Visual C# .Net using framework 4.5

Eng. Mahmoud Ouf

Lecture 11

Files and Streams

Most of Files and Streams classes are defined in System.IO namespace

Many of the types within the System.IO namespace focus on the:

- Programmatic manipulation of physical directories and files (Navigating File System).
- Consistent programming interface for reading and writing across a variety of I/O types (Streams).

Navigating File System

This task includes navigating and gathering information about drives, folders, and files.

The file system classes are separated into two types of classes:

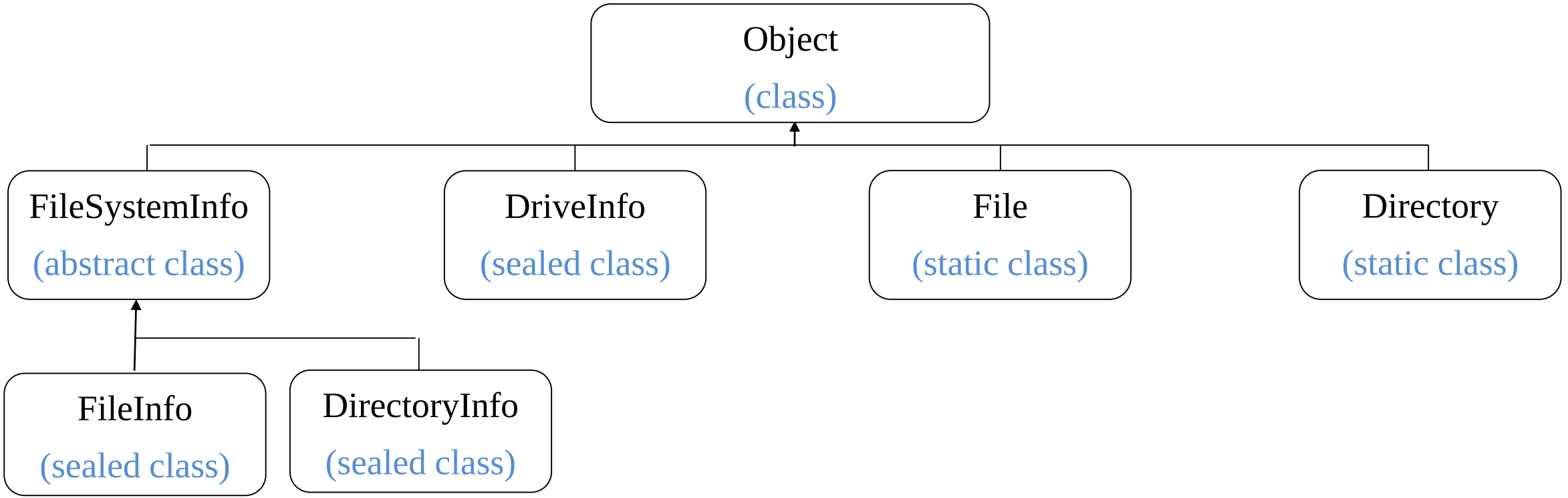
Informational:

- expose all the system information about file system objects specifically, files, directories, and drives.
- These classes are named FileInfo and DirectoryInfo, they inherits from FileSystemInfo base class.
- In addition, the DriveInfo class represents a drive in the file system

Utility:

- provide static methods to perform certain operations on file system objects such as files, directories, and file system paths.
- These utility classes include the File, Directory, and Path classes.

Navigating File System



Navigating File System (Informational)

FileSystemInfo class

The FileSystemInfo class provides the basic functionality for all informational file system classes.

Properties	*Description*
Attributes	*Gets or sets FileAttributes enumeration of the current file or directory.*
CreationTime	*Gets or sets the time that the current file or directory was created*
Exists	*Determines whether the file or directory exists. .*
Extension	*Gets a string representation of the extension part of the file or directory*
FullName	*Gets the full path to the file or directory*
LastAccessTime	*Gets or sets the time the file or directory was accessed*
LastWriteTime	*Gets or sets the time the file or directory was last written to. .*

Navigating File System (Informational)

FileSystemInfo class

Properties	*Description*
Name	*Gets the simple name for the file or directory. For a file, this is the name within the directory. For a directory, this is the last directory name in the directory hierarchy*

Methods	*Description*
Delete	*Removes the file or directory from the file system .*

Navigating File System (Informational)

FileInfo class

It provides the basic functionality to access and manipulate a single file in the file system..

Properties	*Description*
Directory	*Gets the DirectoryInfo object that represents the directory that this file is stored within.*
DirectoryName	*Gets the name of the directory that this file is stored within*
IsReadOnly	*Gets or sets a value that determines if the current file is read only..*
Length	*Gets the length of the file*

Navigating File System (Informational)

FileInfo class

Methods	*Description*
AppendText	*Creates a new StreamWriter that will allow appending text to the file.*
CopyTo	*Makes a copy of the file in a new location*
Create	*Creates a file based on the current file information.*
CreateText	*Creates a new StreamWriter and a new file for writing text*
Decrypt	*Decrypts a file that was encrypted by the current user*
Encrypt	*Encrypts a file so that only the account used to encrypt the file can decrypt it.*
MoveTo	*Moves the file to a new location*

Navigating File System (Informational)

FileInfo class

Methods	*Description*
Open	*Opens the file with specific privileges (read, read/write, and so on).*
OpenRead	*Opens the file with read-only access*
OpenText	*Opens the file and returns a StreamReader to allow reading of text within the file.*
OpenWrite	*Opens the file with write-only access*
Replace	*Replaces a file with the information in the current FileInfo object.*

Navigating File System (Informational)

DirectoryInfo class

This class contains a set of members used for creating, moving, deleting, and enumerating over directories and subdirectories.

Properties	*Description*
Parent	*Gets the DirectoryInfo object for the parent directory of the current directory in the directory hierarchy*
Root	*Gets the root part of the directory's path as a string*

Navigating File System (Informational)

DirectoryInfo class

Methods	*Description*
Create	*Creates the directory described in the current DirectoryInfo object*
CreateSubdirectory	*Creates a new directory as a child directory of the current directory in the directory hierarchy*
Delete	*Deletes a directory and all its contents*
GetDirectories	*Retrieves an array of DirectoryInfo objects that represent sub directories of the current directory*
GetFiles	*Retrieves an array of FileInfo objects that represent all the files in the current directory*
GetFileSystemInfos	*Retrieves an array of FileSystemInfo objects that represent both files and subdirectories in the current directory*
MoveTo	*Moves the current directory to a new location*

Navigating File System (Informational)

Begin working with the DirectoryInfo type by specifying a particular directory path as a constructor parameter.

```
DirectoryInfo dir2 = new DirectoryInfo(@"C:\Windows");
```

We use the "." notation to obtain access to the current application directory.

```
DirectoryInfo dir1 = new DirectoryInfo(".");
```

if you attempt to interact with a nonexistent directory, a System.IO.DirectoryNotFoundException is thrown.

Thus, if you specify a directory that is not yet created, you will need to call the Create() method before proceeding:

```
DirectoryInfo dir3 = new DirectoryInfo(@"C:\Windows\Testing");
dir3.Create();
```

Navigating File System (Informational)

DriveInfo class

Properties	*Description*
AvailableFreeSpace	*Gets the amount of available space on the drive. The amount might be different from the amount returned by TotalFreeSpace, depending on disk quotas.*
DriveFormat	*Gets the format of the drive, such as NTFS or FAT32.*
IsReady	*Gets the status of the drive, indicating whether it is ready to be accessed..*
Name	*Gets the name of the drive.*
RootDirectory	*Gets a DirectoryInfo object that represents the root directory of the drive*
VolumeLabel	*Gets or sets the label of the drive. It might be set only on drives that are not readonly..*
TotalFreeSpace	*Gets the total amount of free space on the drive*

Navigating File System (Informational)

DriveInfo class

Properties	*Description*
TotalSize	*Gets the total size of the drive*
DriveType	*Gets the type of drive in the form of the DriveType enumeration*

Methods	*Description*
GetDrives	*A static method that returns all the drives on the current system.*

The DriveType Enumeration

The DriveType enumeration provides the possible types of drives that can be represented by a DriveInfo object.

CDRom *Fixed* *Network* *Ram* *Removable*

Navigating File System (Utility)

File class

This is a static class contains a set of static Methods help in crating moving deleting …files. Also, it has some extra methods helps when dealing with I/O stream. There are some methods are duplicated in function with other methods in FileInfo class, so you can use anyone of them.

Methods	*Description*
AppendAllText	*Appends a specified string into an existing file, alternatively creating the file if it does not exist*
AppendText	*Opens a file (or creates a new file if one does not exist) and returns a StreamWriter that is prepared to allow text to be appended to the file*
Copy	*Copies a file to a new file. The new file must not exist for Copy to be successful..*
Create	*Creates a new file and returns a FileStream object*

Navigating File System (Utility)

File class

Methods	*Description*
Move	*Moves a file from one place to another*
CreateText	*Creates or opens a file and returns a StreamWriter object that is ready to have text written into it*
OpenRead	*Opens an existing file and returns a read-only FileStream object.*
OpenText	*Opens an existing file and returns a StreamReader object*
OpenWrite	*Opens an existing file for writing and returns a StreamWriter object*
ReadAllBytes	*Opens a file, reads the contents of it into a byte array, and closes the file in one atomic operation*
ReadAllLines	*Opens a file, reads the contents of it into an array of strings (one per line), and closes the file in one atomic operation*

Navigating File System (Utility)

File class

Methods	*Description*
ReadAllText	*Opens a file, reads the contents of it into a string, and closes the file in one atomic operation.*
WriteAllBytes	*Opens a file, writes the contents of a byte array into it (over¬writing any existing data), and closes the file in one atomic operation.*
WriteAllLines	*Opens a file, writes the contents of a string array into it (overwriting any existing data), and closes the file in one atomic operation.*
WriteAllText	*Opens a file, writes the contents of a string into it (overwrit¬ing any existing data), and closes the file in one atomic operation*

Navigating File System (Utility)

Directory class

The .NET Framework supports the Directory class, which presents a static/shared interface for manipulating and creating directories in the file system. The Directory class provides the basic functionality to open file streams for reading and writing.

Methods	*Description*
CreateDirectory	*Creates all the directories in a supplied path*
Delete	*Deletes a specified directory*
Exists	*Determines whether a directory exists in the file system*
GetCreationTime	*Returns the creation time and date of a directory*
GetCurrentDirectory	*Returns a DirectoryInfo object for the current working directory of the application*

Navigating File System (Utility)

Directory class

Methods	*Description*
GetDirectories	*Gets a list of names for subdirectories in a specified directory*
GetDirectoryRoots	*Returns the volume and/or root information for a specified directory*
GetFiles	*Returns the names of files in a directory.*
GetFileSystemEnteries	*Returns a list of subdirectories and files in the specified directory*
GetLastAccessTime	*Returns the time that a specified directory was last accessed*
GetLastWriteTime	*Returns the time that a specified directory was last written to*
GetLogicalDrives	*Gets a list of drives in the current system as strings with the pattern of "C:\"*

Navigating File System (Utility)

Directory class

Methods	*Description*
GetParent	*Gets the parent directory of the specified directory.*
Move	*Moves a file or directory (and its contents) to a specified place.*
SetCreationTime	*Sets the time a specific directory was created.*
SetCurrentDirectory	*Sets the specified directory to be the current working directory for an application*
SetLastAccessTime	*Sets the last time a directory was accessed*
SetLastWriteTime	*Sets the last time a directory was written to*

Streams (Reading / Writing files)

Stream represents a chunk of data flowing between a source and a destination.

Streams provide a common way to interact with a sequence of bytes, regardless of what kind of device store (e.g., file, network connection, and printer)

The abstract Stream class defines several members that provide support for synchronous and asynchronous interactions with the storage medium.

The Stream class is the base class for all types of streams

Streams (Reading / Writing files)

Stream class	
Properties	*Description*
CanRead	*Determines whether the stream supports reading*
CanSeek	*Determines whether the stream supports seeking*
CanTimeOut	*Determines whether the stream can time out*
CanWrite	*Determines whether the stream can be written to*
Length	*Gets the length (in bytes) of the stream*
Position	*Gets or sets the virtual cursor for determining where in the stream the current position is. The value of Position cannot be greater than the value of the stream's Length*
ReadTimeOut	*Gets or sets the stream's timeout for read operations*

Streams (Reading / Writing files)

Stream class	
Methods	*Description*
Close	*Closes the stream and releases any resources associated with it*
Flush	*Clears any buffers within the stream and forces changes to be written to the underlying system or device*
Read	*Performs a sequential read of a specified number of bytes from the current position and updates the position to the end of the read upon completion of the operation*
ReadByte	*Performs the read of a single byte and updates the position by moving it by one. Identical to calling Read to read a single byte*
Write	*Writes information to the stream as a number of bytes and updates the current position to reflect the new write position*
WriteByte	*Writes a single byte to the stream and updates the position. Identical to calling Write with a single byte*

Streams (Reading / Writing files)

All stream classes in the .NET Framework derive from the Stream class. These derived classes include the following:

- FileStream (System.IO)
- MemoryStream (System.IO)
- CryptoStream (System.Security)
- NetworkStream (System.Net)
- GZipStream (System.Compression)

By learning how to work with a stream in general, you can apply that knowledge to any type of stream

Working with FileStream

The FileStream class provides an implementation for the abstract Stream members in a manner appropriate for file-based streaming.

It is a fairly primitive stream; it can read or write only a single byte or an array of bytes.

1. Create an object from FileStream, using the Open Method of File Class
2. As FileStream operates only on raw bytes, we need to encode the data to corresponding byte array. System.Text namespace contains class Encoding which contains method to encode and decode the data.
3. Write array of bytes to file using Write method of FileStream class, OR use WriteByte in an array
4. Reset Internal Position of FileStream, set Position to 0
5. Read from File to an array of byte, use ReadByte method in a loop, OR use Read Method
6. Decode the array of byte to original data type

Working with FileStream

```
class Test
{
        static void Main(string[] args)
        {
                // Obtain a FileStream object.
                FileStream fStream = File.Open(@"C:\myMessage.dat",
                                               FileMode.Create);
                // Encode a string as an array of bytes.
                string msg = "Hello!";
                byte[] msgAsByteArray = Encoding.Default.GetBytes(msg);
                // Write byte[] to file.
                fStream.Write(msgAsByteArray, 0, msgAsByteArray.Length);
                // Reset internal position of stream.
                fStream.Position = 0;
```

Working with FileStream

```
            // Read the types from file and display to console.
            Console.Write("Your message as an array of bytes: ");
            byte[] bytesFromFile = new byte[msgAsByteArray.Length];
            for (int i = 0; i < msgAsByteArray.Length; i++)
            {
                bytesFromFile[i] = (byte)fStream.ReadByte();
                Console.Write(bytesFromFile[i]);
            }
            // Display decoded messages.
            Console.Write("\nDecoded Message: ");
            Console.WriteLine(Encoding.Default.GetString(bytesFromFile));
            Console.ReadLine();
        }
}
```

Working with Files

While the using of FileStream class is useful for writing and reading data to files, it is a complicated process as it use the data in the raw format.

So, we need to change the data to the raw format before writing data

We need to restore the raw format to the initial form after reading data

Four classes are defined for reading and writing data.

These classes encapsulates the process of Writing data to stream and reading data from stream in an easy manner than the classical FileStream

StreamReader and StreamWriter are used to deal with text data

BinaryReader and BinaryWriter are used to deal with binary data

Writing to a Text File:

1. Create a StreamWriter object either by:
   a) Using CreateText method from File class
   b) Using Create method from File, that return FileStream, and then send this object to the constructor of the StreamWriter
2. Use the StreamWriter methods
3. Close the StreamWriter using Close method

Note: If you create the StreamWriter from available FileStream, YOU MUST close the FileStream object after closing the StreamWriter object

Writing to a Text File:

```
class Test
{
        public static void Main(string[] args)
        {
                // Get a StreamWriter and write string data.
                StreamWriter writer = File.CreateText("reminders.txt")
                writer.WriteLine(“Welcome to Text Files");
                writer.WriteLine(“Welcome to StreamWriter");
                for(int i = 0; i < 10; i++)
                        writer.Write(i + " ");
                // Insert a new line.
                writer.Write(writer.NewLine);
                writer.Close();
                Console.ReadLine();
        }
}
```

Reading from a Text File:

1. Create a StreamReader object either by:
   a) Using OpenText method from File class
   b) Using Create method from File, that return FileStream, and then send this object to the constructor of the StreamReader
2. Use the StreamReader methods
3. Close the streamReader using Close method

Note: If you create the StreamReader from available FileStream, YOU MUST close the FileStream object after closing the StreamReader object.

Note: If all you need to do is read out the entire file, the File class supports reading the file in a single method call, hiding all the details of the stream and reader implementation by calling its ReadAllText method

```
Console.WriteLine(File.ReadAllText(@”c:\myfile.txt”));
```

Reading from a Text File:

```
class Test
{
        public static void Main(string[] args)
        {
                // Now read data from file.
                StreamReader sr = File.OpenText("reminders.txt")
                string input;
                while ((input = sr.ReadLine()) != null)
                {
                        Console.WriteLine (input);
                }
                sr.Close();
                Console.ReadLine();
        }
}
```

Writing to a Binary File:

1. Create a BinaryWriter object by sending its constructor an object from Stream “FileStream in case to deal with files”(this is created by using File class methods, or FileInfo class methods)
2. Use the BinaryWriter methods
3. Close the BinaryWriter using Close method

Writing to a Binary File:

```
class Test
{
        public static void Main(string[] args)
        {
                // Open a binary writer for a file.
                FileInfo f = new FileInfo("BinFile.dat");
                BinaryWriter bw = new BinaryWriter(f.OpenWrite());
                // Create some data to save in the file
                double aDouble = 1234.67;
                int anInt = 34567;
                string aString = "A, B, C";
                // Write the data
                bw.Write(aDouble);
                bw.Write(anInt);
                bw.Write(aString);
                bw.Close();
        }
}
```

Reading from a Binary File:

1. Create a BinaryReader object by sending its constructor an object from Stream “FileStream in case to deal with files”(this is created by using File class methods, or FileInfo class methods)
2. Use the BinaryReader methods
3. Close the BinaryReader using Close method

Reading from a Binary File:

```
class Test
{
        public static void Main(string[] args)
        {
                FileInfo f = new FileInfo("BinFile.dat");
                // Read the binary data from the stream.
                BinaryReader br = new BinaryReader(f.OpenRead())
                Console.WriteLine(br.ReadDouble());
                Console.WriteLine(br.ReadInt32());
                Console.WriteLine(br.ReadString());
                br.Close();
                Console.ReadLine();
        }
}
```