Input/Output

By Pakita Shamoi,

Spring 2018

Part 1. Navigating the file system

- Inside the *System.IO* namespace are a set of classes used to navigate and manipulate files, directories, and drives
- FileInfo and DirectoryInfo classes expose all the system information about file system objects—specifically, files, directories, and drives.
- *DriveInfo* class represents a drive in the file system
- Utility classes include the *File*, *Directory*, and *Path* classes
- Some properties and methods of FI, DI:
  - CreationTime, Exists, Extension, FullName, Name, Delete,…

FileInfo

- The *FileInfo* class provides the basic functionality to access and manipulate a single file in the file system
  - Properties: Directory, Length, IsReadOnly…
  - Methods: CopyTo, MoveTo, Open, OpenRead…

```
FileInfo ourFile = new FileInfo(@"c:\a.txt");
if (ourFile.Exists)
{
Console.WriteLine("Filename : {0}", ourFile.Name);
Console.WriteLine("Path : {0}", ourFile.FullName);
ourFile.CopyTo(@"c:\a-copy.txt");
}
```

DirectoryInfo

- The *DirectoryInfo* class provides the basic functionality to access and manipulate a single directory in the file system.
  - Properties: Parent, Root…
  - Methods: Create, MoveTo, GetDirectories, GetFiles…

```
DirectoryInfo ourDir = new DirectoryInfo(@"c:\windows");
Console.WriteLine("Directory: {0}", ourDir.FullName);
foreach (FileInfo file in ourDir.GetFiles())
{
       Console.WriteLine("File: {0}", file.Name);
}
```

DriveInfo, DriveType

- Properties: DriveFormat, IsReady, TotalFreeSpace…
- 1 static method – GetDrives()
- DriveType – enumeration (CDRom, Fixed, Removable…)

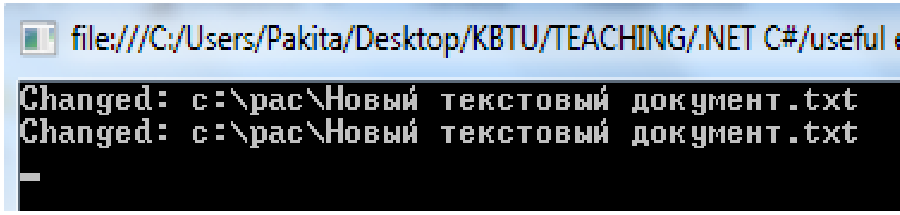
```
DriveInfo[] drives = DriveInfo.GetDrives();
  foreach (DriveInfo drive in drives)
    {
      Console.WriteLine("Drive: {0}", drive.Name);
      Console.WriteLine("Type: {0}", drive.DriveType);
     }
```

Path class

- The Path class provides methods for manipulating a file system path
  - Methods – GetExtension, GetFileName, GetFullPath,…

```
        string ourPath = @"c:\b.txt";
        Console.WriteLine(ourPath);
        Console.WriteLine("Ext: {0}",
Path.GetExtension(ourPath));
Console.WriteLine(Path.ChangeExtension(ourPath, "doc")+"");
```

FileSystemWatcher



file:///C:/Users/Pakita/Desktop/KBTU/TEACHING/.NET C#/useful

```
Changed: c:\pac\Новый текстовый документ.txt
Changed: c:\pac\Новый текстовый документ.txt
_
```

- The *FileSystemWatcher* class provides methods for monitoring file system directories for changes
  - Properties: EnableRaisingEvents, Filter, Path…
  - WaitForChanged method
  - Events: Changed, Created, Deleted, Renamed

```
FileSystemWatcher watcher = new FileSystemWatcher(@"c:\");
watcher.Filter = "*.txt";
watcher.IncludeSubdirectories = true;
watcher.NotifyFilter = NotifyFilters.Size | NotifyFilters.Attributes;
watcher.Changed += new FileSystemEventHandler(watcher_Changed);
watcher.EnableRaisingEvents = true;
```

```
static void watcher_Changed(object sender, FileSystemEventArgs e){
  Console.WriteLine("Changed: {0}", e.FullPath);
}
```

Part 1 - Summary

- The FileInfo, DirectoryInfo, and DriveInfo classes can be used to enumerate and inspect the properties of file system objects
- The Path class can be used to interrogate a file system path and should be used instead of parsing the string manually.
- The FileSystemWatcher class can be used to monitor the file system for changes

  such as additions, deletions, and renamings.

Part 2. Reading and Writing Files

- Abstract class Stream
  - Properties: CanRead, CanRead, Length, Position, …
  - Methods: WriteByte, ReadByte, Seek, Close…
- File class
  - Read/write a file
  - Create/ open files
  - Some simpe operations- File.Exists, File.Delete
  - Static methods: AppendText, Cope, Create, Move, OpenRead, OpenWrite, …
- Directory class - for manipulating and creating directories in the file system.

FileAccess, FileMode Enums

- FileAccess – Read, Write, ReadWrite
- FileMode - Append, Create, Open, CreateOpen, Truncate,….

Reading from file

- Opening a file involves asking the *File* class to open a stream by specifying the path to the file.
- When opening a file to read its contents, you use the *FileMode.Open* enumeration member to specify an existing file, as well as *FileAccess.Read* to get read-only access to the file:

```
FileStream theFile =
File.Open(@"C:\a.txt", FileMode.Open, FileAccess.Read);
StreamReader rdr = new StreamReader(theFile);
Console.Write(rdr.ReadToEnd());
rdr.Close();
theFile.Close();
```

- Reading the file in a single method call:

```
Console.WriteLine(File.ReadAllText(@"C:\boot.ini"));
```

Writing to file

- Open file for writing

```
FileStream theFile =File.Create(@"c:\somefile.txt");
StreamWriter writer = new StreamWriter(theFile);
writer.WriteLine("Hello");
```

- Write text directly into your new file.

```
 StreamWriter writer = File.CreateText(@"c:\somefile.txt"
writer.WriteLine("Hello");
```

// write all text at once

```
File.WriteAllText(@"c:\somefile.txt", "Hello");
```

Writing to an existing file

```
FileStream theFile = null;
theFile = File.Open(@"c:\somefile.txt",FileMode.Open, FileAccess.Write);
```

- Or just write:

```
theFile = File.OpenWrite(@"c:\somefile.txt");
```

- You can use the Open method of the File class to specify that you want to open or create a file

```
theFile = File.Open(@"c:\somefile.txt", FileMode.OpenOrCreate,
FileAccess.Write);
```

- Write to and read from in-memory strings using StringReader and StringWriter

Memory Stream

- Often you will need to create a stream before you really need to store it somewhere (like in a file).
- The MemoryStream class has the job of helping you create streams in memory.

```
MemoryStream memStrm = new MemoryStream();
StreamWriter writer = new StreamWriter(memStrm);
writer.WriteLine("Hello");
// Force the writer to push the data
writer.Flush();
// Create a file stream
FileStream theFile = File.Create(@"c:\inmemory.txt");
// Write the entire Memory stream to the file
memStrm.WriteTo(theFile);
```

Buffered Streams

- Used to wrap streams to improve performance by buffering reads and writes through the stream

1. Create a new *FileStream* object, using the *File* class to specify a new file.
2. Create a new buffered stream, specifying the file stream as the underlying stream.
3. Use a *StreamWriter* to write data into the buffered stream.

```
FileStream newFile = File.Create(@"c:\test.txt");
BufferedStream buffered = new BufferedStream(newFile);
StreamWriter writer = new StreamWriter(buffered);
writer.WriteLine("Some data");
writer.Close();
```

Part 2 - Summary

- The *File* class can be used to open files, create new files, read whole files atomically, and even write files.
- The *FileStream* class represents a file in the file system and allows for reading and writing (depending on how it is created).
- The *StreamReader* and *StreamWriter* classes are used to simplify the writing of strings to streams.
- The *MemoryStream* is a specialized stream for creating content in memory and supports saving the stream to other streams.

Part 3 – Compressing streams

- Two methods for compressing data: **GZIP** and **DEFLATE**.
- Both of these compression methods are industry-standard compression algorithms that are also free of patent protection.
- Limit is 4gb.
- Two corresponding streams are **GZipStream** and **DeflateStream**

Compressing data

- Open the file to be compressed and the file you are going to write to:

```
FileStream sourceFile = File.OpenRead("a.txt");
FileStream destFile = File.Create("azip.txt");
```

- Create GZipStream object and specify the destination stream

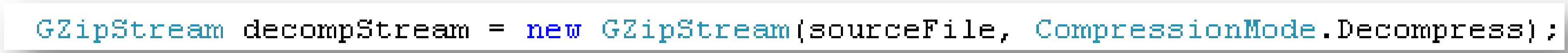
```
GZipStream compStream  = new GZipStream(destFile, CompressionMode.Compress);
```

- Then read data from the source stream and feed it into the compression stream

```
int theByte = sourceFile.ReadByte();
while (theByte != -1)
{
    compStream.WriteByte((byte)theByte);
    theByte = sourceFile.ReadByte();
}
```

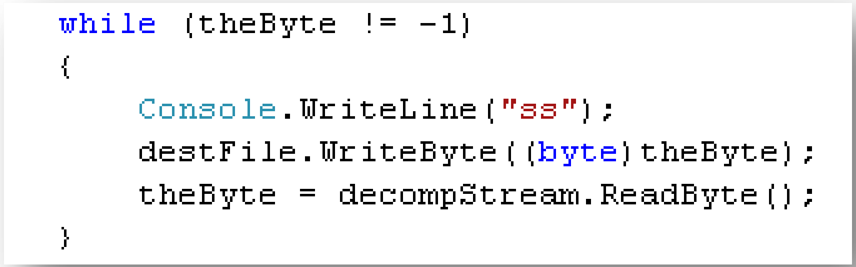
Decompressing data

- In this case, the source file is a compressed file and the destination file is going to be written as a decompressed file



```
GZipStream decompStream = new GZipStream(sourceFile, CompressionMode.Decompress);
```

- Now you need to read from the decompression stream instead of from the source file and write out to the file directly



```
while (theByte != -1)
{
    Console.WriteLine("ss");
    destFile.WriteByte((byte)theByte);
    theByte = decompStream.ReadByte();
}
```