open Source** -- .NET Core is open source, while a [read-only subset of the .NET Framework](https://github.com/microsoft/referencesource) is open source.

Type-safe code accesses only the memory locations it is authorized to access. (For this discussion, type safety specifically refers to memory type safety and should not be confused with type safety in a broader respect.) For example, type-safe code cannot read values from another object's private fields. It accesses types only in well-defined, allowable ways.

DotNet 4.7.1 features

## Support for Runtime Feature Detection - [This new API](https://github.com/dotnet/corefx/issues/17116) provides a way to detect whether a particular runtime supports a certain feature or not. At compile time the API provides a way to do that statically through reflection

## ValueTuple is Serializable

The [System.ValueTuple](https://docs.microsoft.com/en-us/dotnet/api/System.ValueTuple?view=netframework-4.7.1) types in .NET Framework 4.7.1 are now marked as Serializable, which allows binary serialization as shown in the example below.

## 3. ASP.NET HttpCookie parsing

var IsValidCookie = HttpCookie.TryParse("CookieName:Monster; path=/", out HttpCookie myCookie);

* 1. Garbage collection .NET Framework 4.7.1 brings in changes in [Garbage Collection (GC)](https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/) to improve the allocation performance, especially for Large Object Heap (LOH) allocations. This is due to an architectural change to split the heap’s allocation lock into 2, for Small Object Heap (SOH) and LOH. Applications that make a lot of LOH allocations, should see a reduction in allocation lock contention, and see better performance.

C# 7.1 adds the [language version selection](https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-1#language-version-selection) configuration element, three new language features and new compiler behavior.

The new language features in this release are:

* [async Main method](https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-1#async-main)
  + The entry point for an application can have the async modifier.
* [default literal expressions](https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-1#default-literal-expressions)
  + You can use default literal expressions in default value expressions when the target type can be inferred.
* [Inferred tuple element names](https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-1#inferred-tuple-element-names)
  + The names of tuple elements can be inferred from tuple initialization in many cases.

static async Task Main()

{

await SomeAsyncMethod();

}

Func<string, bool> whereClause = default;

int count = 5;

string label = "Colors used in the map";

var pair = (count, label);