Files in C#

Presentation · June 2021

DOI: 10.13140/RG.2.2.29503.84645

1 author:

Tarfa Hamed
University of Mosul
38 PUBLICATIONS **229** CITATIONS

SEE PROFILE

Advanced Programming in C#

Lecture Six

C# File 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: 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:

No.	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

- 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);
```

No.	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`**: Specifies that the operating system should create a new file. If the file already exists, it will be overwritten. • **`CreateNew`**: It specifies that the operating system should create a new file. If the file already exists, an **`IOException`** exception is thrown. • **`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: • **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 myfile = new FileStream("test.dat",
FileMode.OpenOrCreate, FileAccess.ReadWrite);

   for (int i = 0; i <= 20; i++)
    {
```

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

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

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Advanced File Operations in C#

- The preceding example provides simple file operations in C#.
- However, to utilize the immense powers of C# System.IO classes, you need to know the commonly used properties and methods of these classes.

No.	Topic & Description
1.	**Reading from and Writing to Text Files** It involves reading from and writing into text files. The StreamReader and StreamWriter class helps to accomplish it.
2.	**Reading from and Writing into Binary files** It involves reading from and writing into binary files. The BinaryReader and BinaryWriter class helps to accomplish this.
3.	**C# - Windows File System** It gives a C# programmer the ability to browse and locate Windows files and directories.

Reading from and Writing to Text Files

- The **`StreamReader`** and **`StreamWriter`** classes are used for reading from and writing data to text files.
- These classes inherit from the abstract base class **`Stream`**, which supports reading and writing bytes into a file stream.

How to read from an existing file

- Here we are going to learn how to read information from an existing file.
- This task can be performed by using the **`StreamReader`** class.

The StreamReader Class

- The **`StreamReader`** class also inherits from the abstract base class **`TextReader`** that represents a reader for reading series of characters.

- The following table describes some of the commonly used methods of the **`StreamReader`** class:

No.	Method & Description
1.	**public override void Close()** It closes the **`StreamReader`** object and the underlying stream, and releases any system resources associated with the reader.
2.	**public override string ReadLine()** Reads a line of characters from the current stream and returns the data as a string.
3.	**public override int Read()** Reads the next character from the input stream and advances the character position by one.

Example

- The following example demonstrates reading a text file named ***test.txt***. The file reads

Hello

This is my test file

see you next week

```
using System;
using System.IO;
namespace FileApplication {
```

```
   class Program {
      static void Main(string[] args) {
// Create an instance of StreamReader to read
from a file.
// The using statement also closes the
StreamReader.
StreamReader sr = new StreamReader("c:/test.txt")

   string line;
  // Read and display lines from the file until
  // the end of the file is reached.
 while ((line = sr.ReadLine()) != null)
  {
     Console.WriteLine(line);
  }
   }
   }
}
```

Guess what it displays when you compile and run the program!

How to write information to a file

- In order to save the information from being deleted upon turning off the computer, these information could be stored in file on the storage with a specified name.
- This task can be performed using the **`StreamWriter`** class.

The StreamWriter Class

- The **`StreamWriter`** class inherits from the abstract class **`TextWriter`** that represents a writer, which can write a series of character.
- The following table describes the most commonly used methods of this class

No.	Method & Description
1.	**public override void Close()** Closes the current StreamWriter object and the underlying stream.
2.	**public override void Flush()** Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream.
3.	**public virtual void Write(bool value)** Writes the text representation of a Boolean value to the text string or stream. (Inherited from TextWriter.)
4.	**public override void Write(char value)** Writes a character to the stream.
5.	**public virtual void Write(decimal value)** Writes the text representation of a decimal value to the text string or stream.
6.	**public virtual void Write(double value)** Writes the text representation of an 8-byte floating-point value to the text string or stream.

7.	**public virtual void Write(int value)** Writes the text representation of a 4-byte signed integer to the text string or stream.
8.	**public override void Write(string value)** Writes a string to the stream.
9.	**public virtual void WriteLine()** Writes a line terminator to the text string or stream.

Example

- The following example demonstrates writing text data into a file using the **`StreamWriter`** class:

```
using System;
using System.IO;

namespace FileApplication {
   class Program {
  static void Main(string[] args) {
  string[] names = new string[] {"Programming1",
" Programming2"};

 StreamWriter sw = new StreamWriter("names.txt")
 foreach (string s in names)
```

```
   {
     sw.WriteLine(s);
   }
   sw.Close();
 // Read and show each line from the file.
string line = "";
StreamReader sr = new StreamReader("names.txt");
while ((line = sr.ReadLine()) != null)
   {
     Console.WriteLine(line);
   }
sr.Close();
       }
    }
}
```

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

```
Programming1
Programming2
```

Homework:

HW1: Write a program in C# that reads items names and items prices of 50 items from KB and stores them in a file (***prices.txt***) line-by-line. After that, the program reads the previous file and writes the items that are priced over $100 in a separate file (***prices_100.txt***).

Example of ***prices.txt*** contents:

```
Rice 125
Tomato paste 50
Cooking Oil 30
Flour 120
```

The contents of ***prices_100.txt*** will be:

```
Rice 125
Flour 120
```

HW2: Write a program in C# that reads a line consists of an unknown number of words. The program then writes each word on a separate line in a file named (***words.txt***).

HW3: Write a program in C# to read students information from a file named (***f1.csv***). The file contains information for **100** students. Each student has one line consists of:

Name(string), **prg**(int), **logic**(int), **math**(int), **level**(string). The above parts are separated by commas (,).

The program is required to write the following information on a second filed named (***smmary.txt***):

a- The highest mark in programming (**prg**).

b- The lowest mark in **logic**.

c- The average of **math** marks.

After that, the program needs to create two more files named (***passed.txt***) and (***failed.txt***) respectively. The first file contains all the students who passed in all subjects. The second file contains all the students who failed in one or more subjects.

HW4: Write a method in C# to count the number of blank spaces in a text file named "***input.txt***"

HW5: Write a method to count number of words in a text file named "***story.txt***".

HW6: Write a program in C# to print the count of word "the" as a separate word in a file named transcript.txt. For example, if the content of the file ***transcript.txt*** is:

```
There was a monkey in the zoo.
The monkey was very naughty.
```

Then the output of the program should be 2

HW7: Assume that you have a text file named file1.dat contains some text written into it, write a method in C# named **`vowel_words()`**, that reads the file ***file1.dat*** and creates a new file named ***file2.dat***, to include only those words from the file ***file1.dat*** which start with a lowercase vowel (i.e., with 'a', 'e', 'i', 'o', 'u').

For example, if the file ***file1.dat*** contains:

```
Carry umbrella and overcoat when it rains
```

Then the file ***file2.dat*** shall contain:

```
umbrella and overcoat it
```