**.Net Architecture**

**1. What is .Net Architecture?**

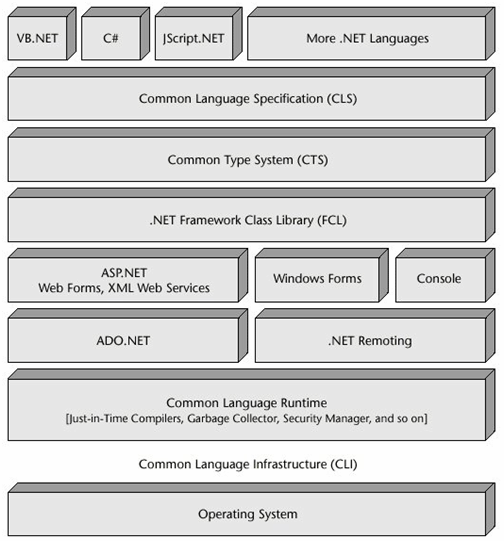
.Net is a programming platform created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services.

Major parts for .Net Framework

CLR - Common Language Runtime, is the core component of .Net framework, which handles the execution of source code

MSIL - MSIL or IL stands for Microsoft Intermediate Language; compiler translates your source code into Microsoft intermediate language or object code

CTS - Common Type System defines how types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for cross-language integration

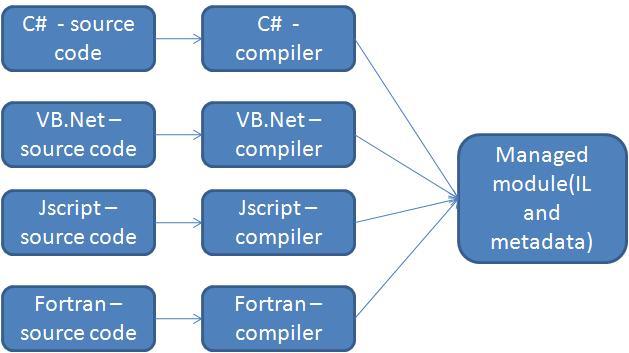


**2. How source code is compiled to managed code?**

.Net allows developers to use different language for developing application. Each language has it own compiler to compile the source code in to IL or managed. Managed module is a standard Windows portable executable (PE) file that requires the CLR to execute. We need to install the CLR (.Net Framework) on our machine to execute any managed module.

Managed Module is composed of the following parts:

* PE header - The PE header indicates the type of file--GUI, CUI, or DL and also has a timestamp indicating when the file was built.
* CLR header - It includes the version of the CLR required to execute, entry point method(Main method), resources, strong name etc
* Metadata - describe the types and members defined in source code
* Intermediate Language (IL) Code - code that was produced by the compiler as it compiled the source Code



**3. What is use of Metadata?**

* Metadata removes the need for header and library files when compiling, because all the information about the referenced types/members is contained in one file along with the IL that implements those type/members. Compilers can read metadata directly from managed modules.
* Visual Studio uses metadata to help you write code. Its IntelliSense feature parses metadata to tell you what methods a type offers and what parameters that method expects.
* The CLR code verification process uses metadata to ensure that your code performs only "safe" operations. Verification is discussed shortly.
* Metadata allows an object's fields to be serialized into a memory block, remoted to another machine, and then deserialized, recreating the object and its state on the remote machine.
* Metadata allows the garbage collector to track the lifetime of objects. For any object, the garbage collector can determine the type of the object, and from the metadata it knows which fields within that object refer to other objects.

**4. What is use of mscoree.dll?**

Mscoree.dll is an assembly containing the .NET Framework functions. This file will reside in % windir% \system32. When you install .Net framework, it will replace mscoree.dll if the existing one is older than the one it carries, and it will leave it alone if the existing one is newer.

**5. How to call unmanaged code inside managed code?**

Developers creating the application using .Net framework are managed code and it will be executed by CLR(common language runtime). Whereas unmanaged codes are source code generated by WIN32 APIs, valuable external libraries, COM components, and COM+ services. To call a functions located in an unmanaged dll using .net canbe done using technique is called as PInvoke.

When platform invoke calls an unmanaged function it firstly locates the DLL containing the function and then loads it into memory. It searches for the function in memory and locates its memory address. After that it pushes the function arguments into the stake and marshaling data as required. Finally the control is transferred to the unmanaged function. Platform invoke throws exceptions generated by the unmanaged function to the managed caller.

**Example:**

Let us create a sample application which will call the message box from one of WIN32 API called "user32.dll". WIN32 APIs may contain two versions for each function that handles characters and strings. One for the ANSI version (MessageBoxA), and the other for the Unicode version (MessageBoxW).

**Step1 :**

Create the wrapping class for unmanaged dll functions, this classwill allows the pinvoke to handle the underlying exported functions automatically. We need to define a static method for each DLL function we want to call. Below class "Unmanaged" defines a static method for calling "MessageBox"

Imports System.Runtime.InteropServices

Public Class Unmanaged

Declare Auto Function MessageBox Lib "User32.dll" (ByVal Hwnd As Integer,

ByVal Text As String,ByVal Caption As String, ByVal Type As Integer) As Integer

End Class

**Step 2:**

Let us create a sample windows application to consume the unmanaged function in our application.

Public Class Form1

Private Sub Form1\_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)

Handles MyBase.Load

Unmanaged.MessageBox(0, "Hello world!!!", "PInvoke example", 0)

End Sub

End Class

**Step 3:**

Output



**6. How can we use COM Components in .NET?**

In general, COM code does not operate within the framework provided by the CLR and it can't use any of the CLR services. .Net framework can call COM components using RunTime-Callable Wrapper (RCW).

**Method 1:**

COM type libraries also contain metadata describing the public interface to COM components. The job of an RCW is to convert this COM metadata to .NET metadata. One tool for performing this conversion is called tlbimp (type library importer), and it is provided as part of the .NET Framework

Tlbimp reads the metadata from a COM type library and creates a matching CLR assembly for calling the COM component.

tlbimp PhysServer.dll /out:NETPhysServer.dll

**Method 2:**

Create a new project and click Add Reference. For example, if we add Microsoft Word COM components then it will generate "Microsoft.Office.Interop.Word.dll" file in /bin directory of the application. This is the .Net dll for COM

