Chapter 2

Introducing the .NET Framework

The .NET Framework enables you to create robust and scalable applications. It consists of common language runtime (CLR), Common Language Specification (CLS), and the just-in-time (JIT) compiler.

Before you can use Visual Studio .NET for creating a Windows application, you need to understand the .NET Framework and the Visual Studio .NET integrated development environment (IDE).

This chapter introduces the features and components of the .NET Framework. In addition, this chapter explains the process of creating and executing a Windows application by using the Visual Studio .NET IDE.

Objectives

In this chapter, you will learn to:

- Identify the features of .NET Framework
- Use Visual Studio .NET integrated development environment to create and execute Windows projects

The .NET Framework

Microsoft introduced the .NET Framework with the intention of enhancing the interoperability of applications. This framework aims at integrating various programming languages and services. It is designed to make significant improvements in code reuse, code specialization, resource management, multilanguage development, security, deployment, and administration. It consists of all the technologies that help in creating and running robust, scalable, and distributed applications. .NET offers a complete suite for developing and deploying applications. This suite consists of .NET Products, .NET Services, and the .NET Framework.

- **.NET Products**: Microsoft has already introduced Visual Studio .NET, which is a tool for developing .NET applications, by using programming languages such as Visual Basic, Visual C#, and Visual C++.

  These products allow developers to create applications that are capable of interacting seamlessly with each other. To ensure interaction between various applications, all .NET products use Extensible Markup Language (XML) for describing and exchanging data between themselves.

Note

XML is a platform independent markup language. It allows computers to store data in a format that can be interpreted by any other computer using any platform. Therefore, XML can be used to transfer structured data between heterogeneous systems. XML is used as a common data interchange format in a number of applications.

- **.NET Services**: .NET helps you to create software such as Web services. A *Web service* is an application or business logic that is accessible through standard Internet protocols such as Hypertext Transfer Protocol (HTTP) and Simple Object Access Protocol (SOAP). You can identify the service by a Uniform Resource Locator (URL). The public interfaces and bindings of the Web service are described by using XML. Therefore, users can subscribe to such a service and use it as long as they need it, regardless of their hardware and software platforms.

  Microsoft has come up with its own set of Web services, known as My Services. These services are based on the Microsoft Passport Authentication service, which is used in Web applications such as Hotmail. This service allows users to access data by linking calendars, phonebooks, address books, and personal references to the passport authentication service.

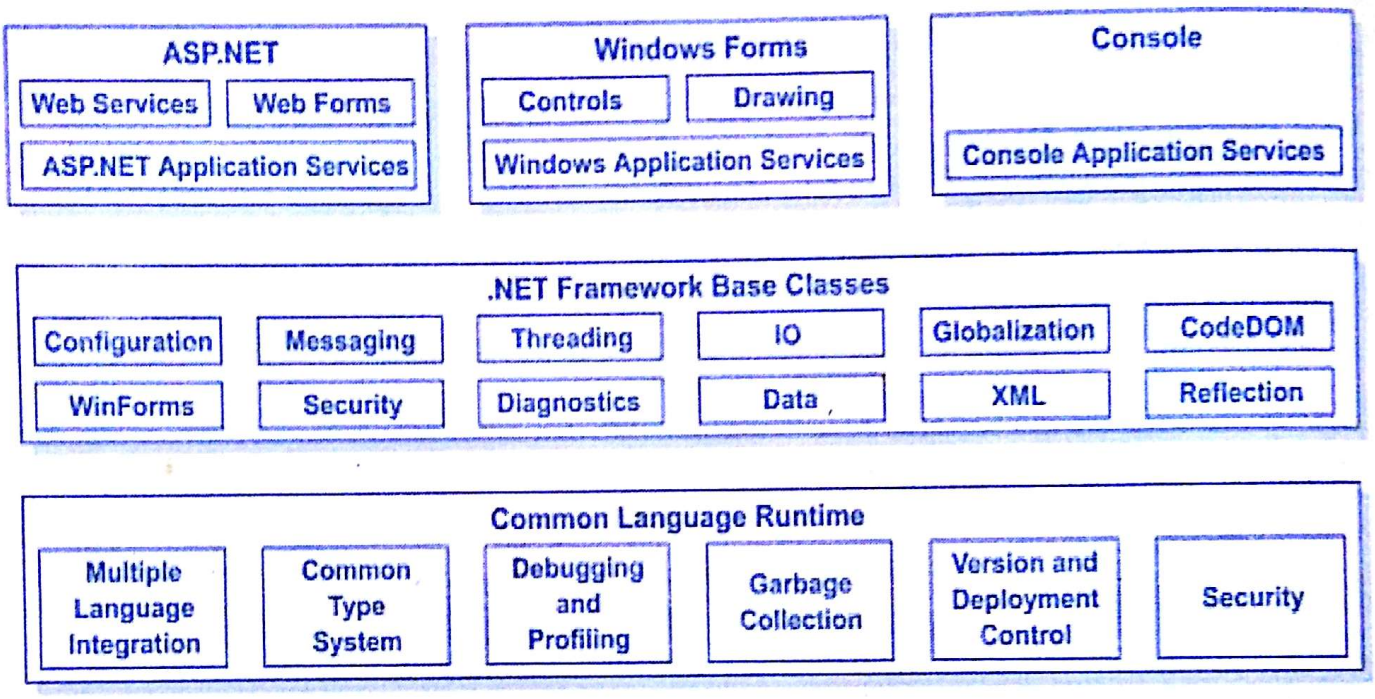
  In addition, third party products and services can be integrated easily with the .NET environment.

- **.NET Framework**: It is the foundation on which you design, develop, and deploy applications. It is a consistent and simplified programming model that helps you to easily build robust applications. It is the core of the .NET infrastructure because it

exists as a layer between .NET applications and the underlying operating system. In other words, the .NET Framework encapsulates most of the basic functionality, such as debugging and security services, which were earlier built into various programming languages in the form of a collection of services and classes.

Components of the .NET Framework

The following figure shows the different components of the .NET Framework.



Components of the .NET Framework

The .NET Framework consists of three main components: the Common Language Runtime (CLR), the .NET Framework Base Classes, and the user and program interfaces.

Common Language Runtime (CLR)

The CLR is one of the essential components of the .NET framework. CLR is the environment in which all programs that use .NET technologies are executed. It provides services such as code compilation, memory allocation, and garbage collection. The CLR allows the execution of code across different platforms by translating code into Intermediate Language (IL). IL is a low-level language that the CLR understands.

IL is converted into machine language during execution by the JIT compiler. During JIT compilation, code is also checked for type safety. Type safety ensures that objects are always accessed in a compatible way. If you try to assign an 8-byte value to a variable of size 4 bytes, the CLR will detect and trap it.

CLR consists of a set of common rules followed by all the languages of the .NET framework. This set of rules is known as CLS. CLS enables an object or application to interact with objects or applications of other languages. The classes that follow the rules specified by CLS are termed as CLS-compliant classes. The classes defined in the .NET Framework class library are CLS-compliant.

One of the specifications defined in CLS is Common Type System (CTS), which provides a type system that is common across all languages. A type system defines how a programming language classifies values and expressions into types, how it can manipulate those types and how they interact. CTS defines how data types are declared, used, and managed in the code during run time.

The CTS also defines the rules that ensure that the data types of objects written in various languages are able to interact with each other. For example, the size of integer and long variables is the same across all CLS-compliant programming languages.

The source code needs to be compiled before execution. CLR plays an important role in the process of compilation and execution of the program.

Identifying the Process of Compilation

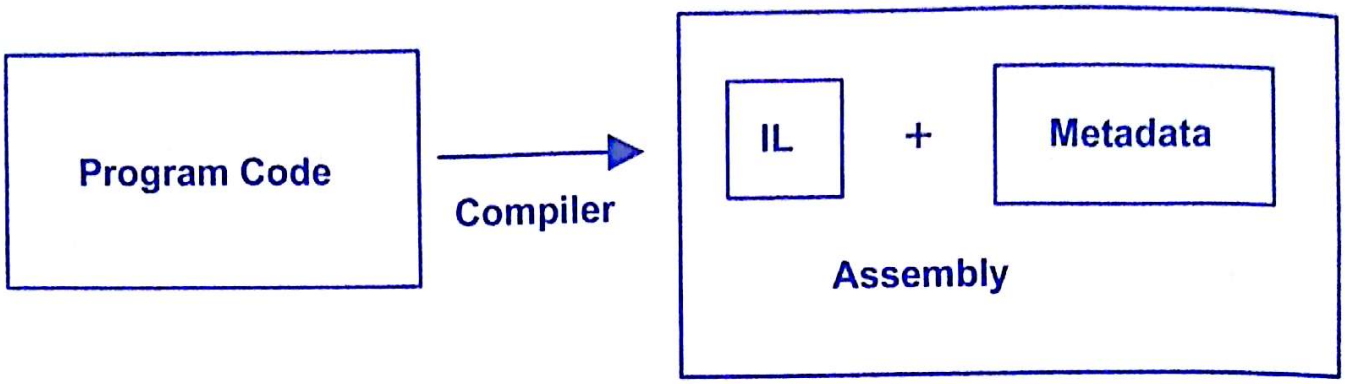
Compilation is the process of creating an executable program from the source code. The source code consists of instructions for the compiler, and an executable program consists of instructions for the processor. Therefore, compilation converts source code into machine language.

However, when you compile a program in .NET, the conversion of source code to machine language happens in two stages. In the first stage, the compiler translates code into an IL instead of machine language or assembly language. In the second stage, the JIT compiler converts IL into machine language at run time.

Irrespective of the CLS-compliant language that is used to develop the application, the source code always gets translated into IL. In addition, during compilation, the compiler also produces metadata of the code. Metadata contains the description of code such as classes and interfaces, dependencies, and versions of the components used in the application. IL and metadata constitute an assembly.

Assemblies also contain metadata, which describe the assembly's internal version number and details of all the data and object types they contain. When you compile a VC# application, Visual Studio .NET creates an assembly, which is a single file with the extension .exe or .dll.

The following figure shows the process of code compilation.



Code Compilation

Identifying the Process of Code Execution

During execution, CLR performs the following steps:

- **Loading assemblies and identifying namespaces**: Assemblies are loaded into the memory. After loading assemblies, CLR identifies namespaces for code in assemblies. Namespaces are a collection of classes. The .NET Framework uses namespaces to organize its classes in a hierarchy. Namespaces implicitly have public access and this cannot be changed.
- **JIT compilation**: Before execution, IL is converted into machine language by the JIT compiler. Next, during the verification process, the IL code is examined to confirm the following points:
  - The memory locations that code needs to access are available.
  - Methods are called only through properly defined types.
  - IL has been correctly generated.
- **Garbage collection**: The garbage collection process begins after JIT compilation and manages the allocation and deallocation of memory for an application. Whenever you create an object, the CLR allocates memory for the object from the managed heap. A managed heap is a region of the memory that is available for program execution. If sufficient memory is not available on the *managed heap*, the garbage collection process is invoked.

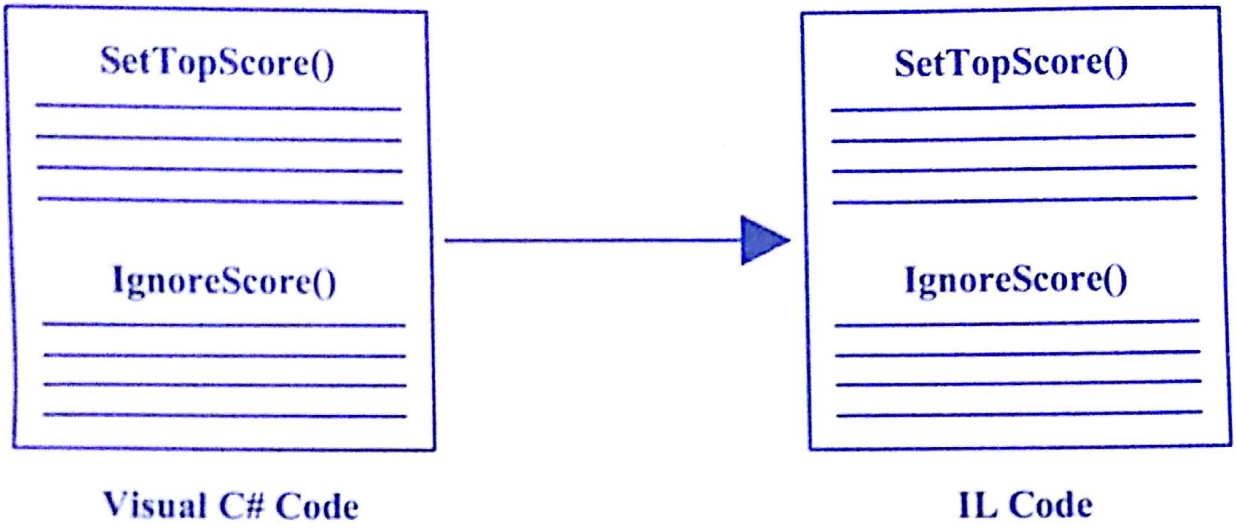
The code developed in .NET is called managed code. The CLR manages the compilation and execution of the managed code to ensure its proper functioning. For example, the CLR takes care of garbage collection, exception handling, and type safety for the managed code.

The unit of execution in the CLR is an assembly. An assembly contains IL and metadata that were generated during compilation. It contains code that the CLR executes. All assemblies contain a manifest, which contains information such as assembly name,

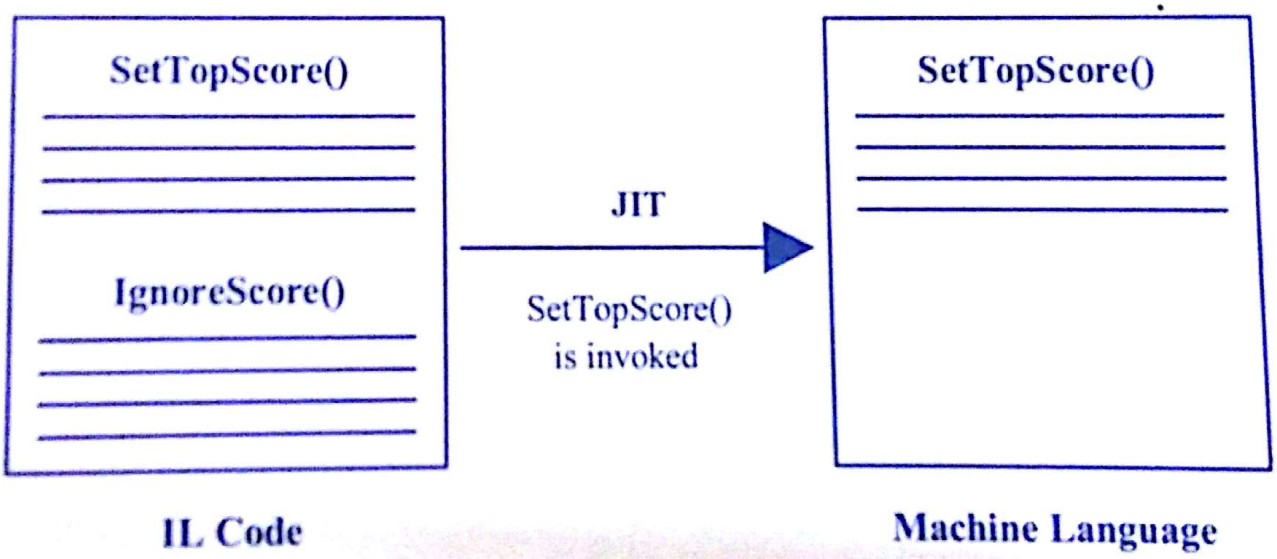
version, and the list of files that form the assembly. The IL code cannot be executed if it does not have an associated assembly.

Instead of compiling the complete IL code, the JIT compiler compiles only the code that is required during execution. This saves time and memory required to convert the complete IL into machine language.

The following figure shows the process of code compilation and execution.



Code Compilation



Code Execution

In the preceding figure, two methods are shown, `SetTopScore()` and `IgnoreScore()`. These methods should be invoked based on the following conditions.

The `SetTopScore()` method will be invoked only if the current score of the player is higher than the top score of the game. The `IgnoreScore()` method will be invoked when the player is not the top scorer and you do not want to add the player to the list of top scorers.

If the player has scored more than the top score, only `SetTopScore()` will be invoked. When it is invoked, the code for the `SetTopScore()` method will be compiled by the JIT compiler. However, the code for `IgnoreScore()` will not be converted to machine language by the JIT compiler because this method is not invoked.

The .NET Framework Class Library

The .NET Framework class library works with any .NET language, such as VB.NET, VC++ .NET, and VC#. This class library is built on the object-oriented nature of the runtime. The library provides classes that can be used in the code to accomplish a range of common programming tasks, such as string management, data collection, database connectivity, and file access.

One of the most important features of the .NET Framework class library is that it can be used in a consistent manner across multiple languages. This means that you can use the same set of classes for performing a specific task in VC# as well as in VC++.

The .NET Framework class library consists of namespaces, which are contained within assemblies. Let us look at what these two terms mean.

Namespaces

Namespaces help you to create logical groups of related classes and interfaces, which can be used by any language targeting the .NET Framework. Namespaces allow you to organize your classes so that they can be easily accessed in other applications. Namespaces can be used to avoid any naming conflicts between classes. For example, you can use two classes with the same name in an application provided they belong to different namespaces.

You can access the classes belonging to a namespace by simply importing the namespace into the application. The .NET Framework uses the dot (.) as a delimiter between classes and namespaces. For example, `System.Console` represents the `Console` class of the `System` namespace. Namespaces are also stored in assemblies.

Assemblies

An assembly is a single deployable unit that contains all the information about the implementation of classes, structures, and interfaces. The assembly stores all the information about itself. This information is called metadata and includes the name and version of the assembly, security information, information about dependencies, and a list of the files that constitute the assembly.

All the applications developed by using the .NET Framework are made up of assemblies. Assemblies and the metadata provide the CLR with the information required for executing

an application. For example, if an application uses a component, the assembly keeps track of the version number of the component used in the application. The assembly provides this information to the CLR while the application is being executed. Assemblies also play an important role in deployment and versioning.

User and Program Interfaces

At the presentation layer, .NET provides three types of user interfaces. They are Windows Forms, Web Forms, and Console Applications. Windows Forms are used in Windows-based applications, whereas Web Forms are used in Web-based applications for providing an interactive user interface. They provide a Web-browser based user interface. You can also create character-based console applications that can be executed from the command prompt.

Advantages of the .NET Framework

Some of the advantages of the .NET Framework are:

- **Consistent programming model**: The .NET Framework provides a common object-oriented programming model across languages. This object model can be used to perform several tasks, such as reading from and writing to files, connecting to databases, and retrieving data.
- **Multi-platform applications**: There are several versions of Windows most of which run on x86 CPUs. Some versions, such as Windows CE and the 64-bit Windows, run on non-x86 CPUs as well. A .NET application can execute on any architecture that is supported by the CLR. In future, a CLR version could even be built for non-Windows platforms.
- **Multi-language integration**: .NET allows multiple languages to be integrated. For example, it is possible to create a class in VC# that is derived from a class implemented in VB.NET. To enable objects to interact with each other regardless of the language used to develop them, a set of language features has been defined in CLS.

  This specification includes the basic language features required by many applications. The CLS enhances language interoperability. The CLS also establishes certain requirements, which help you to determine whether your managed code conforms to the CLS. Most of the classes defined in the .NET Framework class library are CLS-compliant.
- **Automatic resource management**: While creating an application, a programmer may be required to write code for managing resources such as files, memory, network connections, and database resources. The CLR automatically tracks resource usage and relieves a programmer of the task of manual resource management.
- **Ease of deployment**: One of the goals of the .NET Framework is to simplify application deployment. .NET applications can be deployed simply by copying files

to the target computer. Deployment of components has also been simplified. Till now, Microsoft's Component Object Model (COM) had been used for creating components. However, COM suffers from various problems related to deployment. For example, every COM component needs to be registered before it can be used in an application.

Moreover, two versions of a component cannot run at the same time. In such a case, a new application that installs the newer version of the component may cause the existing applications that depend on the earlier version of the component to stop functioning. However, the .NET Framework provides zero-impact deployment. Installation of new applications or components does not have an adverse effect on the existing applications. Assemblies are used to handle problems related to versioning.