Stream and File

Windows Programming Course

Agenda

1. Managing the File System
2. Working with Streams

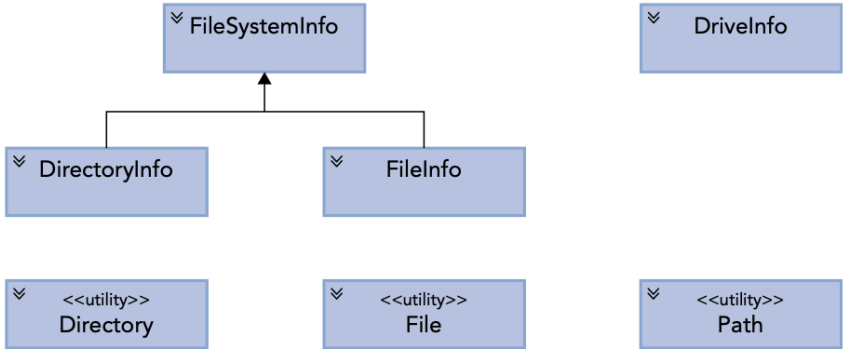
1.
Managing the File System

Managing the File System

`FileSystemInfo` - The base class represents any file system object.

`FileInfo` and `File` - These classes represent a file on the file system.

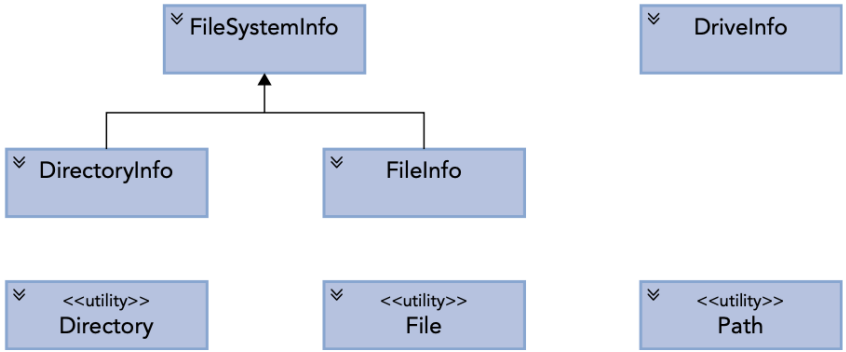
`DirectoryInfo` and `Directory` - These classes represent a folder on the file system.



Managing the File System

`Path` - This class contains static members that you can use to manipulate pathnames.

`DriveInfo` - This class provides properties and methods that provide information about a selected drive.



Checking Drive Information

```
DriveInfo[] drives = DriveInfo.GetDrives();

foreach (DriveInfo drive in drives)
{
  if (drive.IsReady) {
    Console.WriteLine($"Drive name: {drive.Name}");
    Console.WriteLine($"Format: {drive.DriveFormat}");
    Console.WriteLine($"Type: {drive.DriveType}");
    Console.WriteLine($"Root directory: {drive.RootDirectory}");
    Console.WriteLine($"Volume label: {drive.VolumeLabel}");
    Console.WriteLine($"Free space: {drive.TotalFreeSpace}");
    Console.WriteLine($"Available space:
    {drive.AvailableFreeSpace}");
    Console.WriteLine($"Total size: {drive.TotalSize}");
    Console.WriteLine();
  }
}
```

Working with the Path Class

The Path class combines multiple folders and files using string concatenation operators and it also deals with different platform requirements on Windows- and Unix-based systems.

```
Console.WriteLine(Path.Combine(@"D:\Projects",
"ReadMe.txt"));
```

Creating Files and Folders

```
// Using File
File.WriteAllText(fileName, "Hello, World!");
File.Copy(fileName1, fileName2);

// Using FileInfo and DirectoryInfo
var file = new FileInfo(fileName1);
file.CopyTo(fileName2);

var myFolder = new DirectoryInfo(directory1);
```

Accessing and Modifying File Properties

```
var file = new FileInfo(fileName);
Console.WriteLine($"Name: {file.Name}");
Console.WriteLine($"Directory: {file.DirectoryName}");
Console.WriteLine($"Read only: {file.IsReadOnly}");
Console.WriteLine($"Extension: {file.Extension}");
Console.WriteLine($"Length: {file.Length}");
Console.WriteLine($"Creation time: {file.CreationTime:F}");
Console.WriteLine($"Access time: {file.LastAccessTime:F}");
Console.WriteLine($"File attributes: {file.Attributes}");
```

Using File to Read and Write

```
string[] lines = File.ReadAllLines(fileName);
int i = 1;
foreach (var line in lines) {
  Console.WriteLine($"{i++}. {line}");
}
```

Using File to Read and Write (cont.)

```
string fileName = "movies.txt";
string[] movies =
{
  "Snow White And The Seven Dwarfs",
  "Gone With The Wind",
  "Casablanca",
  "The Bridge On The River Kwai",
  "Some Like It Hot"
};
File.WriteAllLines(fileName, movies);
```

Hands-on Exercise

Implement a console app to:

- List information of all drives on your PC
- List all file in your MyDocuments folder (Try to avoid hard-code the MyDocuments path)
- Read a text file in MyDocuments folder and print to the console.

2.
Working with Streams

The idea of a stream

A stream is an object used to transfer data. The data can be transferred in one of two directions:

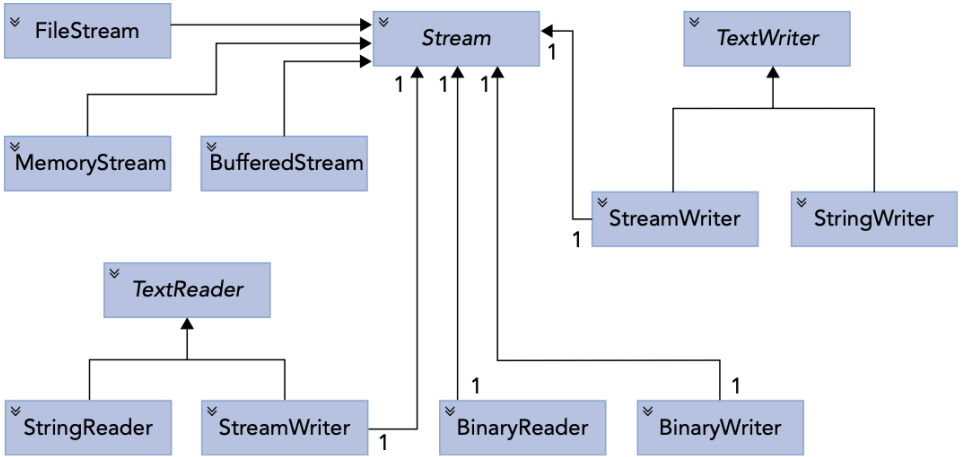
- If the data is being transferred from some outside source into your program, it is called reading from the stream.
- If the data is being transferred from your program to some outside source, it is called *writing* to the stream.

Working with Streams

`FileStream` - This class is intended for reading and writing binary data in a file.

`StreamReader` and `StreamWriter` - These classes are designed specifically for reading from and writing to streams offering APIs for text formats.

`BinaryReader` and `BinaryWriter` - These classes are designed for reading and writing to streams offering APIs for binary data.



Closing and Disposing stream when done

```
FileStream s = new FileStream(@"D:\Test.txt");
try {
  ReadFile(s);
} finally {
  if (s != null) ((IDisposable)s).Dispose();
}
```

`using` Statement

The using statement obtains one or more resources, executes a statement, and then disposes the resources.

```
using (var s = FileStream(@"D:\Test.txt")) {
 ReadFile(s);
}
```

Working with File Streams

A FileStream instance is used to read or write data to or from a file. To construct a FileStream, you need four pieces of information:

- *Path of file*
- *FileMode:* Append, Create, CreateNew, Open, OpenOrCreate, or Truncate
- *FileAccess:* Read, ReadWrite, or Write
- *FileShare*: Delete, Inheritable, None, Read, ReadWrite, or Write

Creating a FileStream

```
private void ReadFileUsingFileStream(string fileName) {
const int bufferSize = 4096;
using (var stream = new FileStream(fileName, FileMode.Open,
FileAccess.Read, FileShare.Read))
{
  ShowStreamInformation(stream);
  // Reading stream…
```

Getting Stream Information

```
private void ShowStreamInformation(Stream stream) {
  Console.WriteLine($"stream can read: {stream.CanRead}, " +
  $"can write: {stream.CanWrite}, can seek: {stream.CanSeek}, " +
  $"can timeout: {stream.CanTimeout}");

  Console.WriteLine($"length: {stream.Length}, position:
{stream.Position}");
  if (stream.CanTimeout)
  {
    Console.WriteLine($"read timeout: {stream.ReadTimeout} " +
$"write timeout: {stream.WriteTimeout} ");
  }
}
```

Reading Streams

```
byte[] buffer = new byte[bufferSize]; bool completed = false;
do
{
  int nread = stream.Read(buffer, 0, BUFFERSIZE);
  if (nread == 0) completed = true; if (nread < BUFFERSIZE)
  {
    Array.Clear(buffer, nread, BUFFERSIZE - nread); }
    string s = encoding.GetString(buffer, 0, nread);
    Console.WriteLine($"read {nread} bytes");
    Console.WriteLine(s);
} while (!completed);
```

Writing Streams

```
string textFileName = ”textfile.txt";
using (FileStream stream = File.OpenWrite(tempTextFileName))
{
  string hello = "Hello, World!";
  byte[] buffer = Encoding.UTF8.GetBytes(hello);
  stream.Write(buffer, 0, buffer.Length);
  Console.WriteLine($"file {stream.Name} written");
}
```

Copying Streams

```
const int BUFFERSIZE = 4096;
using (var inputStream = File.OpenRead(inputFile))
using (var outputStream = File.OpenWrite(outputFile)) {
  byte[] buffer = new byte[BUFFERSIZE];
  bool completed = false;
  do
  {
    int nRead = inputStream.Read(buffer, 0, BUFFERSIZE);
    if (nRead == 0) completed = true;
    outputStream.Write(buffer, 0, nRead);
  } while (!completed);
}
```

Copying Streams (cont.)

```
using (var inputStream = File.OpenRead(inputFile))
using (var outputStream = File.OpenWrite(outputFile)) {
  inputStream.CopyTo(outputStream);
}
```

Thanks!

Any questions?

You can find me at:

tranminhphuoc@gmail.com