Vimal Ghorecha

C# Programming Made Easy. Visual Studio 2008

Bibliographic information published by the German National Library:

The German National Library lists this publication in the National Bibliography; detailed bibliographic data are available on the Internet at http://dnb.dnb.de .

Imprint:

Copyright © 2013 GRIN Verlag
ISBN: 9783656675877

Vimal Ghorecha

C# Programming Made Easy. Visual Studio 2008

GRIN Verlag

Chapter 1: Introduction to Visual Studio 2008

Introduction to the Visual Studio Development System

Visual Studio 2008 provides advanced development tools, debugging features, database functionality, and innovative features for quickly creating tomorrow's cutting-edge applications across a variety of platforms.

Visual Studio 2008 includes enhancements such as visual designers for faster development with the .NET Framework 3.5, substantial improvements to web development tools and language enhancements that speed development with all types of data. Visual Studio 2008 provides developers with all the tools and framework support required to create compelling, expressive, AJAX-enabled web applications.

Developers will be able to take advantage of these rich client-side and server-side frameworks to easily build client-centric web applications that integrate with any back-end data provider, run within any modern browser and have complete access to ASP.NET application services and the Microsoft platform.

Rapid Application Development

To help developers rapidly create modern software, Visual Studio 2008 delivers improved language and data features, such as Microsoft Language Integrated Query (LINQ), that make it easier for individual programmers to build solutions that analyse and act on information.

Visual Studio 2008 also provides developers with the ability to target multiple versions of the .NET Framework from within the same development environment. Developers will be able to build applications that target the .NET Framework 2.0, 3.0 or 3.5, meaning that they can support a wide variety of projects in the same environment.

Breakthrough User Experience

Visual Studio 2008 offers developers new tools that speed creation of connected applications on the latest platforms including the web, Windows Vista, Office 2007, SQL Server 2008 and Windows Server 2008. For the web, ASP.NET, AJAX and other new technologies will enable developers to quickly create a new generation of more efficient, interactive and personalised web experiences.

Effective Team Collaboration

Visual Studio 2008 delivers expanded and improved offerings that help improve collaboration in development teams, including tools that help integrate database professionals and graphic designers into the development process.

Use the Microsoft .NET Framework 3.5

The .NET Framework enables the rapid construction of connected applications that provide outstanding end-user experiences by providing the building blocks (pre-fabricated software) for solving common programming tasks. Connected applications built on the .NET Framework model business processes effectively and facilitate the integration of systems in heterogeneous environments.

Together Visual Studio and the .NET Framework reduce the need for common plumbing code, reducing development time and enabling developers to concentrate on solving business problems.

The .NET Framework 3.5 builds incrementally on the .NET Framework 3.0. Enhancements have been made to feature areas including the base class library, Windows Workflow Foundation, Windows Communication Foundation, Windows Presentation Foundation and Windows CardSpace.

What's New in 2008

- Build applications that use the latest web technologies with improved support for AJAX and web controls and the Microsoft AJAX Library
- Create web applications more easily with an improved design surface and standards support
- Use data from any data source more smoothly with LINQ, a set of language extensions to Visual Basic and Visual C#
- Manage and build applications that target multiple versions of the .NET Framework. For the first time, you can use one tool to work on applications that run on .NET Framework versions 2.0, 3.0, and 3.5
- Ensure application correctness more easily with integrated unit testing in Visual Studio 2008 Professional Edition
- Discover the full power of the .NET Framework 3.5 with integrated tools that simplify building great user experiences and connected systems
- Build stunning user experiences with integrated designers for Windows Presentation Foundation. Experiences built with Windows Presentation Foundation can interoperate seamlessly with Windows Forms
- Create connected applications using new visual designers for Windows Communications Foundation and Windows Workflow Foundation
- Use Visual Studio’s professional development environment to build Microsoft Office-based solutions that are reliable, scalable and easy to maintain (available in Visual Studio 2008 Professional Edition only)
- Enhance collaboration between developers and designers to create more compelling user experiences

Feature Highlights

- Build applications for Windows, the web, the Microsoft Office system, the .NET Framework, SQL Server and Windows Mobile with integrated drag-and-drop designers
- Visual Studio integrates Visual Basic, Visual C# and Visual C++ to support a wide variety of development styles
- Editor features such as Edit and Continue and Microsoft IntelliSense simplify the cycle of designing, developing and debugging an application
- Deploy client applications easily with ClickOnce, which enables developers and IT pros to deploy an application and its prerequisites and then ensure that the application remains up-to-date
- Build applications which target the .NET Framework, shortening development time by reducing the need for infrastructure code and helping to enhance application security
- Use ASP.NET to speed the creation of interactive, highly appealing web applications and web services. Master Pages allow developers to easily manage a consistent site layout in one place
- A community of millions of developers ensures that developers can find partners and other community members addressing the same challenges

Visual Studio Editions with Features

Visual Studio is available in several editions: Standard, Professional, Tools for Office, and Team System.

The core languages included with Visual Studio — Visual Basic, Visual C++, Visual C#, and Visual J# — as well as Visual Web Developer are each also offered in separate Express editions.

The following table lists the different features and tools available with Express editions, Visual Studio Standard and Professional editions, and Visual Studio Tools for Office.

Feature	Express	Standard	Professional	Visual Studio Tools for Office
Programming languages included	Visual Basic, Visual C#, Visual C++, and Visual J# are single language. Visual Web Developer includes Visual C# and Visual Basic.	All	All	Visual Basic and Visual C# only.
User experience	Simplified menu options and defaults	Simplified menu options and defaults	Full	Full
Documentation	10mb "Getting Started"; Starter Kits targeted at first-time programmers 200mb optional MSDN Express	Full MSDN Library	Full MSDN Library	Full MSDN Library
IntelliSense	Yes	Yes	Yes	Yes
Code editor	Yes	Yes	Yes	Yes
Code snippets	Yes	Yes	Yes	Yes
Windows Forms designer	Not available with Visual Web Developer	Yes	Yes	Yes
Web Forms designer	Only available with Visual Web Developer	Yes	Yes	Yes
Mobile Device support	No	Yes	Yes	No
Database design tools to create and modify tables and stored procedures	Local only	Local and remote	Local and remote	Local and remote
Data access designers	Visual Basic, Visual C#, Visual C++, local only. Visual Web Developer includes local and remote.	Local and remote	Local and remote	Local and remote

Class Designer	No	Yes	Yes	Yes
Object Test Bench	No	Yes	Yes	Yes
XML editor support	XML editing available. Basic XSLT support available only with Visual Web Developer	Full XML and XSLT	Full XML and XSLT	Full XML and XSLT
Deployment tools	ClickOnce only	ClickOnce only	All tools	All tools
Extensibility	Add external tools to the menu only. Use 3rd party controls.	Consume extensions	Full	Full
Server Explorer Servers node	No	No	Yes	No
Reporting	SQL Server Reporting Services Add-in available with Visual Web Developer only	SQL Server Reporting Services	SQL Server Reporting Services and Crystal Reports	SQL Server Reporting Services
Source Code Control	No	MSSCCI-compatible (Visual SourceSafe sold separately)	MSSCCI-compatible (Visual SourceSafe sold separately)	MSSCCI-compatible (Visual SourceSafe sold separately)
Local debugging	Yes	Yes	Yes	Yes
Remote debugging	No	No	Yes	No
SQL Server 2005 integration	No	No	Yes	Yes
SQL Server 2005 Editions	SQL Server 2005 Express Edition	SQL Server 2005 Express Edition	SQL Server 2005 Developer Edition	SQL Server 2005 Developer Edition

Introduction to the IDE (Visual C#)

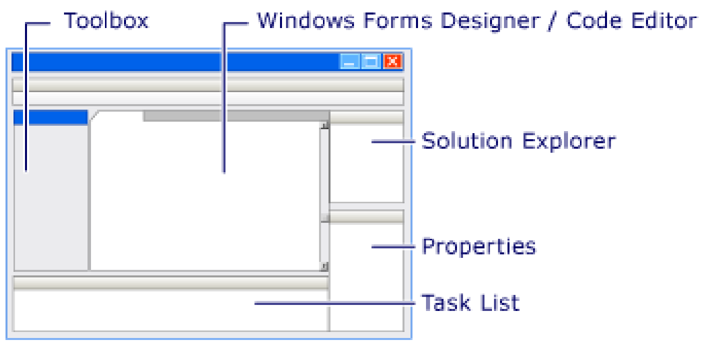
The Visual C# integrated development environment (IDE) is a collection of development tools exposed through a common user interface. Some of the tools are shared with other Visual Studio languages, and some, such as the C# compiler, are unique to Visual C#. The documentation in this section provides an overview of how to use the most important Visual C# tools as you work in the IDE in various phases of the development process.

The following are the most important tools and windows in Visual C#. The windows for most of these tools can be opened from the **View** menu.

- The Code Editor, for writing source code.
- The C# compiler, for converting C# source code into an executable program.
- The Visual Studio debugger, for testing your program.
- The **Toolbox** and **Designer**, for rapid development of user interfaces using the mouse.
- **Solution Explorer**, for viewing and managing project files and settings.
- **Project Designer**, for configuring compiler options, deployment paths, resources, and more.
- **Class View**, for navigating through source code according to types, not files.
- **Properties Window**, for configuring properties and events on controls in your user interface.
- Object Browser, for viewing the methods and classes available in dynamic link libraries including .NET Framework assemblies and COM objects.
- Document Explorer, for browsing and searching product documentation on your local machine and on the Internet.

How the IDE Exposes the Tools

You interact with the tools through windows, menus, property pages, and wizards in the IDE. The basic IDE looks something like this:



You can quickly access any open tool windows or files by pressing CTRL + TAB.

Editor and Windows Form Designer Windows

The large main window is used by both the Code Editor and the Windows Forms Designer. You can toggle between code view and Design view by pressing F7, or clicking **Code** or **Designer** on the **View** menu. While in Design view, you can drag controls onto the window from the **Toolbox**, which you can make visible by clicking on the **Toolbox** tab on the left margin.

The **Properties** window in the lower right is populated only in Design view. It enables you to set properties and hook up events for user interface controls such as buttons, text boxes, and so on. When you set this window to **Auto Hide**, it will collapse into the right margin whenever you switch to **Code View**.

Solution Explorer and Project Designer

The window in the top right is **Solution Explorer**, which shows all the files in your project in a hierarchical tree view. When you use the **Project** menu to add new files to your project, you will see them reflected in **Solution Explorer**. In addition to files, **Solution Explorer** also displays your project settings, and references to external libraries required by your application.

The **Project Designer** property pages are accessed by right-clicking on the **Properties** node in **Solution Explorer**, and then clicking **Open**. Use these pages to modify build options, security requirements, deployment details, and many other project properties

Compiler, Debugger, and Error List Windows

The C# compiler has no window because it is not an interactive tool, but you can set compiler options in the **Project Designer**. When you click **Build** on the **Build** menu, the C# compiler is invoked by the IDE. If the build is successful, the status pane displays a Build Succeeded message. If there were build errors, the **Error List** window appears below the editor/designer window with a list of errors. Double-click an error to go to the problem line in your source code. Press F1 to see Help documentation for the highlighted error.

The debugger has various windows that display values of variables and type information as your application is running. You can use the Code Editor window while debugging to specify a line at which to pause execution in the debugger, and to step through code one line at a time.

Customizing the IDE

All of the windows in Visual C# can be made dockable or floating, hidden or visible, or can be moved to new locations. To change the behavior of a window, click the down arrow or push-pin icons on the title bar and select from among the available options. To move a docked window to a new docked location, drag the title bar until the window dropper icons appear. While holding down the left mouse button, move the mouse pointer over the icon at the new location. Position the pointer over the left, right, top or bottom icons to dock the window on the specified side. Position the pointer over the middle icon to make the window a tabbed window. As you position the pointer, a blue semi-transparent rectangle appears, which indicates where the window will be docked in the new location.

Following are different templates under Project Types and their use.

Windows Application:

This template allows to create standard windows based applications. Windows Applications are form based standard Windows desktop applications for common day to day tasks. (Ex: Microsoft word).

Class Library:

Class libraries are those that provide functionality similar to Active X and DLL by creating classes that access other applications. Class library contains components and libraries to be used inside other applications. A Class library can not be executed and thus it does not have any entry point.

Windows Control Library:

This allows to create our own windows controls. Also called as User Controls, where you group some controls, add it to the toolbox and make it available to other projects. Windows Control Library contains user defined windows controls to be used by Windows applications.

Web Application:

This allows to create web-based applications using IIS. We can create web pages, rich web applications and web services. Web applications are programs that used to run inside some web server (Ex:IIS) to fulfill the user requests over the http. (Ex: Hotmail and Google).

Web Service:

Allows to create XML Web Services. Web services are web applications that provide services to other applications over the internet.

Web Control Library:

Allows to create User-defined controls for the Web. Similar to user defined windows controls but these are used for Web. Web Control Library contains user defined web controls to be used by web applications.

Console Application:

A new kind of application in Visual Studio .NET. They are command line based applications. Console applications are light weight programs run inside the command prompt (DOS) window. They are commonly used for test applications.

Windows Service:

These run continuously regardless of the user interaction. They are designed for special purpose and once written, will keep running and come to an end only when the system is shut down.

Other:

This template is to develop other kinds of applications like enterprise applications, database applications etc.

❖ .NET Framework & Architecture

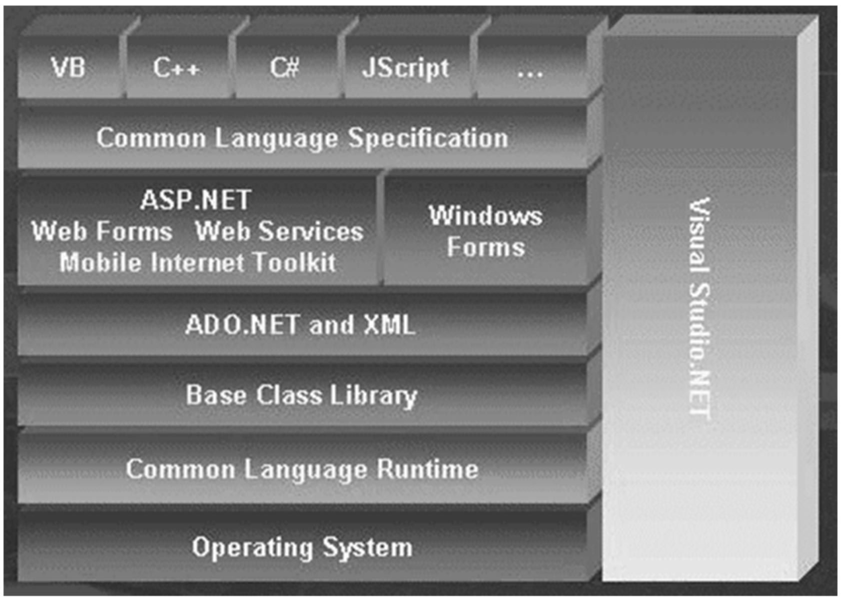
Microsoft .NET supports not only language independence, but also language integration. This means that you can take the advantage of classes, inheritance of classes etc. across different languages. The **.NET Framework** makes this possible with a specification called the **Common Type System (CTS)** that all .NET components must follow. The term .Net Framework refers to the group of technologies that form the development environment for the Microsoft .Net platform.

It also includes a **Common Language Specification (CLS)**, which provides a series of basic rules that are required for language integration. The CLS determines the minimum requirements for being a .NET language. Compilers that conform to the CLS create objects that can interoperate with one another. The entire Framework Class Library (FCL) can be used by any language that conforms to the CLS.

The .NET Framework sits on top of the operating system, which can be any flavor of Windows from Win 98 forward, and consists of a number of components.

The .NET Framework consists of:

- Different languages: VB.NET, C#, J#, and JScript .NET
- The **Common Language Runtime**, an object-oriented platform for Windows and web development that all these languages share
- A number of related class libraries, collectively known as the Framework Class Library (FCL) (BCL).



The most important component of the .NET Framework is the CLR, which provides the environment in which programs are executed. The CLR includes a virtual machine. At a high level, the CLR activates objects, performs security checks on them, store them out in memory, executes them, and garbage-collects them. The Common Type System is also part of the CLR.

The layer on top of the CLR is a set of framework base classes (BCL), followed by an additional layer of data and XML classes, plus another layer of classes intended for web services, Web Forms, and Windows Forms. Collectively, these classes are known as the Framework Class

Library, one of the largest class libraries in history and one that provides an object-oriented API to all the functionality that the .NET platform encapsulates. With more than 4,000 classes, the FCL facilitates rapid development of desktop, client/server, and other web services and applications.

Web Forms and Windows Forms allow you to apply Rapid Application Development techniques to building web and Windows applications. Simply drag and drop controls onto your form, double-click a control, and write the code to respond to the associated event.

❖ Features of .NET Platform

The core component of the .NET platform is found in the Common Language Runtime, Base Class Library & the Common Language Specification. The .NET BCL expose the features of the CLR in much the same way that the Windows API allows you to utilize the features of the Windows operating system; however, it also provides many higher-level features that facilitate code reuse.

This architecture gives a great number of benefits, not the least of which is a consistent API. By writing to the CLR & using the .NET BCL, all application services are available via a common object oriented programming model.

The new programming model greatly simplifies the efforts that were required when writing Windows DNA applications or fir that matter, almost Win32 & OM project.

Another great benefit for .NET benefit for .NET developers is its model for error handling via exceptions.

In .NET all errors are reported via exceptions, which greatly simplify writing, reading, & maintaining code. Thanks to Common Language Specification & Common Type System, .NET exception work across module & language boundaries.

1) Multilanguage Development:

Different types of application and programmer required to work with different types of programming languages. With the use of CLR the component of .Net framework, it is easy to work with different types of languages in .net platform. In .Net platform programmers can choose the languages which best for them. Additionally, .Net framework allows the integration of these different languages through the use of MSIL. Currently .Net supports languages like VB.Net, C#.net, J#.Net, Jscript.Net, COBOL, Fortran, Perl, Python and many more. In .Net platform, we can develop different types of application like windows, web or portable application which runs on PDA or mobile device.

2) Platform and Processor Independence

When we compile any application in .Net platform first it converted into MSIL code. This MSIL code is CPU-Independent and is higher then any machine language. After creation of MSIL code, this code can be run on any platform and processor which supports Common Language Runtime. So, programmers can develop application which can be run on any platform and processor.

3) Automatic Memory Management:

The programmers always worry about the Memory while it is developing any type of application. We already use, program like C and C++, we are allocating memory to particular object using different function and we release this memory when program is terminated. In .Net environment, Microsoft is provide facility to make developing easier. It provide a component called Garbage Collection which handles this memory management task. When any object required memory it automatically allocates it and when that is no longer valid then it automatically free up those memory to resue.

4) Easy Deploying:

After developing any type of application, programmers always worry about how they can deploy (install) their application. Most of the companies use third party software to build their installation. These installation contain a large number of files installed in different directory, various registry setting required, COM componets and short cuts may required. The .Net application doesn't require any extra activity than copying their files to a directory. For uninstalling an application will be easy as deleting those files. This is because .Net components are not referenced in registry because of Metadata.

5) Distributed Architecture:

Today, application or websites which are presented to user come from different sources like servers located at different places and different application running on this server which fetches data from many database. This called distributed architecture. This type of application are very complex to build and maintain. Each interface uses different types of programming concept to fetch data. The .Net provide architecture for developing this type of complex application using different concept like XML, SOAP, UDDI.

6) Interoperability with Unmanaged Code:

The code which is generated by CLR are is known as managed code. So, code outside from CLR and not managed by CLR is unmanaged code. However, this code is still run by the CLR but we cant take the advantage like CTS and automatic memory management. There are many situation arrive, when you have to work with code that is outside from .Net platform. Companies doesn't deliver a .Net component version of their products every time. So, Microsoft add a functionality in CLR so it can work with this unmanaged code. Examples of unmanaged code are calling DLL files, Calling COM components.

7) Security:

Distributed component application require security. The .Net designer follow the approach which provides separation and access control based on user account. Security for .Net applications starts as soon as a class is loaded by the CLR. Before the class is loaded, the accessibility rules and requirement are checked. Additionally, when code request, access to the certain resources, the class identification are verified.

8) Performance and Scalability:

There is no magic tool that will allow a poorly designed application to scale and perform well. The .Net framework gives you a tools that make it easier to design better performing software.

❖ Components of .NET Architecture

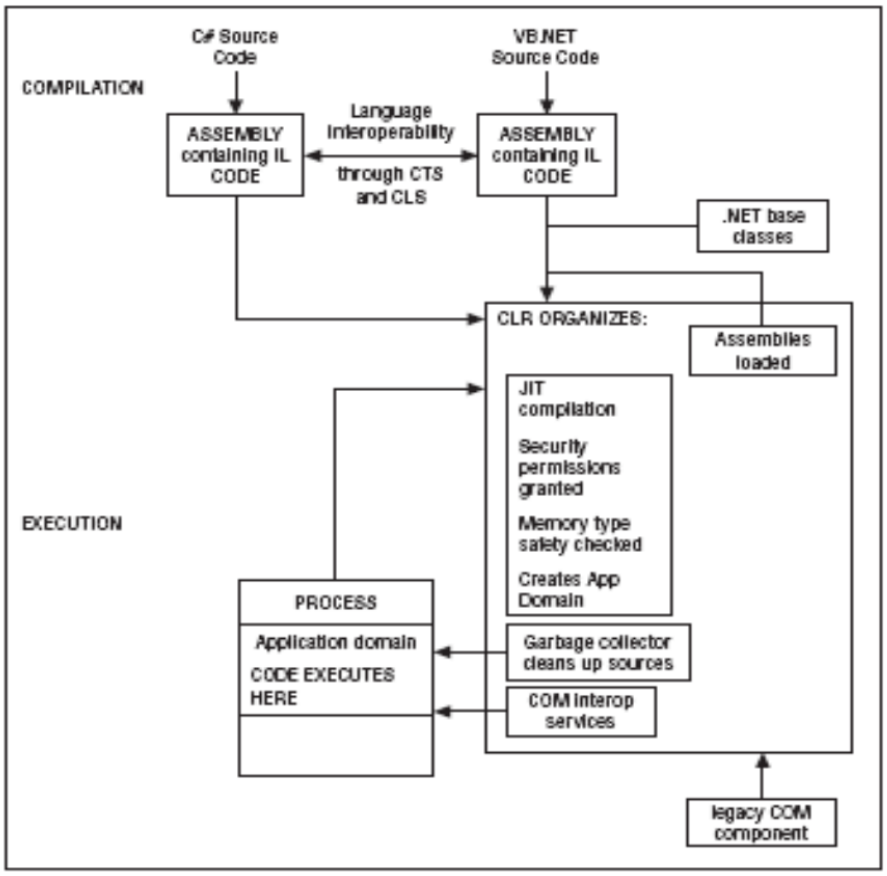
Following are the different components of .Net Architecture. Each explain in detail with their functionality.

1) .Net Runtime (Common Language Runtime):

The heart of the .Net framework is the CLR. This is similar to the Java Virtual Machine. It is an environment that executes MSIL code. In java, the JVM is the concept of one language for all purpose while .Net platform supports multiple programming languages through the use of Common Language Specification. It is also referred to as a managed environment, one in which common services, such as garbage collection and security, are automatically provided. Following are the different feature provided by

Process of compilation and executing:

Code that we have develop with language compiler and run under the CLR is called managed code. It provides the functionality like cross-language integration, exception handling, security. To provide services to managed code, language compiler are required to add metadata with it. It means they provide information that describes the types, members and reference used in code. Metadata is stored with code. When class loader loads the code, it uses the metadata to locate and load classes, allocate required memory, solve method invocation, generate native code, apply security and set up run time boundaries for application.



2) Managed and Unmanaged Code:

Managed Code is what Visual Basic .NET and C# compilers create. It compiles to Intermediate Language (IL), not to machine code that could run directly on your computer. The IL is kept in a file called an assembly, along with metadata that describes the classes, methods, and attributes (such as security requirements) of the code you've created. This assembly is the one-stop-shopping unit of deployment in the .NET world. You copy it to another server to deploy the assembly there—and often that copying is the only step required in the deployment.

Managed code runs in the Common Language Runtime. The runtime offers a wide variety of services to your running code. In the usual course of events, it first loads and verifies the assembly to make sure the IL is okay. Then, just in time, as methods are called, the runtime arranges for them to be compiled to machine code suitable for the machine the assembly is running on, and caches this machine code to be used the next time the method is called. As the assembly runs, the runtime continues to provide services such as security, memory management, threading, and the like.

Unmanaged code is what you use to make before Visual Studio .NET was released. Visual Basic 6, Visual C++ 6, heck, even that 15-year old C compiler you may still have kicking around on your hard drive all produced unmanaged code. It compiled directly to machine code that run on the machine where you compiled it and on other machines as long as they had the same chip, or nearly the same.

It didn't get services such as security or memory management from an invisible runtime; it got them from the operating system. And importantly, it got them from the operating system explicitly, by asking for them, usually by calling an API provided in the Windows

3) Intermediate Language:

MSIL is the CPU-independent instruction set into which .Net framework programs are compiled. It contains instructions for loading, storing, initializing and calling method of different objects. MSIL contains code and Metadata which is true cross language integration. It is also known as CIL (common intermediate language) or IL (intermediate language). We can directly code into MSIL language but it is rare case.

4) Common Type System:

As Microsoft .Net provide application development using different programming languages. For this, Microsoft has provided Common Type System which means we don’t have to worry when we are developing multiple languages about how a data types declared in one language needs to be declared in another. Any .Net type has the same attributes regardless of the language it is used in. In .Net all data types are objects which derived from System. Object. So, all data types derive from a common base class, they all share some basic functionality.

5) Base Class Library:

In C and C++, we include header files like stdio.h, conio.h for using library functions. In .NET, BCL, is the collection of classes and namespaces which we use in application for variety of inbuilt functionality. The .NET framework Base Class Library provides an extensive collection of classes which are hierarchically organized via namespaces. This library consists of classes related to Data, XML, Web Forms, Windows Form, Smart Device, Input Output etc.

The namespace is logical container or partition which group the different classes related to same functionality. It looks like drive’s or folder in our computer. We organized our data into different folders or drives. For specific information or files we go into the specific folder or drive for faster retrieval of information.
The root of the hierarchy of Base Class Library is System namespace. Here, two classes can have same name but they must reside in different namespace. Following are the listing of some common namespace with its description.

Namespace	Contains
System	Fundamental and base classes that defines commonly used value and references data types, events, interface, attributes
System.Data	Classes related to ADO.NET which useful for working with database
System.Collections	Classes used for various objects like lists, queue, array etc.

System.Drawing	Classes related to GUI drawings
System.IO	Classes related to reading and writing data streams and files
System.Web.UI	Classes related to create controls and pages that will appear on Web applications as user interface on a web page.
System.Windows.Form	Classes for creating Windows-based applications that take full advantage of rich user interface
System.XML	Provide standards-based support for processing XML.

6) Assemblies:

Assemblies are the building blocks of .NET Framework applications; they form the fundamental unit of deployment, version control, reuse, activation scoping, and security permissions. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An assembly provides the common language runtime with the information it needs to be aware of type implementations.

It contains code that the common language runtime executes. Microsoft intermediate language (MSIL) code in a portable executable (PE) file will not be executed if it does not have an associated assembly metadata.

Assemblies can be static or dynamic. Static assemblies can include .NET Framework types (interfaces and classes), as well as resources for the assembly (bitmaps, JPEG files, resource files, and so on). Static assemblies are stored on disk in portable executable (PE) files. You can also use the .NET Framework to create dynamic assemblies, which are run directly from memory and are not saved to disk before execution. A static assembly can consist of four elements: Assembly metadata, Type metadata, Microsoft intermediate language (MSIL) code that implements the types, A set of resources.
A private assembly is used by a single application and is store in that application's install directory. A shared assembly is one that can be referenced by more than one application.

7) Metadata:

The data about data is called Metadata. It is a feature that lets the CLR know the details about a particular component or object. The metadata for an object is persisted at compiled time and then queried at runtime so that the CLR know how to initialize object, call their methods and access their properties. An application can interrogate metadata and learn what an object exposes. This process is known as **reflection.** This data is stored in component itself in a binary format inside an assembly. It contains a declaration for every type including names and its members like methods, fields, properties and event. When a class loader of CLR loads an assembly at that time it uses a metadata to locate the body of method.

8) Assembly Cache:

The assembly cache is a directory which contains the different assembly on the machine. There are two types of assembly cache: A global assembly cache and a transient (temporary) assembly cache.

When assemblies are downloaded to the local machine using browser, it is automatically installed in the transient assembly cache. Each computer where the common language runtime is installed has a machine-wide code cache called the global assembly cache. The global assembly cache stores assemblies specifically designated to be shared by several applications on the computer.

You should share assemblies by installing them into the global assembly cache only when you need to. As a general guideline, keep assembly dependencies private, and locate assemblies in the application directory unless sharing an assembly is explicitly required.

9) Reflection:

Reflection is the process by which .NET applications can access an assembly’s metadata information and discover its methods and data types at runtime. We can also dynamically invoke methods and use type information through late binding through Reflection API. System.type class is the core of the reflection. It is used to represent a Common Type System which includes methods that allow to determined type’s name, which module it contain and its namespace with it contains value or reference type.

Using System.Reflection.Assembly class you can retrieve all of the types in an assembly and all modules contained in the assembly. To invoke any method of class named Activator class to create instance and then GetMethod method to invoke the method.

10) Just-In-Time Compilation (JIT):

When any .NET application compile, it is not converted into machine code but it converted into MSIL (Intermediate Code). This code is machine independent. So, CLR provide Just In Time compilation technology to convert the MSIL code into a platform/device-specific code so that it can be executed.

The .NET provide three types of JIT compilers:

Pre-JIT: This JIT compiles an assembly’s entire code into native code at one cycle. Normally, it is used at installation time.

Econo-JIT: This compiler is used on devices with limited resources. It compiles the IL code bit-by-bit freeing resources used by the cached native code when required.

Normal-JIT: The default JIT compiles code only as it is called and places the resulting native code in the cache.

When, JIT compiles, the native code is placed into the cache, so that when the next call is made to the same method, the cached code is executed. So it increase the application speed.

11) Garbage Collection:

Memory management is one of the housekeeping duty that takes a lot of programming time in developing application. We doesn't like spent a time for programming related to memory. So .NET provides a environment with the garbage collection system. Garbage collection runs when application needs free memory. There is no exact time of execution of garbage collection system.

When application required more memory and the memory allocator reports that there is no free memory than garbage collection is called. It start by assuming that everything should be deleted from the memory. First it create a graph of used memory by application. When it has complete graph of memory used by application then it copies this data and free up the whole memory. And reallocate the memory to the application.

Garbage collector also free up the memory for unused object at regular interval. We do not have to write code to perform memory management task when developing managed applications

❖ Explain CTS, CLR and JIT

CLR (Common Language Runtime):- The most important concept of the .net framework is the existence and functionality of the .net common language runtime (CLR), also called .net runtime and short. It is a framework layer that resides above the OS and handles the execution of all the .net applications. Our programs don't directly communicate with the OS but go through the CLR.

CTS (Common Type System):- .NET also defines a common type system (CTS). Like CLS, CTS is also a set of standards. CTS defines the basic data types that IL understands. Each .Net compliant language should map its data types to these standard data types. This makes it possible for the 2 languages to communicate with each other by passing/receiving parameters to and from each other. For example, CTS defines a type, int32, an integral data type of 32 bits(4 bytes) which is mapped by C# through int and VB.net through its integer data type.

JIT (Just In Time Compilers):- When out IL compiled code needs to be executed, the CLR invokes the JIT compiler, which compile the IL code to native executable code(.exe of .dll) that is designed for specific machine and OS. JITers in many ways are different from traditional compliers as they compile the IL to native code JIT. So, the part of code that is not used by that particular run is never converted to native code. If some IL code is converted to native code, then the next time it's needed, the CLR reuses the same (already compiled) copy without re-compiling. So, if a program runs for some time (assuming that all or most of the functions get called). Then it won't have any just-in-time performance penalty.

As JITers are aware of the specific processor and OS at runtime and OS at runtime. They can optimize the code extremely 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 (like replacing body of small function when its called in a loop, saving the function call time). Although Microsoft stated that C# and .net are not competing with language like C++ in efficiency and speed of execution. JITers can makes your code even faster than C++ code in some cases when the program in run over an extended period of time.

❖ Managed Vs. Unmanaged Code

Manage code: -

The execution of manage code is done by following steps.

1. Selecting language complier.
2. Completing the code to IL (information language).
3. Completing IL to native code.
4. Executing code.

The CLR is used by selecting one or more language complier such as VB, C++, VC++, java, JavaScript etc. The language complier will decide the syntax of the code. When the program is compile then that code is called manage code. The complier will convert the source code to IL which is dependent on CPU. Just in time complier converts IL into native or CPU specific code. When you compile source code to IL required metadata is generated.

Unmanaged Code:-

The unmanaged code directly complies with the mechanic code and executed when a work it is complete. It does not have services such as security and memory management but the OS is gives this services at runtime. The OS gives these services by calling the API of the window.

❖ Explain Assemblies and Metadata

Meta data describes source code or program which is in binary information stored in CLR portable executable file or in the memory. When the compilation when the code take place in DE file then metadata in inserted in the file. The metadata describes data types and members used in the program. When the code is executed the CLR loads the metadata into the memory and finds information about classes and its members.

Meta Data Contains:-

Assembly information such as name, version, type of assembly, reference assembly and security permission. Information about type such as name, interfaces used, methods, fields, properties, events based class etc. It maintains attributes information which are modified by the user.

Assemblies:-

The assembly contains code which is executed by CLR (Common Language Runtime). There are two types of assembly.

1. Dynamic:- Dynamic assembly will run directly from the memory without saving but it can be saved after execution.

2. Static:- Static assembly includes interfaces classes and resources. This assembly are stored in PE files.

Assembly Contains:-

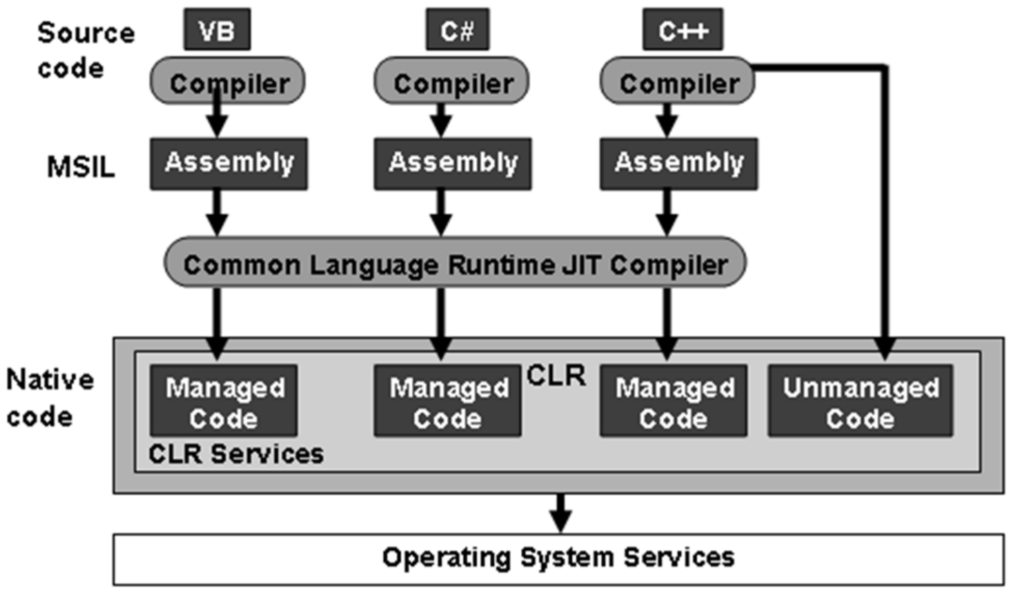
Assembly in a logical unit which consist of four parts.

1. Manifest:-
2. Type Metadata

Every assembly, whether static or dynamic, contains a collection of data that describes how the elements in the assembly relate to each other. The assembly manifest contains this assembly metadata. An assembly manifest contains all the metadata needed to specify the assembly's version requirements and security identity, and all metadata needed to define the scope of the assembly and resolve references to resources and classes. The assembly manifest can be stored in either a PE file (an .exe or .dll) with Microsoft intermediate language (MSIL) code or in a standalone PE file that contains only assembly manifest information. It contains Assembly name, Version number, Culture, Strong name information, List of all files in the assembly, Type reference information, Information on referenced assemblies.

Type Metadata is information stored in the assembly that describes the types and methods of the assembly and provides other useful information about the assembly. Assemblies are said to be self-describing because the metadata fully describes the contents of each module

❖ Managed execution process of .NET applications.



The process of executing .NET application is divided into different steps. First steps is select a compiler based on language which you have written source code. There are different compiler available in .NET framework like for VB.NET select VBC, for C# select CSC etc.

In, second step the source code is converted into MSIL code which contains assembly and metadata information.