By: Mukunda Paudel
paudelmuku@gmail.com

C# | Namespaces

- ❖ Namespaces are used to organize the classes. It helps to control the scope of methods and classes in larger **.Net** programming projects.
- ❖ Simply you can say that it provides a way to keep one set of names (like class names) different from other sets of names.
- ❖ The biggest advantage of using namespace is that the class names which are declared in one namespace will not clash with the same class names declared in another namespace.
- ❖ It is also referred as **named group of classes** having common features.
- ❖ Namespaces are used in C# to organize and provide a level of separation of codes. They can be considered as a container which consists of other namespaces, classes, etc. A namespace can have following types as its members:
    - ✓ Namespaces (Nested Namespace) Classes
    - ✓ Interfaces
    - ✓ Structures
    - ✓ Delegates.

Namespace also solves the problem of **naming conflict**. Two or more classes when put into different namespaces can have same name.

Defining a Namespace

To define a namespace in C#, we will use the **namespace** keyword followed by the name of the namespace and curly braces containing the body of the namespace as follows:

Syntax:

```
namespace name_of_namespace
   {

       // Namespace (Nested Namespaces)
       // Classes
       // Interfaces
       // Structures
       // Delegates

   }
```

Example:

```
// defining the namespace name1
namespace name1
   {

         // C1 is the class in the namespace name1

class C1

 {
            // class code
       }

   }
```

Accessing the Members of Namespace

The members of a namespace are accessed by using dot(.) operator. A class in C# is fully known by its respective namespace.

[namespace_name].[member_name]

// C# program to illustrate the use of namespaces

```
// namespace declaration
namespace first
{
   // name_1 namespace members i.e. class
   class EverestEngineeringCollege
   {
      // function of class EverestEngineeringCollege
      public static void display()
      {
            //Here System is the namespace under which Console class is defined
         System.Console.WriteLine("Hello From Everest Engineering College");
      }
   }

/* removing comment will give the error because no two classes can have the
   Same name under a single namespace

   class EverestEngineeringCollege
   {

   } */

} // ending of first namespace

// Class declaration
class ComputerIVSem
{
   // Main Method
   public static void Main(String[] args)
   {
    //calling the display method of class EverestEngineeringCollege by using
    two dot operator as one is use to access the class of first namespace and
    another is use to access the static method of class
    EverestEngineeringCollege Termed as fully qualified name
      first. EverestEngineeringCollege.display();

   }
}
```

Nested Namespaces

You can also define a namespace into another namespace which is termed as the nested namespace. To access the members of nested namespace user has to use the dot (.) operator.

For example, Generic is the nested namespace in the *collections* namespace as *System.Collections.Generic*

Syntax:

```
  namespace name_of_namespace_1
 {

 // Member declarations & definitions
 namespace name_of_namespace_2
 {

 // Member declarations & definitions
 .
 .
 .
 }
}
```

EXAMPLE:

```
// C# program to illustrate use of nested namespace
using System;

// you can also use using Main_name.Nest_name; to avoid the use of fully
qualified name
   // main namespace
   namespace Main_name
   {
      // nested namespace
      namespace Nest_name
      {
         // class within nested namespace
         class EEC
         {
            // Constructor of nested namespace class EEC
            public EEC()
            {
               Console.WriteLine("Nested Namespace Constructor");
            }
         }
      }
   }

   // Driver Class
   class Driver
   {
      // Main Method
      public static void Main(string[] args)
      {
                Accessing the Nested Namespace by using fully qualified name "new"
                is used as EEC() is the Constructor
         new Main_name.Nest_name.EEC();

      }
   }
```

I/O

- ❖ A **file** is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a **stream**.
- ❖ The stream is basically the sequence of bytes passing through the communication path.
- ❖ There are two main streams:
    - ✓ **input stream** and
    - ✓ **Output stream**.

The **input stream→**used for reading data from file (read operation) and,

The **output stream→** is used for writing into the file (write operation).

C# I/O Classes

The System.IO namespace has various classes that are used for performing numerous operations with files, such as creating and deleting files, reading from or writing to a file, closing a file etc.

The following table shows some commonly used non-abstract classes in the System.IO namespace –

S.N.	I/O Class & Description
1	**BinaryReader→** Reads primitive data from a binary stream.
2	**BinaryWriter→** Writes primitive data in binary format.
3	**BufferedStream→**A temporary storage for a stream of bytes.
4	**Directory→** Helps in manipulating a directory structure.
5	**DirectoryInfo→** Used for performing operations on directories.
6	**DriveInfo→** Provides information for the drives.

7	**File→** Helps in manipulating files.
8	**FileInfo→** Used for performing operations on files.
9	**FileStream→** Used to read from and write to any location in a file.
10	**MemoryStream→** Used for random access to streamed data stored in memory.
11	**Path→** Performs operations on path information.
12	**StreamReader→** Used for reading characters from a byte stream.
13	**StreamWriter→** Is used for writing characters to a stream.
14	**StringReader→** Is used for reading from a string buffer.
15	**StringWriter→** Is used for writing into a string buffer.

The FileStream Class

S.N	Parameter & Description
1	**FileMode** The **FileMode** enumerator defines various methods for opening files. The members of the FileMode enumerator are • **Append** – It opens an existing file and puts cursor at the end of file, or creates the file, if the file does not exist. • **Create** – It creates a new file.

	• **CreateNew** − It specifies to the operating system, that it should create a new file. • **Open** − It opens an existing file. • **OpenOrCreate** − It specifies to the operating system that it should open a file if it exists, otherwise it should create a new file. • **Truncate** − It opens an existing file and truncates its size to zero bytes.
2	**FileAccess** **FileAccess** enumerators have members: ✓ **Read**, ✓ **ReadWrite** and ✓ **Write**.
3	**FileShare** **FileShare** enumerators have the following members − • **Inheritable** − It allows a file handle to pass inheritance to the child processes • **None** − It declines sharing of the current file • **Read** − It allows opening the file for readin. • **ReadWrite** − It allows opening the file for reading and writing • **Write** − It allows opening the file for writing

The **FileStream** class in the System.IO namespace helps in reading from, writing to and closing files. This class derives from the abstract class Stream.

You need to create a **FileStream** object to create a new file or open an existing file. The syntax for creating a **FileStream** object is as follows –

FileStream < object_name > = new FileStream(< file_name >, < FileMode Enumerator >, < FileAccess Enumerator >, < FileShare Enumerator >);

For example, we create a FileStream object **F** for reading a file named **sample.txt**

```
FileStream F = new FileStream("sample.txt", FileMode.Open, FileAccess.Read,
  FileShare.Read);
```

The following program demonstrates use of the **FileStream** class –

```
using System;
using System.IO;

namespace FileIOApplication
   {
      class Program
      {
         static void Main(string[] args)
         {
            FileStream F = new FileStream("test.dat", FileMode.OpenOrCreate,
              FileAccess.ReadWrite);

            for (int i = 1; i <= 20; i++)
            {
               F.WriteByte((byte)i);
            }
            F.Position = 0;
            for (int i = 0; i <= 20; i++)
            {
               Console.Write(F.ReadByte() + " ");
            }
            F.Close();
            Console.ReadKey();
         }
      }
   }
```

Out Put

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

Dot Net Library:

- ❖ **Library** is usually a collection of classes intended for a specific purpose,

  For. E.g. System.**Net library** in .**NET** that is intended for internet communication.

- ❖ In .NET there are various liberary, some of them are:

1. **Base Class Library** (BCL)
   - ✓ Literally it is the **base**.
   - ✓ It contains basic, fundamental types like System.String and System.DateTime .
2. The Framework **Class Library** (FCL)
   - ✓ The wider **library** that contains the totality: ASP.**NET**, WinForms, the XML stack, ADO.**NET** and more.
3. A Dynamic Link **library** (DLL)
   - ✓ It contains functions and codes that can be used by more than one program at a time.
   - ✓ Once we have created a DLL file, we can use it in many applications.

- ❖ As a Built in libraries, .NET implementations include classes, interfaces, delegates, and value types that optimize the development process and provide access to system functionality.
- ❖ .NET types/ libraries are the foundation on which .NET applications, components, and controls are built.
- ❖ .NET implementations include types that perform the following functions:

- ✓ Represent base data types and exceptions.
- ✓ Encapsulate data structures.
- ✓ Perform I/O.
- ✓ Access information about loaded types.
- ✓ Invoke .NET Framework security checks.
- ✓ Provide data access, rich client-side GUI, and server-controlled, client-side GUI.