Answer on Question #58574 – Programming & Computer Science – C#

Question:

Input and output operations in the files

Solution:

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: the **input stream** and the **output stream**. The **input stream** is 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:

I/O Class	Description
BinaryReader	Reads primitive data from a binary stream.
BinaryWriter	Writes primitive data in binary format.
BufferedStream	A temporary storage for a stream of bytes.
Directory	Helps in manipulating a directory structure.
DirectoryInfo	Used for performing operations on directories.
DriveInfo	Provides information for the drives.
File	Helps in manipulating files.
FileInfo	Used for performing operations on files.
FileStream	Used to read from and write to any location in a file.
MemoryStream	Used for random access to streamed data stored in memory.
Path	Performs operations on path information.
StreamReader	Used for reading characters from a byte stream.
StreamWriter	Is used for writing characters to a stream.

StringReader	Is used for reading from a string buffer.
StringWriter	Is used for writing into a string buffer.

The FileStream Class

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 as shown**:

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

Parameter	Description
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.
FileAccess	**FileAccess** enumerators have members: **Read**, **ReadWrite**and **Write**.
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 reading **ReadWrite**: It allows opening the file for reading and writing

	Write: It allows opening the file for writing

Example

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();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result:

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