Unit-1
Introduction to .Net framework

1. 1 Introduction to Dot Net Framework

The .NET is the technology from Microsoft, on which all other Microsoft technologies will be depending on in future.

It is a major technology change. Just like the computer world moved from DOS to Windows, now they are moving to .NET. But don't be surprised if you find anyone saying that "I do not like .NET and I would stick with the good old COM and C++". There are still lot of people who like to use the bullock-cart instead of the latest Honda car.

.NET technology was introduced by Microsoft, to catch the market from the SUN's Java. Few years back, Microsoft had only VC++ and VB to compete with Java, but Java was catching the market very fast. With the world depending more and more on the Internet/Web and java related tools becoming the best choice for the web applications, Microsoft seemed to be loosing the battle. Thousands of programmers moved to java from VC++ and VB. To recover the market, Microsoft announced .NET.

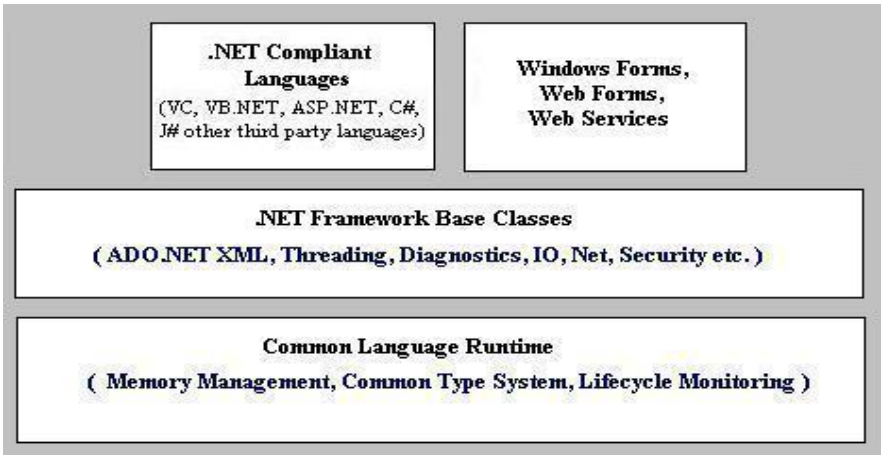
.NET framework comes with a single class library. And thats all programmers need to learn!! Whether they write the code in C# or VB.NET or J#, it doesn't matter, you just use the .NET class library. There is no classes specific to any language. There is nothing more you can do in a language, which you can't do in any other .NET language. You can write code in C# or VB.NET with the same number of lines of code, same performance and same efficiency, because everyone uses same .NET class library.

Features of .NET

- It is a platform neutral framework.
- It is a layer between the operating system and the programming language.
- It supports many programming languages, including VB.NET, C# etc.
- .NET provides a common set of class libraries, which can be accessed from any .NET based programming language. There will not be separate set of classes and libraries for each language. If you know any one .NET language, you can write code in any .NET language.
- In future versions of Windows, .NET will be freely distributed as part of operating system and users will never have to install .NET separately.

Major Components of .NET

The diagram given below describes various components of .NET Framework.



The .NET framework can only be exploited by languages that are compliant with .NET. Most of Microsoft languages have been made to fully comply with .NET.
.NET also introduces Web Forms, Web Services and Windows Forms. The reason why they have been shown separately and not as a part of a particular language is that these
technologies can be used by any .NET compliant language. For example Windows Forms is used by VC, VB.NET, C# all as a mode of providing GUI.
The next component of .NET is the .NET Framework Base Classes. These are the common class libraries (much like Java packages) that can be used by any .NET compliant language. These classes provide the programmers with a high degree of functionality that they can use in their programs. For example their are classes to handle reading, writing and manipulating XML documents, enhanced ADOs etc.
The bottom most layer is the CLR - the common runtime language.

Origin of .net Technology

1. OLE Technology
2. COM Technology
3. .net Technology

OLE Technology(Object Linking and Embedding)

- Easy interprocess communication
- Embed documents from one application into another application
- To enable one application to manipulate objects located in another application
- Ex: inter operability between various products such as MS word and MSExcel

COM Technology(Component Object Model)

- Monolithic approach leads to many problem of maintainability and testing
- A program is broken into number of independent components where each one offers a particular service
- Each component can be developed and tested independently and then integrated into main system.
- Benefits:
- Reduces the overall complexity of software.
- Enables distributed development across multiple organization or departments.
- Enhances software maintainability

.net Technology

- Third generatiom component model
- IPC in COM is replaced by Intermediate Language(IL or MSIL)
- Interoperability by compiling code into IL.
- Metadata

1.2 Common Language Runtime (CLR)

The CLR is the heart of .NET framework. It is .NET equivalent of Java Virtual Machine (JVM). It is the runtime that converts a MSIL (Micro Soft Intermediate Language) code into the host machine language code, which is then executed appropriately.
The CLR provides a number of services that include:

- Loading and execution of codes
- Memory isolation for application
- Verification of type safety
- Compilation of IL into native executable code
- Providing metadata
- Automatic garbage collection
- Enforcement of Security

- Interoperability with other systems
- Managing exceptions and errors
- Provide support for debugging and profiling

1.3 Common Type System (CTS)

The language interoperability, and .NET Class Framework, are not possible without all the language sharing the same data types. What this means is that an "int" should mean the same in VB, VC++, C# and all other .NET compliant languages. Same idea follows for all the other data types. This is achieved through introduction of Common Type System (CTS).

CTS, much like Java, defines every data type as a Class. Every .NET compliant language must stick to this definition. Since CTS defines every data type as a class; this means that only Object-Oriented (or Object-Based) languages can achieve .NET compliance. Given below is a list of CTS supported data types:

Data Type	**Description**
System.Byte	1-byte unsigned integer between 0-255
System.Int16	2-bytes signed integer in the following range: 32,678 to 32,767
System.Int32	4-byte signed integer containing a value in the following range: -2,147,483,648 to 2,147,483,647
System.Int64	8-byte signed integer containing a value from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
System.Single	4-byte floating point. The value limits are: **for negative values:** *-3.402823E38 to - 1.401298E-45* **for positive values:** 1.401298E-45 TO 30402823E38
System.Double	8-bytes wide floating point. The value limits are: **for negative values:** *-1.79769313486231E308 to - 4.964065645841247E-324* **for positive values:** *4.964065645841247E-324 to 1.79769313486232E308*
System.Object	4-bytes address reference to an object
System.Char	2-bytes single Unicode Character.
System.String	string of up to 2 billion Unicode characters.
System.Decimal	12-bytes signed integer that can have 28 digits on either side of decimal.
System.Boolean	4-Bytes number that contains true(1) or false (0)

1.4 Common Language Specification (CLS)

One of the obvious themes of .NET is unification and interoperability between various programming languages. In order to achieve this; certain rules must be laid and all the languages

must follow these rules. In other words we can not have languages running around creating their own extensions and their own fancy new data types. CLS is the collection of the rules and constraints that every language (that seeks to achieve .NET compatibility) must follow. Microsoft has defined three level of CLS compatibility/compliance. The goals and objectives of each compliance level have been set aside. The three compliance levels with their brief description are given below:

Compliant producer

The component developed in this type of language can be used by any other language.

Consumer

The language in this category can use classes produced in any other language. In simple words this means that the language can instantiate classes developed in other language. This is similar to how COM components can be instantiated by your ASP code.

Extender

Languages in this category can not just use the classes as in CONSUMER category; but can also extend classes using inheritance.

Languages that come with Microsoft Visual Studio namely Visual C++, Visual Basic and C#; all satisfy the above three categories. Vendors can select any of the above categories as the targeted compliance level(s) for their languages.

1.5 Microsoft Intermediate Language (MSIL)

A .NET programming language (C#, VB.NET, J# etc.) does not compile into executable code; instead it compiles into an intermediate code called Microsoft Intermediate Language (MSIL). As a programmer one need not worry about the syntax of MSIL - since our source code in automatically converted to MSIL. The MSIL code is then send to the CLR (Common Language Runtime) that converts the code to machine language which is then run on the host machine.

MSIL is similar to Java Byte code. A Java program is compiled into Java Byte code (the .class file) by a Java compiler, the class file is then sent to JVM which converts it into the host machine language.

Managed Code

The role of CLR doesn't end once we have compiled our code to MSIL and a JIT compiler has compiled this to native code. Code written using the .NET framework, is managed code when it is executed. This stage is usually referred to as being at runtime. This means that the CLR looks after our applications, by managing memory, handling security, allowing cross language debugging and so on. By contrast, applications that do not run under the control of the CLR are said to be unmanaged and certain languages such as C++ can be used to write such applications, that for example, to access low level functions of the operating systems. However in C# we can only write code that runs in a managed environment.

Unified classes

The term .NET framework refers to the group of technologies that form the development foundation for the Microsoft .NET platform. The key technologies in this group are the run time and the class libraries.

The run time is responsible for managing your code and providing services to it while it executes, playing a role similar to that of the Visual Basic 6.0 run time.

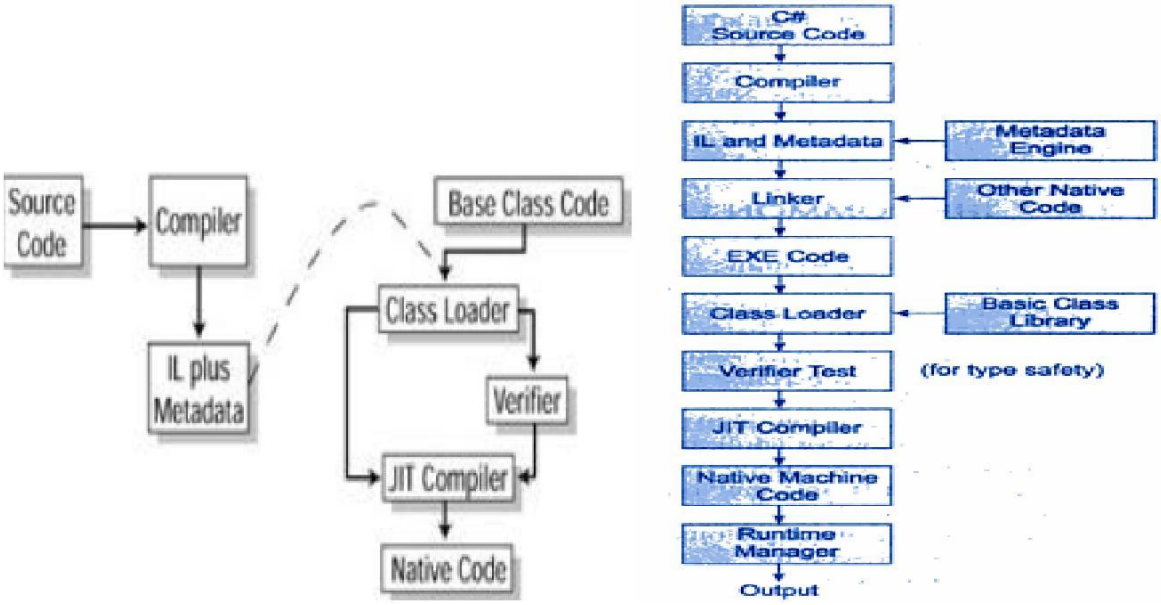
The .NET programming languages including Visual Basic .NET , Microsoft Visual C# and C++ managed extensions and many other programming languages from various vendors utilize .NET services and features through a common set of unified classes.

The .NET unified classes provide foundation of which you build your applications, regardless of the language you use. Whether you simply concating a string, or building a windows Services or a multiple-tier web-based applications, you will be using these unified classes.

The unified classes provide a consistent method of accessing the platforms functionality. Once you learn to use the class library, you'll find that all tasks follow the same uniform architecture, you no longer need to learn and master different API architecture to write your applications.
By building your applications on a unified, integrated framework, you maximize your return on the time you spend learning this framework, and you end up with more robust applications that are easy to deploy and maintain.

1.6 Just In Time Compiler

- To make it easy for language writers to port their languages to .NET, Microsoft developed a language akin to assembly language called Microsoft intermediate language (MSIL). To compile applications for .NET, compilers take source code as input and produce MSIL as output.
- MSIL itself is a complete language that you can write applications in. However, as with assembly language, you would probably never do so except in unusual circumstances. Because MSIL is its own language, each compiler team makes its own decision about how much of the MSIL it will support. However, if you're a compiler writer and you want to create a language that does interoperate with other languages, you should restrict yourself to features specified by the CLS.



- You write source code in C# and compile it using the C# compiler (csc.exe) into an EXE.
- The C# compiler outputs the MSIL code and a manifest into a read-only part of the EXE that has a **standard PE (Win32-portable executable) header.** When the compiler creates the output, it also imports a function named **_ *CorExeMain* from the .NET runtime.**
- When the application is executed, the operating system loads the PE, as well as any dependent dynamic-link libraries (DLLs), such as the one that exports the _ *CorExeMain* function (mscoree.dll), just as it does with any valid PE.

1.7 Framework Base Classes

The .NET Framework has an extensive set of class libraries. This includes classes for:

- **Data Access:** High Performance data access classes for connecting to SQL Server or any other OLEDB provider.
- **XML Supports:** Next generation XML support that goes far beyond the functionality of MSXML.
- **Directory Services:** Support for accessing Active Directory/LDPA using ADSI.
- **Regular Expression :** Support for above and beyond that found in Perl 5.
- **Queuing Supports:** Provides a clean object-oriented set of classes for working with MSMQ.

These class libraries use the CLR base class libraries for common functionality.

Base Class Libraries

The Base class library in the .NET Framework is huge. It covers areas such as:

- **Collection :** The System.Collections namespaes provides numerous collection classes.
- **Thread Support:** The System.Threading namespace provides support for creating fast, efficient, multi-threaded appliction.
- **Code Generation:** The System.CodeDOM namespace provides classes for generating source files in numerous language. ASP.NET uses these classes when converting ASP.NET pages into classes, which are subsequently compiled.
- **IO:** The System.IO provides extensive support for working with files and all other stream types.
- **Reflection:** The System.Reflection namespace provides support for load assemblies, examining the type with in assemblies, creating instances of types, etc.
- **Security:** The System.Security namespace provides support for services such as authentication, authorization, permission sets, policies, and cryptography. These base services are used by application development technologies like ASP.NET to build their security infrastructure.

The list of support base classes goes on forever in .NET, but if you ever find yourself lost looking for a specific class, you can use the WinCV tool to locate it. You can run this from the Start bar Run menu. The file is typically located in the c:\program files\ Microsoft.Net\FrameworkSDK\Bin directory.