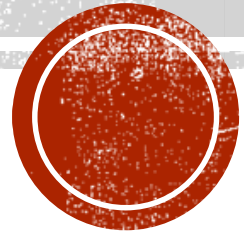
# C# FILE IO PART 1

Elizabeth Bourke



# FILE AND STREAMS

- A file is an ordered and named collection of bytes that has persistent storage.
- When you work with files, you work with directory paths, disk storage, and file and directory names.
- 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).

# SYSTEM.IO

- In the .NET Framework, the System.IO namespaces contain types that enable reading and writing, on data streams and files.
- You can use the types in the System.IO namespace to interact with files and directories. For example, you can get and set properties for files and directories, and retrieve collections of files and directories based on search criteria.

Here are some commonly used file and directory classes:

- File - provides static methods for creating, copying, deleting, moving, and opening files, and helps create a FileStream object.
- FileInfo - provides instance methods for creating, copying, deleting, moving, and opening files, and helps create a FileStream object.
- Directory - provides static methods for creating, moving, and enumerating through directories and subdirectories.
- DirectoryInfo - provides instance methods for creating, moving, and enumerating through directories and subdirectories.
- Path - provides methods and properties for processing directory strings in a cross-platform manner.

# STREAMS

- FileStream – for reading and writing to a file.

The abstract base class Stream supports reading and writing bytes.
All classes that represent streams inherit from the Stream class.
The Stream class and its derived classes provide a common view of data sources and repositories, and isolate the programmer from the specific details of the operating system and underlying devices.

- There are several kinds of streams other than file streams, such as network, memory, and pipe streams.
- Streams only flow in one direction

# READERS AND WRITERS

The System.IO namespace also provides types for reading encoded characters from streams and writing them to streams.

Typically, streams are designed for byte input and output.

The reader and writer types handle the conversion of the encoded characters to and from bytes so the stream can complete the operation.

Each reader and writer class is associated with a stream, which can be retrieved through the class's BaseStream property.

# READERS AND WRITERS

Here are some commonly used reader and writer classes:

- BinaryReader and BinaryWriter – for reading and writing primitive data types as binary values.
- StreamReader and StreamWriter – for reading and writing characters by using an encoding value to convert the characters to and from bytes.
- StringReader and StringWriter – for reading and writing characters to and from strings.
- TextReader and TextWriter – serve as the abstract base classes for other readers and writers that read and write characters and strings, but not binary data.

# THE FILESTREAM CLASS

- Provides a Stream for a file,
- https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx

# C# File IO – 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, for creating a FileStream object **F** for reading a file named **sample.txt**:

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

# FILE MODE

- 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.

# THE FILESTREAM CLASS

- FileAccess -has  members: Read , ReadWrite and Write .
- FileShare – has 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

# C# File IO – FileStream Class Example

```
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();
            }
      }
}
```

# C# File IO – FileStream Class Example

- When the previous code is compiled and executed, it produces the following result to the screen:

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

- Note the -1 at the end, this is the end of file stream marker
- The file test.dat is created in the bin/debug folder
- Take a look at the contents

# STREAM READER AND STREAM WRITER

- 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.

```
using System;
using System.IO;

class Test
{
    public static void Main()
    {
        try
        {
            // Create an instance of StreamReader to read from a file.
            // The using statement also closes the StreamReader.
            using (StreamReader sr = new StreamReader("TestFile.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);
                }
            }
        }
        catch (Exception e)
        {
            // Let the user know what went wrong.
            Console.WriteLine("The file could not be read:");
            Console.WriteLine(e.Message);
        }
    }
}
```

```
namespace StreamReadWrite
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get the directories currently on the C drive.
            DirectoryInfo[] cDirs = new DirectoryInfo(@"c:\").GetDirectories();

            // Write each directory name to a file.
            using (StreamWriter sw = new StreamWriter("CDriveDirs.txt"))
            {
                foreach (DirectoryInfo dir in cDirs)
                {
                    sw.WriteLine(dir.Name);

                }
            }

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

# USING STATEMENT

Its also easier to read.

- Placing code in a using block ensures that the objects are disposed (though not necessarily collected) as soon as control leaves the block.
- C#, through the .NET Framework common language runtime (CLR), automatically releases the memory used to store objects that are no longer required. Memory is released whenever the CLR decides to perform garbage collection. However, it is usually best to release limited resources such as file handles and network connections as quickly as possible

Using calls Dispose() after the using-block is left, even if the code throws an exception.
So you usually use using for classes that require cleaning up after them, like IO.

```
using (MyClass mine = new MyClass())
{
  mine.Action();
}
```

# EXCERCIES

- Create a file – (if not found) and allow the user to input name and address and save details to file.
- Read and display these details to the user.