**C# File I/O Handling Operations [Examples]**

C# has a wide array of file operations. These operations include opening a file, reading or writing to a file. There can be instances wherein you want to work with files directly, in which case you would use the file operations available in C#. Some of the basic file operations are mentioned below.

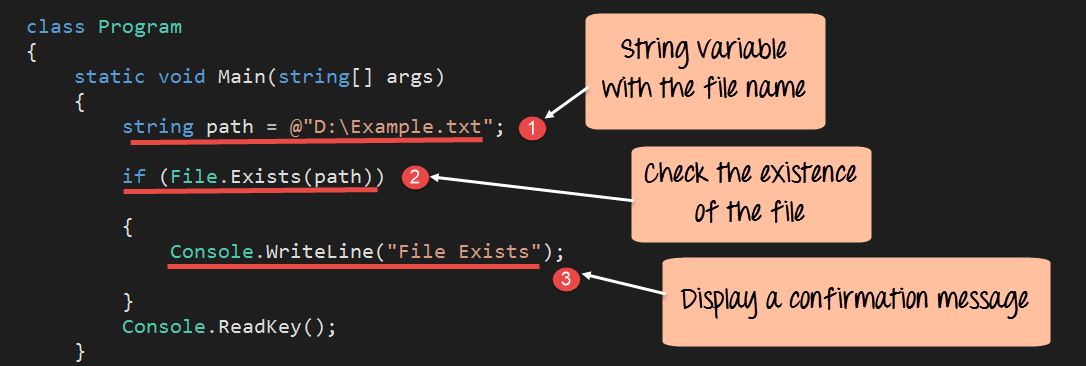
1. Reading – This operation is the basic read operation wherein data is read from a file.
2. Writing - This operation is the basic write operation wherein data is written to a file. By default, all existing contents are removed from the file, and new content is written.
3. Appending – This operation also involves writing information to a file. The only difference is that the existing data in a file is not overwritten. The new data to be written is added at the end of the file.

In this tutorial, you will learn-

* [File.Exists](https://www.guru99.com/c-sharp-file-operations.html#2)
* [File.ReadAlllines](https://www.guru99.com/c-sharp-file-operations.html#3)
* [File.ReadAllText](https://www.guru99.com/c-sharp-file-operations.html#4)
* [File.Copy](https://www.guru99.com/c-sharp-file-operations.html#5)
* [File.Delete](https://www.guru99.com/c-sharp-file-operations.html#6)

## File.Exists

The File exists method is used to check if a particular file exists. So now let's see the code which can be used to check if our Example.txt file exists or not. Enter the below code in the program.cs file.

[](https://www.guru99.com/images/c-sharp-net/052716_0700_CFileOperat1.png)

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DemoApplication

{

class Tutorial

{

static void Main(string[] args)

{

String path = @"D:\Example.txt";

if (File.Exists(path))

{

Console.WriteLine("File Exists");

}

Console.ReadKey();

}

}

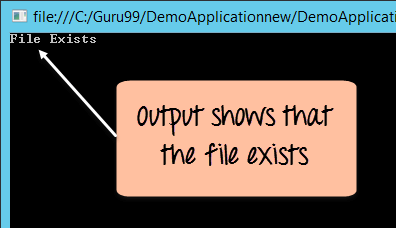
}

**Code Explanation:-**

1. First, we are setting a string variable with the path to our Example.txt file.
2. Next, we use the File.Exists method to check if the file exists or not. If the File exists, a true value will be returned.
3. If we get a true value and the file does exist, then we write the message "File Exists" to the console.

When the above code is set, and the project is executed using Visual Studio, you will get the below output.

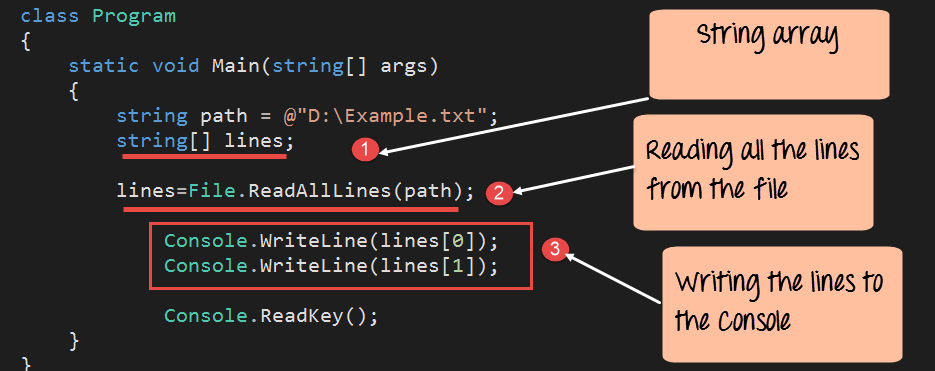
**Output:-**

[](https://www.guru99.com/images/c-sharp-net/052716_0700_CFileOperat2.png)

From the above output, you can see that the File.Exists command was executed successfully, and the correct message was displayed in the console window.

## File.ReadAlllines

The method is used to read all the lines one by one in a file. The lines are then stored in a string array variable. Let's look at an example. Enter the below code in the program.cs file.

[](https://www.guru99.com/images/c-sharp-net/052716_0700_CFileOperat3.png)

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DemoApplication

{

class Tutorial

{

static void Main(string[] args)

{

String path = @"D:\Example.txt";

String[] lines;

lines = File.ReadAllLines(path);

Console.WriteLine(lines[0]);

Console.WriteLine(lines[1]);

Console.ReadKey();

}

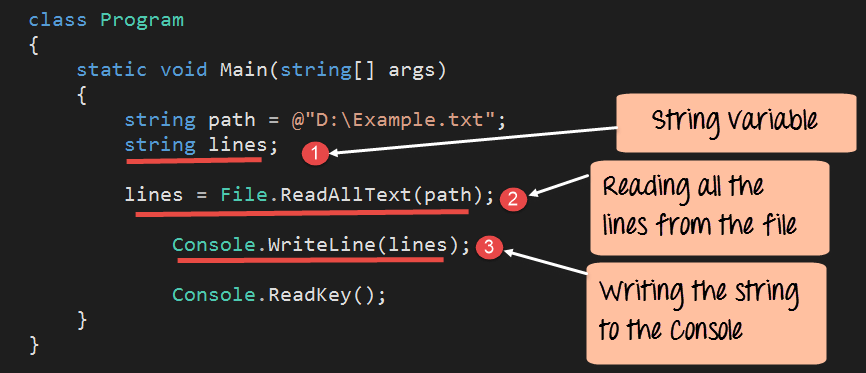
}

}

1. First, we are declaring a string array variable. This will be used to store the result which will be returned by the File.ReadAllLines method.
2. Next, we use the File.ReadAllLines method to read all the lines from our text file. The result is then passed to the lines variable.
3. Since we know that our file contains only 2 lines, we can access the value of the array variables via the lines[0] and lines[1] command.

## File.ReadAllText

This method is used to read all the lines in a file at once. The lines are then stored in a string variable. Let's look at an example. Enter the below code in the program.cs file.

[](https://www.guru99.com/images/c-sharp-net/052716_0700_CFileOperat5.png)

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DemoApplication

{

class Tutorial

{

static void Main(string[] args)

{

String path = @"D:\Example.txt";

String lines;

lines = File.ReadAllText(path);

Console.WriteLine(lines);

Console.ReadKey();

}

}

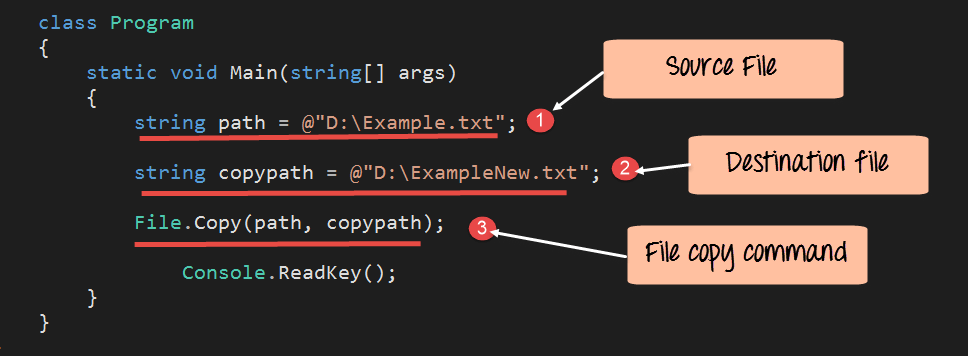
}

**Code Explanation:-**

1. First, we are declaring a string variable called Lines. This will be used to store the result which will be returned by the File.ReadAllText method.
2. Next, we use the File.ReadAllText method to read all the lines from our text file. The result is then passed to the lines variable.
3. We can directly use the Console.Writeline method to display the value of the Lines variable.

## File.Copy

The method is used to make a copy of an existing file. Let's look at an example. Enter the below code in the program.cs file.

[](https://www.guru99.com/images/c-sharp-net/052716_0700_CFileOperat7.png)

using System;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DemoApplication

{

class Tutorial

{

static void Main(string[] args)

{

String path = @"D:\Example.txt";

String copypath = @"D:\ExampleNew.txt";

File.Copy(path,copypath);

Console.ReadKey();

}

}

}

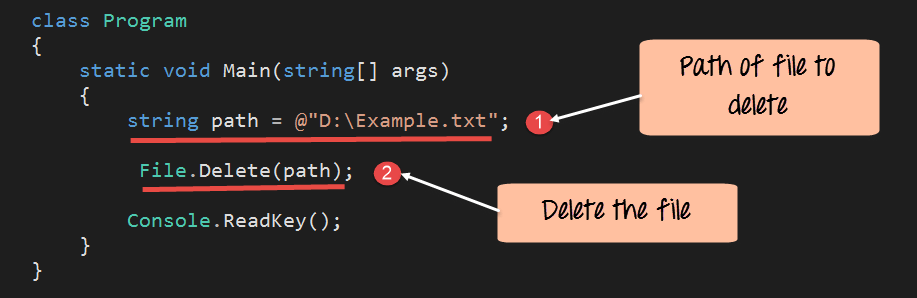
**Code Explanation:-**

1. First, we are declaring a string variable called path. This will be the location of our Example.txt file. This file will be the source file used for the copy operation.
2. Next, we are declaring a string variable called copypath. This will be the location of a new file called ExampleNew.txt file. This will be the destination file in which the contents will be written from the source file Example.txt.
3. We then call the File.Copy method to copy the file Example.txt file to the file ExampleNew.txt.

When the above code is set, and the project is run using Visual Studio, the file Example.txt will be copied to ExampleNew.txt.

## File.Delete

The method is used to delete an existing file. Let's look at an example. Enter the below code in the program.cs file.

[](https://www.guru99.com/images/c-sharp-net/052716_0700_CFileOperat8.png)

using System;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DemoApplication

{

class Tutorial

{

static void Main(string[] args)

{

String path = @"D:\Example.txt";

File.Delete(path);

Console.ReadKey();

}

}

}

**Code Explanation:-**

1. First, we are declaring a string variable called path. This will be the location of our Example.txt file. This is the file which will be deleted.
2. Next, we are calling the File.Delete method to delete the file.

When the above code is set, and the project is run using Visual Studio, the file Example.txt will be deleted from the D drive.

|  |  |
| --- | --- |
| **File Method** | **Description** |
| File.Exists | File exists method is used to check if a particular file exists. |
| File.ReadAlllines | The method is used to read all the lines one by one in a file. |
| File.ReadAllText | This method is used to read all the lines in a file at once. |
| File.Copy | The method is used to make a copy of an existing file. |
| File.Delete | The method is used to delete an existing file. |

# C# FileStream

### WHAT IS FILESTREAM CLASS IN C#?

**FileStream** Class is used to perform the basic operation of reading and writing operating system files. FileStream class helps in reading from, writing and closing files.

### HOW TO USE FILESTREAM CLASS IN C#?

In order to use **FileStream**class you need to include **System.IO** namespace and then create FileStream Object to create a new file or open an existing file.

1. **FileStream** <object\_name> = new **FileStream**( <file\_name>, <**FileMode** **Enumerator**>, <**FileAccess** **Enumerator**>, <**FileShare** **Enumerator**>);

**FileMode**– It specifies how to operation system should open the file. It has following members

1. **Append**- Open the file if exist or create a new file. If file exists then place cursor at the end of the file.
2. **Create**- It specifies operating system to create a new file. If file already exists then previous file will be overwritten.
3. **CreateNew**- It create a new file and If file already exists then throw **IOException**.
4. **Open**– Open existing file.
5. **Open or Create**– Open existing file and if file not found then create new file.
6. **Truncate**– Open an existing file and cut all the stored data. So the file size becomes 0.

**FileAccess**– It gives permission to file whether it will open **Read**, **ReadWrite**or **Write**mode.

**FileShare**– It opens file with following share permission.

1. **Delete**– Allows subsequent deleting of a file.
2. **Inheritable**– It passes inheritance to child process.
3. **None**– It declines sharing of the current files.
4. **Read**- It allows subsequent opening of the file for reading.
5. **ReadWrite**– It allows subsequent opening of the file for reading or writing.
6. **Write**– Allows subsequent opening of the file for writing.

### CREATE A BLANK .TXT FILE USING FILESTREAM

1. using **System**;
2. using **System**.**Collections**.**Generic**;
3. using **System**.**Linq**;
4. using **System**.**Text**;
5. using **System**.**Threading**.**Tasks**;
6. using **System**.IO;
8. namespace **FileStream\_CreateFile**
9. {
10. class **Program**
11. {
12. static void **Main**(string[] args)
13. {
14. **FileStream** fs = new **FileStream**("D:\\csharpfile.txt", **FileMode**.**Create**);
15. fs.**Close**();
16. **Console**.**Write**("File has been created and the Path is D:\\csharpfile.txt");
17. **Console**.**ReadKey**();
18. }
19. }
20. }

Write content into file filestream write()

1. byte[] bdata=**Encoding**.**Default**.**GetBytes**("Hello File Handling!");
2. fs.**Write**(bdata, 0, bdata.**Length**);
3. fs.**Close**();
4. **Console**.**WriteLine**("Successfully saved file with data : Hello File Handling!");

Read content from file to C# using Filestream REadtoEnd()

1. class **Program**
2. {
3. static void **Main**(string[] args)
4. {
5. string data;
6. **FileStream** fsSource = new **FileStream**("D:\\csharpfile.txt", **FileMode**.**Open**, **FileAccess**.**Read**);
7. using (**StreamReader** sr = new **StreamReader**(fsSource))
8. {
9. data = sr.**ReadToEnd**();
10. }
11. **Console**.**WriteLine**(data);
12. **Console**.**ReadLine**();
13. }

Example for streamwriter and streamReader class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace StreamReader\_Class

{

class Program

{

static void Main(string[] args)

{

string file = @"D:\csharpfile.txt";

//Writer data to text file

using (StreamWriter writer = new StreamWriter(file))

{

writer.WriteLine("This tutorial explains how to use StreamReader Class in C# Programming");

writer.WriteLine("Good Luck!");

}

//Reading text file using StreamReader Class

using (StreamReader reader = new StreamReader(file))

{

Console.WriteLine(reader.ReadToEnd());

}

Console.ReadKey();

}

}

}

# TextWriter

The **TextWriter**class represents a writer that can write sequential series of characters. You can use this class to write text in a file. It is an abstract base class of **StreamWriter**and **StringWriter**, which write characters to streams and string respectively.

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

{

string file=@"D:\csharpfile.txt";

using (**TextWriter** writer = **File**.**CreateText**(file))

{

writer.**WriteLine**("Hello TextWriter!");

writer.**WriteLine**("File Handling Tutorial");

}

**Console**.**WriteLine**("Entry stored successfull!");

**Console**.**ReadKey**();

}

}

# TextReader

**TextReader**Class represents a reader that can read a sequential series of Characters. It is abstract class that means you cannot instantiate it. After finishing reading or writing file you must **dispose**or clean memory directly or indirectly. To directly dispose, call **Dispose**Method in **try/catch** block and for indirectly disposal write code inside **using**block.

1. using **System**.IO;
3. namespace **TextReader\_class**
4. {
5. class **Program**
6. {
7. static void **Main**(string[] args)
8. {
9. string filepath = @"D:\csharpfile.txt";
11. //Read One Line
12. using(**TextReader** tr=**File**.**OpenText**(filepath))
13. {
14. **Console**.**WriteLine**(tr.**ReadLine**());
15. }
17. //Read 4 Characters
18. using (**TextReader** tr = **File**.**OpenText**(filepath))
19. {
20. char[] ch = new char[4];
21. tr.**ReadBlock**(ch, 0, 4);
22. **Console**.**WriteLine**(ch);
23. }
25. //Read full file
26. using (**TextReader** tr = **File**.**OpenText**(filepath))
27. {
28. **Console**.**WriteLine**(tr.**ReadToEnd**());
29. }
30. **Console**.**ReadKey**();
31. }
32. }

}

# C# BinaryWriter Class

**BinaryWriter** class writes Primitive type data type as **int**, **uint** or **char** in binary to a stream. As its name says **BinaryWriter** writes binary files that uses a specific data layout for its bytes.

### NOTES:

1. **BinaryWriter** create binary file that is not human understandable but the machine can understand it more smoothly.
2. It supports writing string in a specific encoding.
3. **BinaryWriter** class provides methods that simplify writing primitive data types to a stream.
4. If you don't provide types of encoding while creating object then default encoding **UTF-8** will be used.
5. class **Program**
6. {
7. static void **Main**(string[] args)
8. {
9. using (**BinaryWriter** writer = new **BinaryWriter**(**File**.**Open**("D:\\binaryfile.bin", **FileMode**.**Create**)))
10. {
11. //Writting Error Log
12. writer.**Write**("0x80234400");
13. writer.**Write**("Windows Explorer Has Stopped Working");
14. writer.**Write**(true);
15. }
16. **Console**.**WriteLine**("Binary File Created!");
17. **Console**.**ReadKey**();
18. }
19. }
20. }

### WHAT IS BINARYREADER CLASS IN C#?

If you have bin file stored in your PC and you want to read them then **BinaryReader** may help you lot. **BinaryReader** class is used to reading binary files. A binary file stored data in different layout that is more efficient for machine but not convenient for human. **BinaryReader** makes this job simpler and show you exact data stored in bin file.

**Notes:**

1. BinaryReader handles Binary (.bin) files.
2. It reads primitive data types as binary values in a specific encoding.
3. class **Program**
4. {
5. static void **Main**(string[] args)
6. {
7. **WriteBinaryFile**();
8. **ReadBinaryFile**();
9. **Console**.**ReadKey**();
10. }
11. static void **WriteBinaryFile**()
12. {
13. using (**BinaryWriter** writer = new **BinaryWriter**(**File**.**Open**("D:\\binaryfile.bin", **FileMode**.**Create**)))
14. {
15. //Writting Error Log
16. writer.**Write**("0x80234400");
17. writer.**Write**("Windows Explorer Has Stopped Working");
18. writer.**Write**(true);
19. }
20. }
21. static void **ReadBinaryFile**()
22. {
23. using (**BinaryReader** reader = new **BinaryReader**(**File**.**Open**("D:\\binaryfile.bin", **FileMode**.**Open**)))
24. {
25. **Console**.**WriteLine**("Error Code : " + reader.**ReadString**());
26. **Console**.**WriteLine**("Message : " + reader.**ReadString**());
27. **Console**.**WriteLine**("Restart Explorer : " + reader.**ReadBoolean**());
28. }
29. }
30. }

# C# DirectoryInfo

**DirectoryInfo**class allows you to work with directory and its make directory manipulation as create, delete, info etc easy. It exposes instance methods for creating, moving, enumerating through directories and subdirectories.

#### Notes

1. **DirectoryInfo** class is used for typical operations such as copying, moving, creating or deleting directories.
2. This class cannot be inherited.
3. By default full read/write access to new directories is granted to all users.
4. class **Program**
5. {
6. static void **Main**(string[] args)
7. {
8. string path=@"D:\csharp1";
9. **DirectoryInfo** dir = new **DirectoryInfo**(path);
10. try
11. {
12. if (dir.**Exists**)
13. {
14. **Console**.**WriteLine**("{0} Directory is already exists", path);
15. **Console**.**WriteLine**("Directory Name : " + dir.**Name**);
16. **Console**.**WriteLine**("Path : " + dir.**FullName**);
17. **Console**.**WriteLine**("Directory is created on : " + dir.**CreationTime**);
18. **Console**.**WriteLine**("Directory is Last Accessed on " + dir.**LastAccessTime**);
19. }
20. else
21. {
22. dir.**Create**();
23. **Console**.**WriteLine**(path + "Directory is created successfully");
24. }
25. //Delete this directory
26. **Console**.**WriteLine**("If you want to delete this directory press small y. Press any key to exit.");
27. try
28. {
29. char ch = **Convert**.**ToChar**(**Console**.**ReadLine**());
30. if (ch == 'y')
31. {
32. if (dir.**Exists**)
33. {
34. dir.**Delete**();
35. **Console**.**WriteLine**(path + "Directory Deleted");
36. }
37. else
38. {
39. **Console**.**WriteLine**(path + "Directory Not Exists");
40. }
41. }
42. }
43. catch
44. {
45. **Console**.**WriteLine**("Press Enter to Exit");
46. }
47. **Console**.**ReadKey**();
48. }
49. catch(**DirectoryNotFoundException** d)
50. {
51. **Console**.**WriteLine**("Exception raised : " + d.**Message**);
52. }
53. }
54. }

# C# FileInfo

**FileInfo**class in c# is used for manipulating file as creating, deleting, removing, copying, opening and getting information. It provides properties and instance methods that makes file manipulation easy.

### NOTES

1. **FileInfo**class is used for typical operation like copying, moving, renaming, creating, opening, deleting and appending the file.
2. By default, full read/write access to new files is granted to all users.
3. static void **Main**(string[] args)
4. {
5. string path = @"D:\csharp\fileinfo.txt";
6. **FileInfo** file = new **FileInfo**(path);
7. //Create File
8. using (**StreamWriter** sw = file.**CreateText**())
9. {
10. sw.**WriteLine**("Hello FileInfo");
11. }
13. //Display File Info
14. **Console**.**WriteLine**("File Create on : " + file.**CreationTime**);
15. **Console**.**WriteLine**("Directory Name : " + file.**DirectoryName**);
16. **Console**.**WriteLine**("Full Name of File : " + file.**FullName**);
17. **Console**.**WriteLine**("File is Last Accessed on : " + file.**LastAccessTime**);
19. //Deleting File
20. **Console**.**WriteLine**("Press small y for delete this file");
21. try
22. {
23. char ch = **Convert**.**ToChar**(**Console**.**ReadLine**());
24. if (ch == 'y')
25. {
26. if (file.**Exists**)
27. {
28. file.**Delete**();
29. **Console**.**WriteLine**(path + " Deleted Successfully");
30. }
31. else
32. {
33. **Console**.**WriteLine**("File doesn't exist");
34. }
35. }
36. }
37. catch
38. {
39. **Console**.**WriteLine**("Press Enter to Exit");
40. }
41. **Console**.**ReadKey**();
42. }