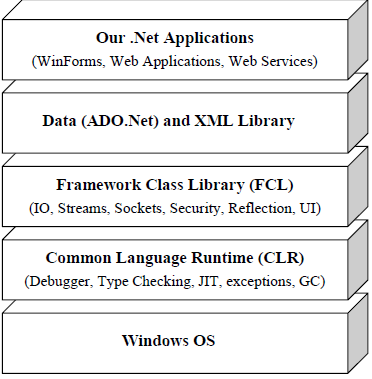
**Dot NET Framework introduction**

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:

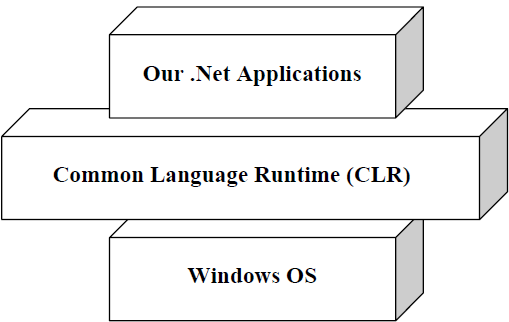
* Provide a runtime environment that minimizes software deployment and versioning conflicts.
* Enable the safe execution of code.
* Use industry standards for all communication to enable integration with non-.NET code.
* Provide a consistent developer experience across all types of applications in a way that is language- and platform-independent.
* Provide a runtime environment that minimizes or eliminates the performance problems of scripted or interpreted environment.

The .NET Framework consists of the common language runtime and the .NET Framework class library.



**Common Language Runtime(CLR)**

It is the foundation of the .NET framework. It is the responsibility of the runtime to take care the code execution of the program. It is a framework layer that resides above the OS and handles the execution of all the .Net applications. Programs don’t directly communicate with the OS but go through the CLR.

****

The CLR manages memory, thread execution, code execution, code safety verification, compilation, and other system services. These features are intrinsic to the managed code that runs on the CLR.

The CLR is the runtime for executing managed code. C# is one of several managed languages that get compiled into managed code. Managed code is packaged into an assembly, in the form of either an executable file (an .exe) or a library (a .dll), along with type information, or metadata.

Managed code is represented in Intermediate Language or IL. When the CLR loads an assembly, it converts the IL into the native code of the machine. This conversion is done by the CLR’s JIT (Just-In-Time) compiler. An assembly retains almost all of the original source language constructs, which makes it easy to inspect and even generate code dynamically.

The CLR performs as a host for numerous runtime services. Examples of these services include memory management, the loading of libraries, and security services. The CLR is language-neutral, allowing developers to build applications in multiple languages (e.g., C#, Visual Basic .NET, Managed C++, Delphi.NET, Chrome .NET, and J#).

**Common Type System(CTS)**

CTS define how types are declared, used and managed in the CLR, 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 ensures that objects written in different languages can interact with each other.
* Provides a library that contains the primitive data types (such as Boolean, Byte, Char, Int32, and UInt64) used in application development.

This makes it possible for the 2 languages to communicate with each other by passing/receiving parameters to and from each other. A type is a representation of data(such as int).

All types in the .NET Framework are either **value types** or **reference types.**

Value types are data types whose objects are represented by the object's actual value. If an instance of a value type is assigned to a variable, that variable is given a fresh copy of the value.

Reference types are data types whose objects are represented by a reference (similar to a pointer) to the object's actual value. If a reference type is assigned to a variable, that variable references (points to) the original value. No copy is made.

**Common Language Specification**

CLS is a set of basic language features that .Net Languages needed to develop Applications and Services.

It is a subset of the CTS. The CLS establishes the minimum set of rules to promote language interoperability.

When there is a situation to communicate Objects written in different .Net Complaint languages , those objects must expose the features that are common to all the languages . Common Language Specification (CLS) ensures complete interoperability among applications, regardless of the language used to create the application.

Microsoft has defined CLS, which are nothing but guidelines, that language should follow so that it can communicate with other .NET languages in a seamless manner.

**The Framework Class Library(FCL) or Base class Library(BCL)**

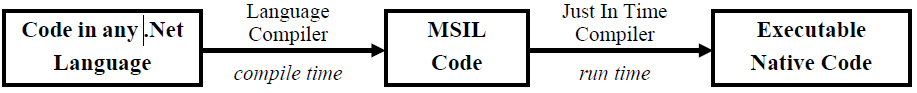
The .Net Framework class library (FCL) provides the core functionality of .Net Framework architecture. The .Net Framework Class Library (FCL) includes a huge collection of reusable classes, interfaces, and value types that ease and optimize the development process and provide access to system functionality.

The .Net Framework provides a huge Framework(or Base) Class Library for common, usual tasks. BCL contains thousands of classes to provide access to Windows API and common functions like String Manipulation, Common Data Structures, IO, Stream, Thread, 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.

**MSIL(Microsoft Intermediate Language) or CIL(Common Intermediate Language) or IL(Intermediate Language)**

When All .NET source code compile it is converted to an intermediate code known as MSIL which is interpreted by the CLR. MSIL is OS and hardware independent code. This MSIL is converted to binary executable code(native code) at the point where the software is installed, or at run time by Just-In-Time(JIT compiler).



**Just-In-Time compilers(JITers)**

When IL compiled code needs to be executed, the CLR invokes the JIT compiler, which compile the IL code to native executable code(.exe or .dll) that is designed for specific machine and OS. JITers in many ways are different from traditional compilers as they compile the IL to native code only when desired. Eg when a function is called, the IL of the function’s body is converted to native code **just in time.** If some IL code is converted to native code, then the next time it’s needed, the CLR reuses the same copy without re-compiling.

As JITers are aware of the specific processor and OS at runtime, they can optimize the code efficiently resulting in very robust applications. Also, since a JIT compiler knows the exact current state of executable code, they can also optimize the code by in-lining small function calls.

**Garbage Collection**

CLR automatically manages memory thus eliminating memory leaks. When any variable and reference are not referred, GC automatically releases those memories and dynamically allocated memory space thus providing efficient memory management. The presence of a standard Garbage Collector frees the programmer from keeping track of dangling data.

Garbage collection does not happen immediately after an object. It happens periodically, although not to a fixed schedule

It is the advantage over C++ language.

**Managed Code**

Managed code runs inside the environment of CLR i.e. .NET runtime. In short, all IL are managed code. However, if you are using some third party software example VB6 or VC++ component they are unmanaged code, as .NET runtime (CLR) does not have control over the source code execution of these languages. This means that the CLR looks after your application by managing memory, handling security, allowing cross language debugging, and so on.

**Assemblies**

An assembly is the basic unit of deployment in .NET and is also the container for all types. An assembly contains compiled types with their IL code, runtime resources, and information to assist with versioning, security, and referencing other assemblies. An assembly also defines a boundary for type resolution and security permissioning. In general, an assembly comprises a single Windows Portable Executable (PE) file—with an **.exe** extension in the case of an application, or a **.dll** extension in the case of a reusable library. Assemblies also contain meta information which describes the program’s types, members, and attributes.

An assembly contains four kinds of things:

*An assembly manifest:*

Provides information to the .NET runtime, such as the assembly’s name, version, requested permissions, and other assemblies that it references

*An application manifest:*

Provides information to the operating system, such as how the assembly should be deployed and whether administrative elevation is required

*Compiled types:*

The compiled IL code and metadata of the types defined within the assembly.

*Resources:*

Other data embedded within the assembly, such as images and localizable text Of these, only the assembly manifest is mandatory, although an assembly nearly always contains compiled types.

**Windows Presentation Foundation(WPF)**

Windows Presentation Foundation (WPF) provides a unified programming model for building rich Windows smart client user experiences that incorporate UI, media, and documents.

Windows Presentation Foundation (WPF) is a next-generation presentation system for building Windows client applications with visually stunning user experiences. With WPF, we can create a wide range of both standalone and browser-hosted applications.

The core of WPF is a resolution-independent and vector-based rendering engine that is built to take advantage of modern graphics hardware. WPF extends the core with a comprehensive set of application-development features that include Extensible Application Markup Language (XAML), controls, data binding, layout, 2-D and 3-D graphics, animation, styles, templates, documents, media, text, and typography. WPF is included in the Microsoft .NET Framework, so you can build applications that incorporate other elements of the .NET Framework class library. WPF came to us as a revolution to UI development.

WPF exists as a subset of .NET Framework types that are for the most part located in the **System.Windows** namespace.

As the name says all, WPF is actually a new framework introduced with .NET framework 3.0 which actually puts forward a new set of classes and assemblies which allow us to write programs more efficiently and flexibly. It uses Direct3D rendering which employs graphics cards to render the output on the screen. Thus the drawing in the form will be smooth and also there is a chance to utilize the hardware capabilities installed in your machine. In case of traditional GDI forms application, it is not possible to use advanced graphics capabilities and hence Windows Forms application will always be inefficient in comparison to WPF. Another important thing that I must address in this regard, GDI Windows forms application uses Operating system controls to build its application. Thus it is basically very hard to customize them in your own application. WPF controls are actually drawn over the screen, and hence you can customize controls totally and modify their behavior when required.

The benefits of WPF over its predecessor, Windows Forms, are as follows:

• It supports sophisticated graphics, such as arbitrary transformations, 3D rendering, and true transparency.

• Its primary measurement unit is not pixel-based, so applications display correctly at any DPI (dots per inch) setting.

• It has extensive dynamic layout support, which means you can localize an application without danger of elements overlapping.

• Rendering uses DirectX and is fast, taking good advantage of graphics hardware acceleration.

• User interfaces can be described declaratively in XAML files that can be maintained independently of the “code-behind” files—this helps to separate appearance from functionality.

WPF’s size and complexity, however, make for a steep learning curve. The types for writing WPF applications are in the System.Windowsnamespace and all subnamespaces except for System.Windows.Forms.

**Windows Communication Foundation (WCF)**

Windows Communication Foundation (WCF) is a sophisticated communications infrastructure for building service-oriented applications which is introduced in Framework 3.0. The WCF client uses Simple Object Access Protocol (SOAP) to communicate with the server. Using WCF, we can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data. A few sample scenarios include:

* A secure service to process business transactions.
* A service that supplies current data to others, such as a traffic report or other monitoring service.
* A chat service that allows two people to communicate or exchange data in real time.
* A dashboard application that polls one or more services for data and presents it in a logical presentation.
* Exposing a workflow implemented using Windows Workflow Foundation as a WCF service.
* A Silverlight application to poll a service for the latest data feeds.

While creating such applications was possible prior to the existence of WCF, WCF makes the development of endpoints easier than ever. In summary, WCF is designed to offer a manageable approach to creating Web services and Web service clients.

WCF, Remoting, and Web Services are all alike in that they implement the following basic model in allowing a client and server application to communicate:

• On the server, we indicate what methods we would like remote client applications to be able to call.

• On the client, we specify or infer the *signatures* of the server methods we would like to call.

• On both the server and the client, we choose a transport and communication protocol (in WCF, this is done through a *binding*).

• The client establishes a connection to the server.

• The client calls a remote method, which executes transparently on the server.

WCF further decouples the client and server through service contracts and data contracts. Conceptually, the client sends an (XML or binary) message to an endpoint on a remote *service*, rather than directly invoking a remote *method*. One of the benefits of this decoupling is that clients have no dependency on the .NET platform or on any proprietary communication protocols.