**Components of .NET Framework 4.0**

The .NET Framework provides the help to developers create new application. It provides all the necessary components to develop and run .NET based programs on your system. The components of .NET Framework 4.0 architecture are as follows:

* Common Language Runtime (CLR)
* Common Type System (CTS)
* Metadata and Assemblies
* .NET Framework class library
* Windows Forms
* ASP.NET and ASP.NET AJAX
* ADO.NET
* Windows Workflow Foundation
* Windows Presentation Foundation
* Windows Communication Foundation
* Windows CardSpace
* LINQ

Let’s now discuss each of them in detail.

**CLR**

CLR is one of the most important components of .NET Framework, better known as the runtime. It provides a runtime environment to run the code and provides various services to develop the application easily. The services provided by CLR are memory management, exception handling, debugging, security, thread execution, code execution, code safety, verification and compilation. It also supports versioning to any language that the CLR interacts with. It means CLR can host a variety of languages and provide common tools to these languages; thereby, ensuring interoperability between the codes.

The runtime manages the execution of the code. Therefore, the code that works on the CLR is called the managed code. CLR also supports services that the application uses to access various resources, such as collections, arrays, and operating system folders. The runtime automatically releases the objects when they are no longer in use. This automatic memory management resolves the issues of memory leaks and invalid memory references.

The runtime imposes Code Access Security(CAS), which is the CLR’s security system that enforces security policies by preventing unauthorized access to protected resources and operations. CAS also helps in making the code robust, which implies that which makes the code extensible and reusable. It implements a strict type and code verification infrastructure called CTS.

The runtime also accelerates the developer’s productivity, as the developer can write the applications in any language, such as C#, Visual Basic, or Visual C++. They can also take advantage of the runtime, class library, and other components written in the other development languages of the .NET platform.

Let’s now explore managed code, unmanaged code, and memory management in CLR.

**Managed code**

Managed code is the code that is executed by the CLR The application that are created using managed code automatically have CLR services, such as type checking, security, and automatic garbage collection These services help to provide platform and language independence to managed code applications The CLR compiles the applications to Intermediate Language (IL) and not the machine code This IL along with the metadata that describes the attributes ,classes , and methods of the code reside in an assembly The compilation takes place in the managed execution environment, which assures the working of the code

The process of executing a piece of managed code is as follows:

1. Selecting a language compiler

2. Compiling the code to IL

3.Compiling IL to native code

4. Executing the code

The benefits provided by the CLR can be used by selecting one or more language compilers, such as Visual Basic, C#, Visual C++, J#, or any of the third party compilers, such as COBOL ,Perl, or Eiffel The language compiler determines the syntax the must be used by the code

When you compile the code into managed code , the compiler the source code into IL which is CPU independent IL must be converted into CPU specific code by the Just in Time (JIT) compiler before the execution of the code When you compile your source code into IL , the required metadata is generated The metadata contains the definition of types , member signature ,the members in the code, and other details that the code uses at the time of execution The runtime locates and extracts the metadata from the file during execution While executing the application, a JIT compiler translates the IL into native code After compiling the code is passed through a verification process that examines the IL and metadata to check whether the code is safe, such as it should be able to access only those memory locations which it is authorized to access