# File Handling in C#

فایل کالکشنی از داده ها است که بر روی دیسک با یک نام بخصوص و پسوند و مسیر ذخیره می شود.

زمانی که یک فایل را در c# باز می کنید و می خواهید بخوانید یا بنویسید به stream تبدیل می شود.

##### **What is Stream?**

دنباله ای از byte ها است sequence of bytes که از یک source یا منبع به یک destination یا مقصد منتقل می شود.

2 main stream

**Input** برای خواندن دیتا کاربرد دارد.

Output برای نوشتن در یک فایل کاربرد دارد.

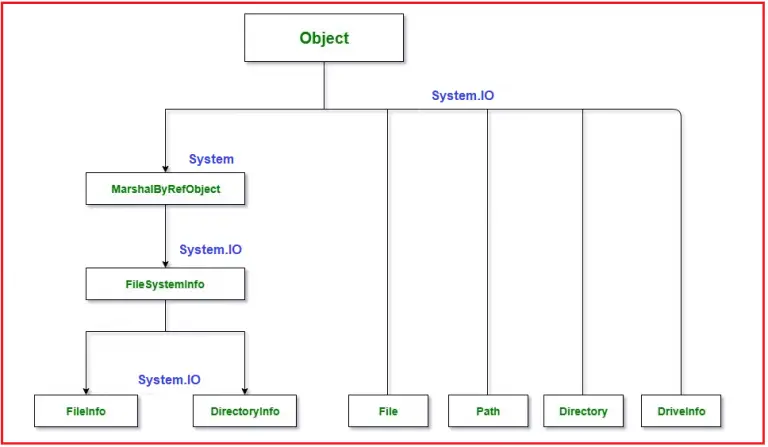
بنابراین می توان گفت که stream دنباله ای byte ها است که برای ارتباط با فایل ها کاربرد دارد. و زمانی که فایلی را برای خواندن یا نوشتن باز می کنید به stream تبدیل می شود.

عموما داده ها را به صورت file ذخیره می کنیم.

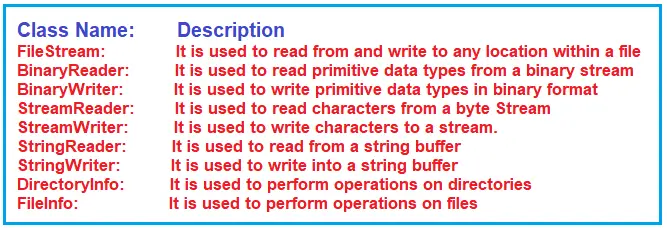
Reading عملیات خوانده داده ها از فایل را شامل می شود.

Writing عملیات نشتن داده ها در فایل را شامل می شود.

Appending مشابه عملیات نوشتن است با این تفاوت که دیتا به دیتا های قبلی فایل اضافه می شود و overwrite نمی شود.

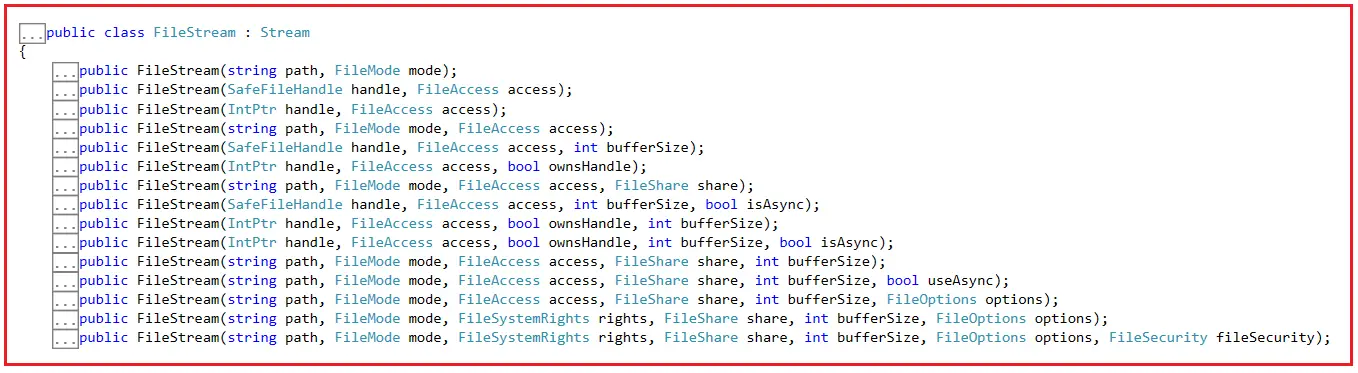


اکثرا از کلاس stream ارث بری کرده اند که یک کلاس abstract است. مثلا fileStream از آن ارث برده و پیاده سازی



# FileStream Class in C#

کلاس FileStream ارائه دهنده stream برای انجام عملیات بر روی file است. و به صورت sync , async است.



**public FileStream(string path, FileMode mode)**

**mode**

**مشخص کننده نحوه باز کردن یا ایجاد فایل است.**

**FileAccess**

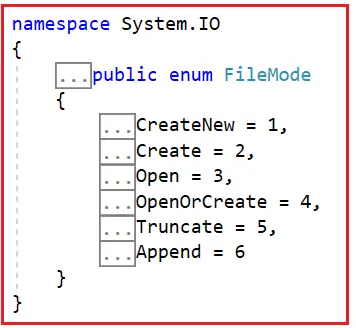
**مشخص می کند که فایل چگونه توسط stream می تواند دسترسی یابد. مشخص می کند مقادیر پروپرتی های canRead, CanWrite را.**

**Share**

**مشخص می کند فایل چگونه توسط process می تواند share شود.**

**Enums – FileMode**

**مشخص می کند که os چطور باید فایل را باز کند.**



1. **CreateNew**: It specifies that the operating system should create a new file. This requires a System.Security.Permissions.FileIOPermissionAccess.Write permission. If the file already exists, a System.IO.IOException exception is thrown.
2. **Create**: It specifies that the operating system should create a new file like the CreateNew constant. But in this case, if the file already exists, it will be overwritten instead of throwing an Exception. This also requires System.Security.Permissions.FileIOPermissionAccess.Write permission. So, FileMode.Create is equivalent to requesting that if the file does not exist, use System.IO.FileMode.CreateNew; otherwise, use System.IO.FileMode.Truncate. If the file already exists but is a hidden file, then an UnauthorizedAccessException Exception is going to be thrown.
3. **Open**: It specifies that the operating system should open an existing file. The ability to open the file is dependent on the value specified by the System.IO.FileAccess Enumeration. A System.IO.FileNotFoundException exception is thrown if the file does not exist.
4. **OpenOrCreate**: It specifies that the operating system should open a file if it exists; otherwise, a new file should be created. If the file is opened with FileAccess.Read, System.Security.Permissions.FileIOPermissionAccess.Read permission is required. If the file access is FileAccess.Write, System.Security.Permissions.FileIOPermissionAccess.Write permission is required. If the file is opened with FileAccess.ReadWrite, both Systems.Security.Permissions.FileIOPermissionAccess.Read and System.Security.Permissions.FileIOPermissionAccess.Write permissions are required.
5. **Truncate**: It specifies that the operating system should open an existing file. When the file is opened, it should be truncated so that its size is zero bytes. This requires a System.Security.Permissions.FileIOPermissionAccess.Write permission. Attempts to read from a file opened with FileMode.Truncate causes a System.ArgumentException exception.
6. **Append**: It opens the file if it exists and then adds the content at the end of the file, or creates a new file. This requires a System.Security.Permissions.FileIOPermissionAccess.Append permission. FileMode.Append can be used only in conjunction with FileAccess.Write. Trying to seek a position before the end of the file throws a System.IO.IOException exception, and any attempt to read fails and throws a System.NotSupportedException exception.

##### **FileAccess**

جهت دادن دسترسی read write , read/write به فایل کاربرد دارد.

1. **Read**– It gives read access to the file. Data can be read from the file. Combine with Write for read/write access.
2. **Write** – It gives Write access to the file. Data can be written to the file. Combine with Read for read/write access.
3. **ReadWrite** – It gives read and writes access to the file. Data can be written to and read from the file.

##### **FileShare**

مشخص کننده این است که زمانی که به یک فایل یک stream دسترسی دارد و هم زمان یک stream دیگر می خواهد به آن فایل دسترسی داشته باشد چه اتفاقی رخ دهد.

1. **None**: Declines sharing of the current file. Any request to open the file (by this process or another process) will fail until the file is closed.
2. **Read**: Allows subsequent opening of the file for reading. If this flag is not specified, any request to open the file for reading (by this process or another process) will fail until the file is closed. However, even if this flag is specified, additional permissions might still be needed to access the file.
3. **Write**: Allows subsequent opening of the file for writing. If this flag is not specified, any request to open the file for writing (by this process or another process) will fail until the file is closed. However, even if this flag is specified, additional permissions might still be needed to access the file.
4. **ReadWrite**: Allows subsequent opening of the file for reading or writing. If this flag is not specified, any request to open the file for reading or writing (by this process or another process) will fail until the file is closed. However, even if this flag is specified, additional permissions might still be needed to access the file.
5. **Delete**: Allows subsequent deleting of a file.
6. **Inheritable**: Makes the file handle inheritable by child processes. This is not directly supported by Win32.

**نکته : به بستن stream ها توجه ویژه شود**

**مثال ایجاد فایل**

//Set the File Path where you want to Create the File in your Disk

string FilePath = @"D:\MyFile.txt";

//Create an Instance of FileStream Class By specifying the File Path and File Mode

//FileMode is going to be Create as we are going to create a New File

FileStream fileStream = new FileStream(FilePath, FileMode.Create);

//Finally close the fileStream Object

fileStream.Close();

خواندن فایل

//Set the File Path where your File is Exist

string FilePath = @"D:\MyFile.txt";

//Create an Instance of FileStream Class By specifying the File Path and File Mode

//FileMode is going to be Append as we are going to append some text in the file

FileStream fileStream = new FileStream(FilePath, FileMode.Append);

//Create the byte array which contains the string data to be appended in the File

byte[] bytedata = Encoding.Default.GetBytes("C# Is an Object Oriented Programming Language");

//Write the Byte Array into the File Stream Object using the Write Method

//array (bytedata): The buffer containing data to write to the stream.

//offset (0): The zero-based byte offset in the array from which to begin copying bytes to the stream.

//count (bytedata.Length): The maximum number of bytes to write.

fileStream.Write(bytedata, 0, bytedata.Length);

//Finally close the fileStream Object

fileStream.Close();

زمانی که می خواهید اطلاعات file را بخوانید لازم است که ابتدا فایل را با fileStream بگیرید و سپس آن را به streamReader بدهید و داده را بخوانید ازش در حالتی که می خواهید در فایل بنویسیم لازم است که از متد write و داده ها به صورت byte باشند تا نوشته شوند.

# StreamReader and StreamWriter in C#

##### **StreamWriter Class in C#**

برای نوشتن text data در فایل کاربرد دارد. کلاس streamWriter از کلاس TextWriter که abstract است ارث بری کرده است.

این کلاس دارای تعداد زیادی ctor است که overload شده اند و مجموعه ی کاملی از متد های مورد نیاز را ارائه می دهد.

##### **Methods:**

1. **Close():** This method closes the current StreamWriter object and the underlying stream.
2. **Flush():** This method Clears data from all buffers for the current writer and causes any buffered data to be written to the underlying stream.
3. **Write():** It Writes data to the stream. It has different overloads for different data types to write in the stream.
4. **WriteLine:** It is the same as Write() but it adds the newline character at the end of the data. It has different overloads for different data types to write in the stream.
5. **Dispose():** It releases the unmanaged resources used by the StreamWriter and optionally releases the managed resources.

##### **Properties:**

1. **AutoFlush:** Gets or sets a value indicating whether the StreamWriter will flush its buffer to the underlying stream after every call to System.IO.StreamWriter.Write(System.Char).
2. **BaseStream:** Gets the underlying stream that interfaces with a backing store.
3. **Encoding:** Gets the System.Text.Encoding in which the output is written.

مثال

ایجاد یک فایل و نوشتن ورودی کاربر در فایل.

static void Main(string[] args)

{

// Create an Instance of StreamWriter by specifying the String Path

// This will create a file with the name MyFile.txt at the specified location i.e. in the D Drive

// Here we are using the StreamWriter constructor which takes the string path as an argument to create an instance of StreamWriter class

StreamWriter streamWriter = new StreamWriter("D://MyFile.txt");

// Asking the user to enter the text that we want to write into the MyFile.txt file

Console.WriteLine("Enter the Text that you want to write on File");

// To read the input from the user

string inputData = Console.ReadLine();

// To write the data into the stream use the Write Method of the StreamWriter Object

streamWriter.Write(inputData);

Console.WriteLine("Data Has Been Written to the File");

// Clears all the buffers for the current writer by calling the Flush Method of the StreamWriter Object

streamWriter.Flush();

// Close the current StreamWriter object and the underlying stream by calling the Flush Method of the StreamWriter Object

streamWriter.Close();

Console.ReadKey();

}

}

##### **StreamReader Class in C#**

برای خواندن فایل Text کاربرد دارد. کلاس TextWriter را که abstract است پیاده سازی کرده است.

به صورت پیش فرض از utf-8 استفاده می کند. و به صورت پیشفرض thread safe نیست.

متد ctor آن دارای overload های مختلفی است که با توجه به نیاز می توانیم از آن استفاده کنیم.

##### **Methods:**

1. **Close():** The Close method Closes the StreamReader object and the underlying stream, and releases any system resources associated with the reader.
2. **Peek():** This method returns the next available character but does not consume it. An integer represents the next character to be read, or -1 if there are no characters to be read or if the stream does not support seeking.
3. **Read():** This method reads the next character from the input stream and advances the character’s position by one character. The next character from the input stream is represented as a System.Int32 object, or -1 if no more characters are available.
4. **ReadLine():** This method Reads a line of characters from the current stream and returns the data as a string. The next line from the input stream, or null if the end of the input stream is reached.
5. **Seek():** It is used to read/write at a specific location from a file.

##### **Properties:**

1. **CurrentEncoding:** It gets the current character encoding that the current System.IO.StreamReader object is using.
2. **EndOfStream:** It gets a value that indicates whether the current stream position is at the end of the stream.
3. **BaseStream:** It returns the underlying stream.

مثال استفاده از متد ها پروپرتی های streamReader

static void Main(string[] args)

{

//Create a Variable to Hold the String Path

string filePath = "D://MyFile.txt";

//Creating an Instance StreamReader Object to Read the Data from the File Path

StreamReader streamReader = new StreamReader(filePath);

Console.WriteLine("Content of the File");

// This is used to specify from where to start reading the input stream

// BaseStream: Returns the underlying stream.

// Seek: The new position within the current stream.

// SeekOrigin.Begi: Specifies the beginning of a stream

streamReader.BaseStream.Seek(0, SeekOrigin.Begin);

//It reads a line of characters from the current stream and returns the data as a string.

//It return the next line from the input stream, or null if the end of the input stream is reached.

string strData = streamReader.ReadLine();

// To Read the whole file line by line use While Loop as long the strData is not null

while (strData != null)

{

//Print the String data

Console.WriteLine(strData);

//Then Read the next String data

strData = streamReader.ReadLine();

}

Console.ReadLine();

//Close the streamReader Object

streamReader.Close();

Console.ReadKey();

}

به جای خط به خط خواندن می توان از متد زیر نیز استفاده کرد

using (StreamReader reader = new StreamReader(filePath))

{

Console.WriteLine(reader.ReadToEnd());

}

# File Class in C#

این کلاس تعدادی متد static را ارائه می دهد که برای عملیات های از قبیل creating file , copying , moving , deleting , و کارکردن با fileStream برای خواندن و نوشتن فایل کاربرد دارد.

**File Class Methods**

1. **Copy**: This method is used to Copy an existing file to a new file. Overwriting a file of the same name is not allowed.
2. **Create**: This method creates or overwrites it in the specified path.
3. **Decrypt**: This method is used to Decrypt a file encrypted by the current account using the System.IO.File.Encrypt(System.String) method.
4. **Delete**: This method is used to delete the specified file.
5. **Encrypt**: This method is used to encrypt a file so that only the account used to encrypt the file can decrypt it.
6. **Open**: This method is used to Open a System.IO.FileStream on the specified path, having the specified mode with reading, write, or read/write access and the specified sharing option.
7. **Move**: This method is used to Move a specified file to a new location, providing the option to specify a new file name.
8. **Exists**: This method is used to determine whether the specified file exists.
9. **OpenRead**: This method is used to open an existing file for reading.
10. **OpenText**: This method opens an existing UTF-8 encoded text file for reading.
11. **OpenWrite**: This method is used to open an existing file or create a new file for writing.
12. **ReadAllBytes**: This method is used to open a binary file, read the file’s contents into a byte array, and then close the file.
13. **ReadAllLines**: This method is used to Open a file, read all lines of the file with the specified encoding, and then close the file.
14. **ReadAllText**: This method is used to Open a text file, read all the text in the file, and then close the file.
15. **ReadLines**: This method is used to read the lines of a file.
16. **Replace**: This method is used to replace a specified file’s contents with another file’s contents, delete the original file, and create a backup of the replaced file.
17. **WriteAllBytes**: This method is used to create a new file, write the specified byte array to the file, and then close the file. If the target file already exists, it is overwritten.
18. **WriteAllLines**: This method is used to create a new file, write the specified string array to the file, and then close the file.
19. **WriteAllText**: This method is used to create a new file, write the specified string to the file, and then close the file. If the target file already exists, it is overwritten.

**public static void Copy(string sourceFileName, string destFileName, bool overwrite);**

**پارامتر آخر بیانگر این است که در صورتی که فایل وجود داشته باشد آن را overwrite کند و خطا ندهد.**

##### **Create Method of File Class in C#**

نکته: تمامی overload های مربوط به متد Create به عنوان return type خود instance of fileStream را برمگیردانند پس بنابراین لازم است که حتما **Call Close Mehtod Of Stream**

static void Main(string[] args)

{

//Set the File Path where you want to create the File

string FilePath = @"D:\MyNewFile.txt";

//Creating the File using the static Create method of the File class

//To the Create Method, we need to pass the FilePath where we want to Create the File

//The Create Method will return the FileStream object

FileStream fileStream = File.Create(FilePath);

//Closing the FileStream Object immediately once the File is Created

fileStream.Close();

//Check if the FilePath Exists using the Exists Method of the File Class

if (File.Exists(FilePath))

{

//If the File Exists, Write Content to the File

//Creating a string array to Hold the Data

string[] content = { "Hello", "And", "Welcome" };

//using the WriteAllLines Method, we are writing the data

File.WriteAllLines(FilePath, content);

Console.WriteLine("MyNewFile.txt File Created with the Following Data");

//Reading the data from the File using ReadAllText Method

string fileContent = File.ReadAllText(FilePath);

//Printing the Data in the Console

Console.WriteLine(fileContent);

}

else

{

Console.WriteLine("MyNewFile.txt File Does Not Exist in D Directory");

}

Console.ReadKey();

}

# TextWriter and TextReader in C#

کلاسی برای خواندن و نوشتن read , write به ترتیب است. حتی اگر این کلاس ها stream نباشند. این دو کلاس base کلاس هستند.

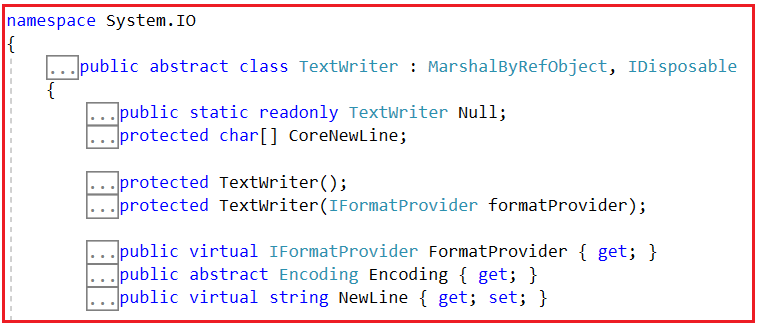
2 کلاس streamReader , stringReader از کلاس abstract textReader مشتق شده اند یا drive.

به همین ترتیب کلاس streamWriter , stringWriter نیز از کلاس abstract textWriter مشتق شده اند.

##### **What is TextWriter Class in C#?**

کلاس است در سی شارپ که قابلیت نوشتن کاراکتر ها به صورت ترتیبی را فراهم می کند. از این کلاس برای نوشتن در فایل Text می توان استفاده کرد.

این کلاس base abstract مربوط به کلاس های streamWriter , stringWriter است که برای نوشتن کاراکتر ها در stream و string کاربد دارد و در فایل.



##### **Methods of TextWriter class in C#:**

1. **Synchronized(TextWriter)**: It is used to Create a thread-safe wrapper around the specified TextWriter.
2. **Close():** It Closes the current writer and releases any system resources associated with the writer.
3. **Dispose():** It releases all resources used by the System.IO.TextWriter object.
4. **Flush():** It Clears all buffers for the current writer and causes any buffered data to be written to the underlying device.
5. **Write(Char):** It is used to write a character to the text stream.
6. **Write(String):** It is used to write the string to the text stream.
7. **WriteAsync(Char):** It is used to write the character to the text stream asynchronously.
8. **WriteLine():** It is used to write a line terminator to the text stream.
9. **WriteLineAsync(String):** It is used to write the string to the text stream asynchronously followed by a line terminator.

کلاس TextWriterیک کلاس abstract است که برای نوشتن سریالی ترتیبی از کاراکتر ها در فایل کاربرد دارد. همچنین کلاس base مربوط به streamWriter, stringWriter استکه برای نوشتن کاراکتر ها در string کاربرد دارد و به صورت پیشفرض threadSafe نیست.

نحوه کارکرد:

از انجایی که این کلاس abstract است برای رسیدن به textWriter نیاز است که از متد CreateText استفاده کنیم از کلاس file

اگر فایل وجود داشته باشد اوررایت می کند و اگر نه می سازد

**static** **void** Main**(string[]** args**)**

**{**

**string** filePath = @"D:\MyFile1.txt";

**using** **(**TextWriter textWriter = File.CreateText**(**filePath**))**

**{**

textWriter.WriteLine**(**"Hello TextWriter Abstract Class!"**)**;

textWriter.WriteLine**(**"File Handling Tutorial in C#"**)**;

**}**

Console.WriteLine**(**"Write Successful"**)**;

Console.ReadKey**()**;

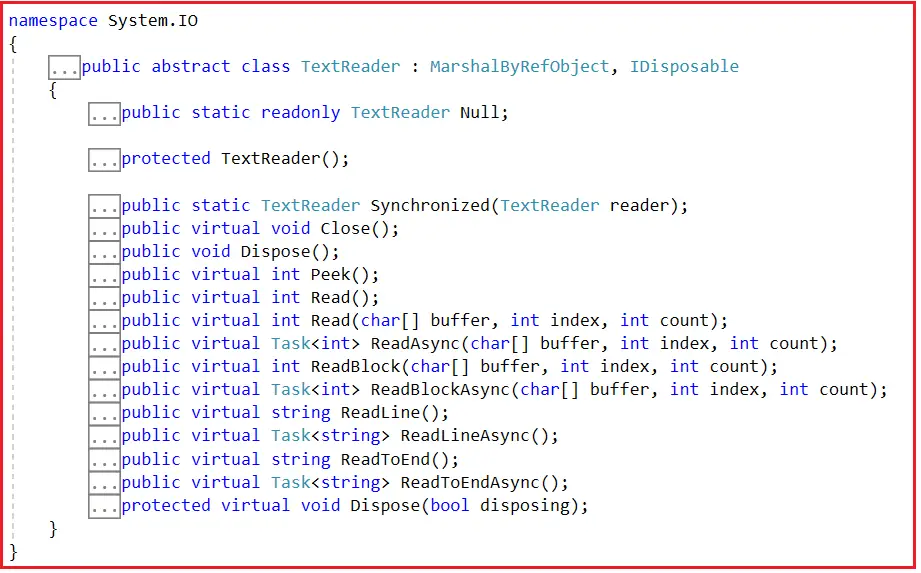
**}**

##### **What is TextReader class in C#?**

این کلاس برای خواندن text به صورت ترتیبی در سریالی از کاراکتر ها کاربرد دارد از یک فایل text. این کلاس یک کلاس abstract است که قابلیت ساخت instance ندارد و به عنوان کلاس base کلاس های streamReader , stringReader است که برای خواندن کاراتر ها از stream , string کاربرد دارد.

##### **Methods of TextReader Class in C#:**

1. **Synchronized():** It is used to create a thread-safe wrapper around the specified TextReader.
2. **Close():** It is used to close the TextReader and release any system resources associated with the TextReader.
3. **Dispose():** It is used to release all resources used by the TextReader object.
4. **Peek():** It is used to read the next character without changing the state of the reader or the character source. Returns the next available character without actually reading it from the reader. It returns an integer representing the next character to be read, or -1 if no more characters are available or the reader does not support seeking.
5. **Read():** It is used to read the next character from the text reader and advances the character’s position by one character. It returns the next character from the text reader, or -1 if no more characters are available. The default implementation returns -1.
6. **ReadLine():** It is used to read a line of characters from the text reader and returns the data as a string. It returns the next line from the reader, or null if all characters have been read.
7. **ReadToEnd():** It is used to read all characters from the current position to the end of the text reader and returns them as one string. That means it reruns a string that contains all characters from the current position to the end of the text reader.



نحوه کارکرد:

برای رسیدن به TextReader به دلیل این که abstract است نمی توانیم instance بسازیم راه حل استفاده از کلاس File و متد OpenText است.

**using(TextReader textReader = File.OpenText(filePath))**  
**{**  
**//Code**  
**}**

**string** filePath = @"D:\MyFile1.txt";

//Read One Line

**using** **(**TextReader textReader = File.OpenText**(**filePath**))**

**{**

Console.WriteLine**(**textReader.ReadLine**())**;

**}**

//Read 4 Characters

**using** **(**TextReader textReader = File.OpenText**(**filePath**))**

**{**

**char[]** ch = new **char[**4**]**;

textReader.ReadBlock**(**ch, 0, 4**)**;

Console.WriteLine**(**ch**)**;

**}**

the //Read full file

**using** **(**TextReader textReader = File.OpenText**(**filePath**))**

**{**

Console.WriteLine**(**textReader.ReadToEnd**())**;

**}**

Console.ReadKey**()**;

# BinaryWriter and BinaryReader in C#

# این کلاس برای نوشتن primitive data types مثل int char در فرمت باینری در یک Stream کاربرد دارد.

# فایل باینری که ایجاد می کند قابل فهم توسط انسان نیست اما ماشین آن را می فهمد.

# قابلیت نوشتن string در specific encoding را نیز دارد.

برای ایحاد binaryWriter لازم است که به ctor آن یک Stream را بدهیم.

##### **Methods of BinaryWriter Class in C#:**

1. **Write(String):** This method is used to write a length-prefixed string to this stream in the current encoding of the BinaryWriter and advances the current position of the stream in accordance with the encoding used and the specific characters being written to the stream.
2. **Write(float):** This method is used to write a four-byte floating-point value to the current stream and advances the stream position by four bytes.
3. **Write(long):** This method is used to write an eight-byte signed integer to the current stream and advances the stream position by eight bytes.
4. **Write(Boolean):** This method is used to write the one-byte Boolean value to the present stream; 0 represents false while 1 represents true.
5. **Write(Byte):** This method is used to write an unsigned byte to the present stream and then it advances the position of the stream by one byte.
6. **Write(Char):** This method is used to write Unicode characters to the present stream and also it advances the present stream position according to the character encoding used and according to the characters being written to the present stream.
7. **Write(Decimal):** This method is used to write a decimal value to the present stream and also it advances the position of the current stream by sixteen bytes.
8. **Write(Double):** This method is used to write an eight-byte floating-point value to the present stream and then it also advances the position of the current stream by eight bytes.
9. **Write(Int32):** This method is used to write a four-byte signed integer to the present stream and then it advances the position of the current stream by four bytes.

##### **How to Create an Instance of BinaryWriter Class in C#?**

There are four overloaded constructors available in BinaryWriter class to create a BinaryWriter instance. They are as follows:

1. **public BinaryWriter(Stream output):** It initializes a new instance of the BinaryWriter class based on the specified stream and uses UTF-8 encoding. Here, the parameter output specifies the output stream.
2. **public BinaryWriter(Stream output, Encoding encoding):** It initializes a new instance of the BinaryWriter class based on the specified stream and character encoding. Here, the parameter output specifies the output stream and the parameter encoding specifies the character encoding to use.
3. **public BinaryWriter(Stream output, Encoding encoding, bool leaveOpen):** It initializes a new instance of the BinaryWriter class based on the specified stream and character encoding, and optionally leaves the stream open. Here, the parameter output specifies the output stream and the parameter encoding specifies the character encoding to use and the parameter leaveOpen specifies true to leave the stream open after the BinaryWriter object is disposed otherwise, false.
4. **protected BinaryWriter():** It initializes a new instance of the System.IO.BinaryWriter class that writes to a stream.

بهتر است استفاده از این کلاس را در بلاک using انجام دهیم تا پس از پایان بلاک تمامی منابع اشغال شده آزاد شوند.

**using** **(**BinaryWriter writer = new BinaryWriter**(**File.Open**(**"D:\\MyBinaryFile.bin", FileMode.Create**)))**

**{**

//Writting Error Log

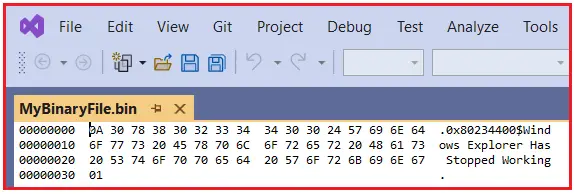
writer.Write**(**"0x80234400"**)**;

writer.Write**(**"Windows Explorer Has Stopped Working"**)**;

writer.Write**(true)**;

**}**

فایل خروجی به صورت زیر خواهد بود



در حالت عادی غیر قابل فهم است که اگر با استفاده از BinaryReader خوانده شود به فایل قابل فهم تبدیل می شود.

مزایای اصلی : عدم فهم توسط انسان و بهترین روش برای استفاده از فضا است.

Extension = > .bin

برای خواندن فایل binary باید از کلاس binaryReader استفاده کرد.

1. **Read():** It is used to read characters from the underlying stream and advances the current position of the stream in accordance with the Encoding used and the specific character being read from the stream. It returns the next character from the input stream, or -1 if no characters are currently available.
2. **ReadBoolean():** It is used to read a Boolean value from the current stream and advances the current position of the stream by one byte. It returns true if the byte is nonzero; otherwise, false.
3. **ReadByte():** It is used to read the next byte from the current stream and advances the current position of the stream by one byte. It returns the next byte read from the current stream.
4. **ReadChar():** It is used to read the next character from the current stream and advances the current position of the stream in accordance with the Encoding used and the specific character being read from the stream. It returns a character read from the current stream.
5. **ReadDecimal()**: It is used to read a decimal value from the current stream and advances the current position of the stream by sixteen bytes. It returns a decimal value read from the current stream.
6. **ReadDouble():** It is used to read an 8-byte floating-point value from the current stream and advances the current position of the stream by eight bytes. It returns an 8-byte floating-point value read from the current stream.
7. **ReadInt32():** It is used to read a 4-byte signed integer from the current stream and advances the current position of the stream by four bytes. It returns a 4-byte signed integer read from the current stream.
8. **ReadInt64():** It is used to read an 8-byte signed integer from the current stream and advances the current position of the stream by four bytes. It returns an 8-byte signed integer read from the current stream.
9. **ReadString():** It is used to read a string from the current stream. The string is prefixed with the length, encoded as an integer seven bits at a time. It returns the string being read.

##### **How to Create an Instance of BinaryReader Class in C#?**

1. **public BinaryReader(Stream input):** It initializes a new instance of the System.IO.BinaryReader class based on the specified stream and using UTF-8 encoding. Here, the parameter input specifies the input stream.
2. **public BinaryReader(Stream input, Encoding encoding):** It initializes a new instance of the System.IO.BinaryReader class based on the specified stream and character encoding. Here, the parameter input specifies the input stream and the parameter encoding specifies the character encoding to use.
3. **public BinaryReader(Stream input, Encoding encoding, bool leaveOpen):** It initializes a new instance of the System.IO.BinaryReader class based on the specified stream and character encoding, and optionally leaves the stream open. Here, the parameter input specifies the input stream and the parameter encoding specifies the character encoding to use and the parameter leaveOpen specifies true to leave the stream open after the BinaryReader object is disposed otherwise, false.

**static** **void** WriteDataToBinaryFile**()**

**{**

**using** **(**BinaryWriter writer = new BinaryWriter**(**File.Open**(**"D:\\MyBinaryFile2.bin", FileMode.Create**)))**

**{**

//Writting Error Log

writer.Write**(**"0x80234400"**)**;

writer.Write**(**"Windows Explorer Has Stopped Working"**)**;

writer.Write**(true)**;

**}**

**}**

**static** **void** ReadDataFromBinaryFile**()**

**{**

**using** **(**BinaryReader reader = new BinaryReader**(**File.Open**(**"D:\\MyBinaryFile2.bin", FileMode.Open**)))**

**{**

Console.WriteLine**(**"Error Code : " + reader.ReadString**())**;

Console.WriteLine**(**"Message : " + reader.ReadString**())**;

Console.WriteLine**(**"Restart Explorer : " + reader.ReadBoolean**())**;

**}**

**}**

نکته مهم : این کلاس فقط برای خواندن و نوشتن primitiveDataType ها کاربرد دارد.

# StringWriter and StringReader in C#

این کلاس از کلاس TextWriter مشتق شده است یعنی drived است. کاربرد اصلی آن برای ایجاد و تغییر در string ها می باشد به دلیل این که string از نوع immutable است و غیر قابل تغییر است و با هر تغییر باعث ایجاد فضای جدید در حافظه می شود و از این کلاس می توان استفاده کرد که یک stringBuilder تحویل می دهد.

##### **Constructors of StringWriter in C#**

1. **public StringWriter():** It initializes a new instance of the System.IO.StringWriter class.
2. **public StringWriter(IFormatProvider formatProvider):** It initializes a new instance of the StringWriter class with the specified format control. The parameter formatProvider specifies a System.IFormatProvider object that controls formatting
3. **public StringWriter(StringBuilder sb):** It initializes a new instance of the StringWriter class that writes to the specified System.Text.StringBuilder. The parameter sb specifies the StringBuilder object to write to.
4. **public StringWriter(StringBuilder sb, IFormatProvider formatProvider):** It initializes a new instance of the StringWriter class that writes to the specified StringBuilder and has the specified format provider. The parameter sb specifies the StringBuilder object to write to and the parameter formatProvider specifies a System.IFormatProvider object that controls formatting.

##### **Methods of StringWriter Class in C#**

1. **Close():** This method is used to close the current StringWriter and the underlying stream.
2. **Dispose():** This method is used to release the unmanaged resources used by the System.IO.StringWriter and optionally releases the managed resources.
3. **FlushAsync():** This method is used to asynchronously clear all buffers for the current writer and causes any buffered data to be written to the underlying device.
4. **GetStringBuilder():** This method is used to return the underlying StringBuilder.
5. **ToString():** This method is used to return a string containing the characters written to the current StringWriter so far.
6. **Write(char value):** This method is used to write a character to the string.
7. **Write(string value):** This method is used to write a string to the current string.
8. **WriteAsync(char value):** This method is used to write a character to the string asynchronously.
9. **WriteAsync(string value):** This method is used to write a string to the current string asynchronously.
10. **WriteLine(String):** This method is used to Write a string followed by a line terminator to the text string or stream.
11. **WriteLineAsync(string value):** This method is used to write a string followed by a line terminator asynchronously to the current string.

مثال ایجاد stringWriter با استفاده از stringBuilder و در نهایت نمایش رشته

**static** **void** Main**(string[]** args**)**

**{**

**string** text = "Hello. This is First Line \nHello. This is Second Line \nHello. This is Third Line";

//Writing string into StringBuilder

StringBuilder stringBuilder = new StringBuilder**()**;

StringWriter stringWriter = new StringWriter**(**stringBuilder**)**;

//Store Data on StringBuilder

stringWriter.WriteLine**(**text**)**;

stringWriter.Flush**()**;

stringWriter.Close**()**;

//Read Entry

StringReader reader = new StringReader**(**stringBuilder.ToString**())**;

//Check to End of File

**while** **(**reader.Peek**()** **>** -1**)**

**{**

Console.WriteLine**(**reader.ReadLine**())**;

**}**

Console.ReadKey**()**;

**}**

##### **What is StringReader Class in C#?**

از کلاس abstract به نام textReader ارث برده است و برای خواندن string کاربرد دارد.

##### **The Constructor of StringReader Class in C#**

**public StringReader(string s):** It initializes a new instance of the StringReader class that reads from the specified string. Here, the parameter “s” specifies the string to which the StringReader should be initialized.

##### **Methods of StringReader Class in C#**

The StringReader class in C# provides the following methods.

1. **Close():** This method is used to close the StringReader.
2. **Peek():** This method is used to return the next available character but does not consume it. It returns an integer representing the next character to be read, or -1 if no more characters are available or the stream does not support seeking.
3. **Read():** This method is used to read the next character from the input string and advances the character position by one character. It returns the next character from the underlying string, or -1 if no more characters are available.
4. **ReadLine():** This method is used to read a line of characters from the current string and returns the data as a string. It returns the next line from the current string, or null if the end of the string is reached.
5. **ReadLineAsync():** This method is used to read a line of characters asynchronously from the current string and returns the data as a string. It returns a task that represents the asynchronous read operation. The value of the TResult parameter contains the next line from the string reader or is null if all the characters have been read.
6. **ReadToEnd():** This method is used to read all characters from the current position to the end of the string and returns them as a single string. It returns the content from the current position to the end of the underlying string.
7. **ReadToEndAsync():** This method is used to read all characters from the current position to the end of the string asynchronously and returns them as a single string. It returns a task that represents the asynchronous read operation. The value of the TResult parameter contains a string with the characters from the current position to the end of the string.
8. **Dispose():** This method is used to release the unmanaged resources used by the System.IO.StringReader and optionally releases the managed resources.

**static** **void** Main**(string[]** args**)**

**{**

**string** text = @"You are reading

this StringWriter and StringReader in C# article at

dotnettutorials.net";

**using** **(**StringReader rtringReader = new StringReader**(**text**))**

**{**

**int** count = 0;

**string** line;

**while** **((**line = rtringReader.ReadLine**())** != **null)**

**{**

count++;

Console.WriteLine**(**"Line {0}: {1}", count, line**)**;

**}**

**}**

Console.ReadKey**()**;

**}**

# FileInfo Class in C#

این کلاس برای کار با فایل ها کاربرد دارد عملیات manipulating مانند create , deleting , removing , copying , opening , دریافت اطلاعات فایل.

به صورت پیشفرض دسترسی کامل read , write برای تمامی user ها دارد.

File یک کلاس sealed است.

##### **The Constructor of FileInfo Class in C#**

**public FileInfo(string fileName):** It initializes a new instance of the System.IO.FileInfo class, which acts as a wrapper for a file path. The parameter fileName specifies the fully qualified name of the new file or the relative file name. Do not end the path with the directory separator character.

##### **Properties of FileInfo Class in C#**

1. **Directory**: It is used to get an instance of the parent directory. It returns a DirectoryInfo object representing the parent directory of this file.
2. **DirectoryName**: It gets a string representing the directory’s full path. It returns a string representing the directory’s full path.
3. **Length**: It is used to get the size, in bytes, of the current file. It returns the size of the current file in bytes.
4. **Name**: It is used to get the name of the file.
5. **IsReadOnly**: It is used to get or set a value determining whether the current file is read-only. It returns true if the current file is read-only; otherwise, false.
6. **Exists**: It is used to get a value indicating whether a file exists. It returns true if the file exists, false if it does not exist, or if it is a directory.

**Methods**

1. **public StreamWriter AppendText():** This method is used to get a value indicating whether a file exists. It returns true if the file exists, false if it does not exist, or if it is a directory.
2. **public FileInfo CopyTo(string destFileName):** This method is used to copy an existing file to a new one, disallowing the overwriting of an existing one. It returns a new file with a fully qualified path. The parameter destFileName specifies the name of the new file to copy to.
3. **public FileInfo CopyTo(string destFileName, bool overwrite):** This method is used to copy an existing file to a new one, allowing the overwriting of an existing one. It returns a new file or an overwrite of an existing file if overwrite is true. If the file exists and overwrite is false, an IOException is thrown. The parameter destFileName specifies the name of the new file to copy to, and the parameter overwrites specifies true to allow an existing file to be overwritten; otherwise, false.
4. **public FileStream Create():** This method creates and returns a new file.
5. **public StreamWriter CreateText():** This method creates a StreamWriter that writes a new text file.
6. **public void Decrypt():** This method is used to decrypt a file encrypted by the current account using the System.IO.FileInfo.Encrypt method.
7. **public override void Delete():** This method permanently deletes a file.
8. **public void Encrypt():** This method is used to encrypt a file so that only the account used to encrypt the file can decrypt it.
9. **public FileSecurity GetAccessControl():** This method is used to get a System.Security.AccessControl.FileSecurity object that encapsulates the access control list (ACL) entries for the file described by the current System.IO.FileInfo object. That means this method returns a System.Security.AccessControl.FileSecurity object that encapsulates the access control rules for the current file.
10. **public void MoveTo(string destFileName):** This method moves a specified file to a new location, providing the option to specify a new file name. Here, the destFileName parameter specifies the path to move the file to, which can specify a different file name.
11. **public FileStream Open(FileMode mode):** This method opens a file in the specified mode. It returns a file opened in the specified mode, with read/write access and unshared
12. **public FileStream Open(FileMode mode, FileAccess access):** This method is used to open a file in the specified mode with read, write, or read/write access. It returns a System.IO.FileStream object opened in the specified mode, accessed, and unshared.
13. **public FileStream Open(FileMode mode, FileAccess access, FileShare share):** This method opens a file in the specified mode with read, write, or read/write access and the specified sharing option. It returns a FileStream object opened with the specified mode, access, and sharing options. Here, the parameter mode specifies a System.IO.FileMode constant specifying the mode (for example, Open or Append) in which to open the file. The parameter access specifies a System.IO.FileAccess constant specifying whether to open the file with Read, Write, or ReadWrite file access, and the parameter share specifies a System.IO.FileShare constant specifying the type of access other FileStream objects have to this file.
14. **public FileStream OpenRead():** This method creates and returns a new read-only System.IO.FileStream.
15. **public StreamReader OpenText():** This method creates System.IO.StreamReader with UTF8 encoding that reads from an existing text file. It returns a new StreamReader with UTF8 encoding.
16. **public FileStream OpenWrite():** This method creates a write-only System.IO.FileStream. It returns a write-only unshared System.IO.FileStream object for a new or existing file.
17. **public FileInfo Replace(string destinationFileName, string destinationBackupFileName):** This method is used to replace the contents of a specified file with the file described by the current System.IO.FileInfo object, deleting the original file and creating a backup of the replaced file. It returns a System.IO.FileInfo object that encapsulates information about the file described by the destFileName parameter.
18. **public void SetAccessControl(FileSecurity fileSecurity):** This method is used to apply access control list (ACL) entries described by a System.Security.AccessControl.FileSecurity object to the file described by the current System.IO.FileInfo object.
19. **public override string ToString():** This method returns the path as a string.

**static** **void** Main**(string[]** args**)**

**{**

**string** path = @"D:\MyTestFile1.txt";

FileInfo fileInfo = new FileInfo**(**path**)**;

fileInfo.Create**()**;

**{**

Console.WriteLine**(**"File has been created"**)**;

**}**

Console.ReadKey**()**;

**}**

**Crete text**

**static** **void** Main**(string[]** args**)**

**{**

**string** path = @"D:\MyTestFile2.txt";

FileInfo fileInfo = new FileInfo**(**path**)**;

StreamWriter str = fileInfo.CreateText**()**;

str.WriteLine**(**"Hello"**)**;

Console.WriteLine**(**"File has been created with text"**)**;

str.Close**()**;

Console.ReadKey**()**;

**}**

**Delete**

**static** **void** Main**(string[]** args**)**

**{**

**string** path = @"D:\MyTestFile2.txt";

FileInfo fileInfo = new FileInfo**(**path**)**;

fileInfo.Delete**()**;

Console.WriteLine**(**"File has been deleted"**)**;

Console.ReadKey**()**;

**}**

**copyTo**

**static** **void** Main**(string[]** args**)**

**{**

**string** path = @"D:\MyTestFile2.txt";

FileInfo fileInfo = new FileInfo**(**path**)**;

fileInfo.Delete**()**;

Console.WriteLine**(**"File has been deleted"**)**;

Console.ReadKey**()**;

**}**

**moveTo**

**static** **void** Main**(string[]** args**)**

**{**

**string** sourcePath = @"D:\MyTestFile1.txt";

**string** destinationPath = @"D:\Dotnet\MyTestFile1.txt";

FileInfo fileInfo = new FileInfo**(**sourcePath**)**;

fileInfo.MoveTo**(**destinationPath**)**;

Console.WriteLine**(**"{0} was moved to {1}.", sourcePath, destinationPath**)**;

Console.ReadKey**()**;

**}**

**openText**

**static** **void** Main**(string[]** args**)**

**{**

**string** Path = @"D:\MyTestFile1.txt";

FileInfo fileInfo = new FileInfo**(**Path**)**;

StreamReader streamReader = fileInfo.OpenText**()**;

**string** s = "";

**while** **((**s = streamReader.ReadLine**())** != **null)**

**{**

Console.WriteLine**(**s**)**;

**}**

Console.ReadKey**()**;

**}**

# DirectoryInfo Class in C#

این کلاس برای دریافت اطلاعات Directory کاربرد دارد و ویژگی ها که ارائه می دهد کاملا مشابه کلاس FileInfo است با این

تفاوت که این کلاس فوکوس بر Directory کرده است نه fileSystem

زمانی که راجب این کلاس صحبت می کنیم در واقع داریم در رابطه با physicalDirectory صحبت می کنیم. با این کلاس می

توانیم عملیات های بر روی directoy , subDirectory ها انجام دهیم.

نوع کلاس Sealed است و قابل ارث بری نیست.

**Ctor**

**public DirectoryInfo(string path):**

It initializes a new instance of the DirectoryInfo class on the specified path.

Here, the variable path specifies the path to create the DirectoryInfo.

##### **Properties of DirectoryInfo Class in C#**

1. **Parent**: It is used to get the parent directory of a specified subdirectory. It returns the parent directory or null if the path is null or if the file path denotes a root (such as “\”, “C:”, or \* “\\server\share”).
2. **FullName**: It is used to get the full path of the directory. It returns a string containing the full path.
3. **Name**: It is used to get the name of this System.IO.DirectoryInfo instance. It returns the directory name.
4. **Exists**: It is used to get a value indicating whether the directory exists. It returns true if the directory exists; otherwise, false.
5. **Root**: It is used to get the root portion of the directory. It returns an object that represents the root of the directory.
6. **CreationTime**: It is used to get or set the creation time of the current file or directory. It returns the creation date and time of the current System.IO.FileSystemInfo object.
7. **LastAccessTime**: It is used to get or set the time the current file or directory was last accessed. It returns the time that the current file or directory was last accessed.
8. **LastWriteTime**: It is used to get or set the time when the current file or directory was last written. It returns the time the current file was last written.
9. **Extension**: It is used to get the string representing the extension part of the file. It returns a string containing the System.IO.FileSystemInfo extension.

##### **DirectoryInfo Class Methods in C#**

1. **Create():** This method is used to create a directory.
2. **Create(DirectorySecurity directorySecurity):** This method creates a directory using a DirectorySecurity object. The directorySecurity parameter specifies the access control to apply to the directory.
3. **CreateSubdirectory(string path):** This method creates a subdirectory or subdirectory on the specified path. The specified path can be relative to this instance of the DirectoryInfo class. The parameter path specified path.
4. **CreateSubdirectory(string path, DirectorySecurity directorySecurity):** This method creates a subdirectory or subdirectories on the specified path with the specified security. The specified path can be relative to this instance of the DirectoryInfo class. The parameter path specified path. This cannot be a different disk volume or Universal Naming Convention (UNC) name. The directorySecurity parameter specifies the security to apply.
5. **Delete():** This method is used to delete the DirectoryInfo if it is empty.
6. **Delete(bool recursive):** This method is used to delete this instance of a DirectoryInfo, specifying whether to delete subdirectories and files. The recursive parameter specifies true to delete this directory, its subdirectories, and all files; otherwise, it is false.
7. **EnumerateDirectories():** This method returns an enumerable collection of directory information in the current directory. It returns an enumerable collection of directories in the current directory.
8. **EnumerateFiles():** This method returns an enumerable file information collection in the current directory. It returns an enumerable collection of the files in the current directory.
9. **GetAccessControl():** This method is used to get the DirectorySecurity object that encapsulates the access control list (ACL) entries for the directory described by the current DirectoryInfo object. This method returns a DirectorySecurity object that encapsulates the access control rules for the directory.
10. **GetDirectories():** This method returns the subdirectories of the current directory. It returns an array of System.IO.DirectoryInfo objects.
11. **GetFiles():** This method returns a file list from the current directory. It returns an array of type System.IO.FileInfo.
12. **MoveTo(string destDirName):** This method moves a DirectoryInfo instance and its contents to a new path. The parameter destDirName specifies the name and path to which to move this directory. The destination cannot be another disk volume or a directory with an identical name. It can be an existing directory to which you want to add this directory as a subdirectory.
13. **SetAccessControl(DirectorySecurity directorySecurity):** This method is used to set access control list (ACL) entries described by a DirectorySecurity object. The parameter directorySecurity specifies an object that describes an ACL entry to apply to the directory described by the path parameter.
14. **ToString():** It returns the original path that the user passed.

**Create new directory**

**static** **void** Main**(string[]** args**)**

**{**

String path = @"D:\MyTestFile1";

DirectoryInfo fl = new DirectoryInfo**(**path**)**;

fl.Create**()**;

**{**

Console.WriteLine**(**"Directory has been created"**)**;

**}**

Console.ReadKey**()**;

**}**

**Create subDirectory**

**static** **void** Main**(string[]** args**)**

**{**

String path = @"D:\MyTestFile1";

DirectoryInfo fl = new DirectoryInfo**(**path**)**;

DirectoryInfo dis = fl.CreateSubdirectory**(**"MyTestFile2"**)**;

**{**

Console.WriteLine**(**"SubDirectory has been created"**)**;

**}**

Console.ReadKey**()**;

**}**

**Move directory**

**static** **void** Main**(string[]** args**)**

**{**

String path1 = @"D:\MyTestFile1";

**string** path2 = @"D:\NewTestFile1";

DirectoryInfo directoryInfo1 = new DirectoryInfo**(**path1**)**;

DirectoryInfo directoryInfo2 = new DirectoryInfo**(**path2**)**;

directoryInfo1.MoveTo**(**path2**)**;

**{**

Console.WriteLine**(**"Directory has been Moved"**)**;

**}**

Console.ReadKey**()**;

**}**

**Delete Directory**

**static** **void** Main**(string[]** args**)**

**{**

**string** path = @"D:\NewTestFile1 ";

DirectoryInfo directoryInfo1 = new DirectoryInfo**(**path**)**;

directoryInfo1.Delete**()**;

**{**

Console.WriteLine**(**"Directory has been deleted"**)**;

**}**

Console.ReadKey**()**;

**}**

# Export and Import Excel Data in C#

استفاده از کتابخانه NPOL می توان عملیات های مختلف بر روی فایل های xls , doc , ppt انجام داد.

این کتابخانه تقریبا تمامی ویژگی های مهم اکسل را پشتیبانی می کند.

لینک مثال کامل

https://dotnettutorials.net/lesson/export-and-import-excel-data-in-csharp