25

### [BCL](https://web.archive.org/web/20100816131838/http:/en.csharp-online.net/Glossary:Definition_-_Base_Class_Library):

A .NET Framework library, BCL is the standard for the C# runtime library and one of the Common Language Infrastructure (CLI) standard libraries. BCL provides types representing the built-in CLI data types, basic file access, collections, custom attributes, formatting, security attributes, I/O streams, string manipulation, and more.

### [FCL](https://web.archive.org/web/20091118143949/http:/en.csharp-online.net/Introducing_.NET%E2%80%94The_.NET_Framework_Class_Library):

The .NET Framework class library is exactly what its name suggests: a library of classes and other types that developers can use to make their lives easier. While these classes are themselves written in C#, they can be used from any CLRbased language

**BCL** stands for Base class library also known as Class library (CL). BCL is a subset of Framework class library (FCL). Class library is the collection of reusable types that are closely integrated with CLR. Base Class library provides classes and types that are helpful in performing day to day operation e.g. dealing with string and primitive types, database connection, IO operations.

while Framework class library contains thousands of classes used to build different types of applications and provides all the basic functionalities and services that application needs. FCL includes classes and services to support different variety of application e.g.

* Desktop application,
* Web application (ASP.Net, MVC, WCF),
* Mobile application,
* Xbox application,
* windows services etc.

The Base Class Library (BCL) is the core set of classes that serve as the basic API of the Common Language Runtime. The classes in mscorlib.dll and some of the classes in System.dll and System.core.dll are considered to be a part of the BCL. It includes the classes in namespaces like System , System.Diagnostics , System.Globalization, System.Resources , System.Text , System.Runtime.Serialization and System.Data etc.

The Framework Class Library (FCL) is a superset of the BCL classes and refers to the entire class library that ships with .NET Framework. It includes an expanded set of libraries, including Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation among others.

So there are differences and you must not use those interchangeably.

Framework Class Library (FCL)

.NET Framework provides huge set of Framework (or Base) Class Library (FCL) for common, usual tasks. FCL contains thousands of classes to provide the access to Windows API and common functions like String Manipulation, Common Data Structures, IO, Streams, Threads, Security, Network Programming, Windows Programming, Web Programming, Data Access, etc. It is simply the largest standard library ever shipped with any development environment or programming language. The best part of this library is they follow extremely efficient OO design (design patterns) making their access and use very simple and predictable. You can use the classes in FCL in your program just as you use any other class and can even apply inheritance and polymorphism on these.

Common Language Specification (CLS)

Earlier we used the term '.NET Compliant Language' and stated that all the .NET compliant languages can make use of CLR and FCL. But what makes a language '.NET compliant language'? The answer is Common Language Specification (CLS). Microsoft has released a small set of specification that each language should meet to qualify as a .NET Compliant Language. As IL is a very rich language, it is not necessary for a language to implement all the IL functionality, rather it meets the small subset of it, CLS, to qualify as a .NET compliant language, which is the reason why so many languages (procedural and OO) are now running under .Net umbrella. CLS basically addresses to language design issues and lays certain standards like there should be no global function declaration, no pointers, no multiple inheritance and things like that. The important point to note here is that if you keep your code within CLS boundary, your code is guaranteed to be usable in any other .Net language.

Common Type System (CTS)

.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/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.

Garbage Collector (GC)

CLR also contains Garbage Collector (GC) which runs in a low-priority thread and checks for un-referenced dynamically allocated memory space. If it finds some data that is no more referenced by any variable/reference, it re-claims it and returns the occupied memory back to the Operating System; so that it can be used by other programs as necessary. The presence of standard Garbage Collector frees the programmer from keeping track of dangling data.

## What is a .Net Assembly?

The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file. All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. The designers of .NET have worked a lot on the component (assembly) resolution.

There are two kind of assemblies in .NET;

1. private
2. shared

Private assemblies are simple and copied with each calling assemblies in the calling assemblies folder.

Shared assemblies (also called strong named assemblies) are copied to a single location (usually the Global assembly cache). For all calling assemblies within the same application, the same copy of the shared assembly is used from its original location. Hence, shared assemblies are not copied in the private folders of each calling assembly. Each shared assembly has a four part name including its face name, version, public key token and culture information. The public key token and version information makes it almost impossible for two different assemblies with the same name or for two similar assemblies with different version to mix with each other.

An assembly can be a single file or it may consist of the multiple files. In case of multi-file, there is one master module containing the manifest while other assemblies exist as non-manifest modules. A module in .NET is a sub part of a multi-file .NET assembly. Assembly is one of the most interesting and extremely useful areas of .NET architecture along with reflections and attributes, but unfortunately very few people take interest in learning such theoretical looking topics.

## Just In Time Compilers (JITers)

When our IL compiled code needs to be executed, CLR invokes JIT compilers which compile the IL code to native executable code (.exe or .dll) for the specific machine and OS. JITers in many ways are different from traditional compilers as they, as their name suggests, compile the IL to native code only when desired e.g., when a function is called, IL of function's body is converted to native code; just in time of need. So, the part of code that is not used by particular run is not converted to native code. If some IL code is converted to native code then the next time when its needed to be used, the CLR uses the same copy without re-compiling. So, if a program runs for sometime, then it won't have any just in time performance penalty. As JITers are aware of processor and OS exactly at runtime, they can optimize the code extremely efficiently resulting in very robust applications. Also, since JITer 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 languages like C++ in efficiency, speed of execution, JITers can make your code even faster than C++ code in some cases when program is run over extended period of time (like web-servers).

An enumeration is a set of named integer constants. An enumerated type is declared using the **enum** keyword.

C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.

## Declaring *enum* Variable

The general syntax for declaring an enumeration is −

enum <enum\_name> {

enumeration list

};

Where,

* The *enum\_name* specifies the enumeration type name.
* The *enumeration list* is a comma-separated list of identifiers.

Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it. By default, the value of the first enumeration symbol is 0. For example −

enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

## Example

The following example demonstrates use of enum variable −

[Live Demo](http://tpcg.io/JkqN9p)

using System;

namespace EnumApplication {

class EnumProgram {

enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

static void Main(string[] args) {

int WeekdayStart = (int)Days.Mon;

int WeekdayEnd = (int)Days.Fri;

Console.WriteLine("Monday: {0}", WeekdayStart);

Console.WriteLine("Friday: {0}", WeekdayEnd);

Console.ReadKey();

}

}

}

In C#, enum is a value type data type. The enum is used to declare a list of named integer constants. It can be defined using the *enum* keyword directly inside a namespace, class, or structure. The enum is used to give a name to each constant so that the constant integer can be referred using its name.

Example: enum

enum WeekDays

{

Monday = 0,

Tuesday =1,

Wednesday = 2,

Thursday = 3,

Friday = 4,

Saturday =5,

Sunday = 6

}

Console.WriteLine(WeekDays.Friday);

Console.WriteLine((int)WeekDays.Friday);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-4DtLXp)

Output:

Friday   
4

By default, the first member of an enum has the value 0 and the value of each successive enum member is increased by 1. For example, in the following enumeration, Monday is 0, Tuesday is 1, Wednesday is 2 and so forth.

Example: enum

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

Console.WriteLine((int)WeekDays.Monday);

Console.WriteLine((int)WeekDays.Friday);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-vaYjfW)

Output:

0   
4

An explicit cast is necessary to convert from enum type to an integral type. For example, to get the int value from an enum:

Example: enum

int dayNum = (int)WeekDays.Friday;

Console.WriteLine(dayNum);

Output:

4

A change in the value of the first enum member will automatically assign incremental values to the other members sequentially. For example, changing the value of Monday to 10, will assign 11 to Tuesday, 12 to Wednesday, and so on:

Example: enum

enum WeekDays

{

Monday = 10,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

Console.WriteLine((int)WeekDays.Monday);

Console.WriteLine((int)WeekDays.Friday);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-WgZvrt)

Output:

10   
14

The enum can includes named constants of numeric data type e.g. byte, sbyte, short, ushort, int, uint, long, or ulong.

https://www.tutorialsteacher.com/Content/images/tips.pngenum cannot be used with string type.

Enum is mainly used to make code more readable by giving related constants a meaningful name. It also improves maintainability.

## Enum methods:

Enum is an abstract class that includes static helper methods to work with enums.

| Enum method | Description |
| --- | --- |
| Format | Converts the specified value of enum type to the specified string format. |
| GetName | Returns the name of the constant of the specified value of specified enum. |
| GetNames | Returns an array of string name of all the constant of specified enum. |
| GetValues | Returns an array of the values of all the constants of specified enum. |
| object Parse(type, string) | Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. |
| bool TryParse(string, out TEnum) | Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. The return value indicates whether the conversion succeeded. |

Example: enum methods

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

Console.WriteLine(Enum.GetName(typeof(WeekDays), 4));

Console.WriteLine("WeekDays constant names:");

foreach (string str in Enum.GetNames(typeof(WeekDays)))

Console.WriteLine(str);

Console.WriteLine("Enum.TryParse():");

WeekDays wdEnum;

Enum.TryParse<WeekDays>("1", out wdEnum);

Console.WriteLine(wdEnum);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-a2Vqnk)

Output:

Friday   
WeekDays constant names:  
Monday   
Tuesday   
Wednesday   
Thursday  
Friday  
Saturday  
Sunday   
Enum.TryParse():   
Tuesday

Visit MSDN to know more about [C# enum methods](https://msdn.microsoft.com/en-us/library/system.enum.aspx).

https://www.tutorialsteacher.com/Content/images/bulb-glow.png Points to Remember :

1. The enum is a set of named constant.
2. The value of enum constants starts from 0. Enum can have value of any valid numeric type.
3. String enum is not supported in C#.
4. Use of enum makes code more readable and manageable.

# C# - Struct

We have learned [class](https://www.tutorialsteacher.com/csharp/csharp-class) in the previous section. Class is a [reference type](https://www.tutorialsteacher.com/csharp/csharp-value-type-and-reference-type). C# includes a value type entity same as class called "structure". Structs are mainly useful to hold small data values. A structure can be defined using the *struct* operator. It can contain parameterized constructor, static constructor, constants, fields, methods, properties, indexers, operators, events and nested types.

## Structure Declaration

A structure is declared using struct keyword with public, private, or internal modifier. The default modifer is internal for the struct and its members.

The following declares the simple structure that holds data for employees.

Example: Structure

struct Employee

{

public int EmpId;

public string FirstName;

public string LastName;

}

A struct object can be created with or without the new operator, same as primitive type variables. When you create a struct object using the new operator, an appropriate constructor is called.

Example: Create struct object using new keyword

struct Employee

{

public int EmpId;

public string FirstName;

public string LastName;

}

Employee emp = new Employee();

Console.Write(emp.EmpId); // prints 0

In the above code, an object of the structure Employee is created using the new keyword. So, this calls the default parameterless constructor that initializes all the members to their default value.

When you create a structure object without using new keyword, it does not call any constructor and so all the members remain unassigned. So, you must assign values to each member before accessing them, otherwise it will give a compile time error.

Example: Create struct object without using new keyword

struct Employee

{

public int EmpId;

public string FirstName;

public string LastName;

}

Employee emp;

Console.Write(emp.EmpId); // Compile time error

emp.EmpId = 1;

Console.Write(emp.EmpId); // prints 1

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-8DKWmG)

## Constructors in Struct

A struct cannot contain parameterless constructor. It can only contain parameterized constructors or a static constructor. You can declare parameterized constructor to initialize struct members, as shown below.

Example: Parameterized Constructor in Struct

struct Employee

{

public int EmpId;

public string FirstName;

public string LastName;

public Employee(int empid, string fname, string lname)

{

EmpId = empid;

FirstName = fname;

LastName = lname;

}

}

Employee emp = new Employee(10, "Bill", "Gates");

Console.Write(emp.EmpId); // prints 10

Console.Write(emp.FirstName); // prints Bill

Console.Write(emp.LastName); // prints Gates

Please note that you must assign values to all the members of a struct in parameterized constructor, otherwise it will give compile time error if any member remains unassigned.

A struct can include static parameterless constructor and static fields.

Example: Static Constructor in Struct

struct Employee

{

public int EmpId;

public string FirstName;

public string LastName;

static Employee()

{

Console.Write("First object created");

}

public Employee(int empid, string fname, string lname)

{

EmpId = empid;

FirstName = fname;

LastName = lname;

}

}

Employee emp1 = new Employee(10, "Bill", "Gates");

Employee emp2 = new Employee(10, "Steve", "Jobs");

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-GrD2xN)

## Methods and Properties in Struct

The structure can contain properties, auto-properties, or methods, same as class.

Example: Methods and Properties in Struct

struct Employee

{

public int EmpId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public Employee(int empid, string fname, string lname)

{

EmpId = empid;

FirstName = fname;

LastName = lname;

}

public string GetFullName()

{

return FirstName + " " + LastName;

}

}

Employee emp = new Employee(10, "Bill", "Gates");

Console.Write(emp.GetFullName()); // prints Bill Gates

## Events in Struct

A struct can contain events to notify subscriber about some action. Consider the following example.

Example: Structure

struct Point

{

private int \_x, \_y;

public int X

{

get

{

return \_x;

}

set

{

\_x = value;

PointChanged(\_x);

}

}

public int Y

{

get

{

return \_y;

}

set

{

\_y = value;

PointChanged(\_y);

}

}

public event Action<int> PointChanged;

}

The above structure contains private fields \_x and \_y, properties X and Y and PointChanged [event](https://www.tutorialsteacher.com/csharp/csharp-event) to notify if points change. Notice that we raise the PointChanged event whenever X or Y changes. The following code handles the PointChanged event.

Example: Handle Structure Events

class Program

{

static void StructEventHandler(int point)

{

Console.WriteLine("Point changed to {0}", point);

}

static void Main(string[] args)

{

Point.StaticMethod();

Point p = new Point();

p.PointChanged += StructEventHandler;

p.X = 10;

}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-Q7TChQ)

A struct is a value type so it is faster than a class object. Use struct whenever you want to just store the data. Generally structs are good for game programming. However, it is easier to transfer a class object than a struct. So do not use struct when you are passing data across the wire or to other classes.

## Characteristics of Structure:

* Structure can include constructors, constants, fields, methods, properties, indexers, operators, events & nested types.
* Structure cannot include parameterless constructor or destructor.
* Structure can implement interfaces, same as class.
* A structure cannot inherit another structure or class, and it cannot be the base of a class.
* Structure members cannot be specified as abstract, virtual, or protected.

## Difference between Struct and Class:

* Class is reference type whereas struct is value type
* Struct cannot declare a default constructor or destructor. However, it can have parametrized constructors.
* Struct can be instasntiated without the new operator. However, you won't be able to use any of its methods, events or properties if you do so.
* Struct cannot be used as a base or cannot derive another struct or class.