## Continue



Opening files in C# involves types from the System.IO namespace. Methods like File.WriteAllText. More advanced methods, like StreamReader and StreamWriter, are often better choices.StreamReaderFor text files, StreamReader and StreamWriter are often the most useful types. StreamReader introduces some complexity in our use of the languagewe see the "using" blockthis allows automatic cleanup of resources. Step 2 We call ReadLine. This is a method on StreamReader. It returns null if no further data is available in the file. Step 3 Here we have the line variable. This contains a line of the file (with no newlines included). using System; using System; using System; using System; using System; line of the file (with no newlines included). It is a static void Main() { // Step 1: open file for reading. using System; using System; using System; using System; line of the file (with no newlines included). It is a static void Main() { // Step 1: open file for reading. using System; us string line; while ((line = reader.ReadLine()) != null) { // Step 3: do something with the line. Console.WriteLine(\$"LINE: {line}"); } } } LINE: Hello my friendLINE: Welcome to the InternetLINE: Third line in fileStreamWriterThis class writes strings or append strings to a text file. We can write numbers or the textual representation of anything. It also uses a "using" block.using System.IO; class Program { static void Main() { // Create or open file and write line to it. // ... If file exists, it contents are erased before writing. using (var writer = new StreamWriter(@"C:\programs\example.txt")) { writer.WriteLine("HELLO"); } }}ReadAllTextThis program uses this method to load in the file "file.txt" on the C: volume. Then it prints the contents of the file. The data is now stored in a string object. Tip ReadAllText is the easiest way to put a file into a string. It is part of the System. IO; class Program { static void Main() { string file = File. ReadAllText("C:\file.txt"); Console. WriteLine(file); }} ReadAllText is the easiest way to put a file into a string. It is part of the System. IO; class Program { static void Main() { string file = File. ReadAllText("C:\file.txt"); Console. WriteLine(file); }} ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string. It is part of the System. IO; class Program { static void Main() { string file = File. ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. Tip ReadAllText is the easiest way to put a file into a string object. The easiest way to put a file into a string object. The easiest way to put a file into a string object. The easies read all the lines from a file and place them in an array. The code reads lines from "file.txt" and uses a foreach-loop on them. This is efficient code.using System.IO; class Program { static void Main() { // Read in every line in specified file. // ... This will store all lines in an array in memory. string[] lines = File.ReadAllLines("file.txt"); foreach (string line in lines) { // Do something with the line. if (line.Length > 80) { // Important code. } } } } } Count lines we count the number of lines in a file with few lines of code. The example here is a bit slow. But it works. It references the Length property.using System.IO; class Program { static void Main() { // Another method of counting lines in a file. // ... This is not the most efficient way. // ... It counts empty lines. int lineCount = File.ReadAllLines("file.txt").Length; }} QueryDoes a line containing a specific string exist in the file? Maybe we want to see if a name or location exists in a line in the file. We use LINQ to find any matching line.using System.Linq; class Program { static void Main() { // See if line exists in a file. // ... Use a query expression to count matching lines. // ... If one matches, the bool is set to true. bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == "Some line matches, the bool exists = (from line in File.ReadAllLines("file.txt") where line == (from line in File.ReadAllLines( We use it in a foreach-loop.using System; using System; Us within-memory processing, we often need to write the data to disk.using System.IO; class Program { static void Main() { // Write a string array to a file. stringArray); }} catdogarrowWriteAllTextA simple method, File.WriteAllText receives two arguments. It receives the path of the output file, and the exact string contents of the text file. Note The file is created if it does not exist, or replaced with a new file if it does exist (no appends ever occur). using System. IO; class Program { static void Main() { File. WriteAllText("C:\\perls.txt", "Dot Net Perls"); }} Append AllTextWe could read in a file, append to that in memory, and then write it out completely again. That is slowit's more efficient to use an append. Argument 1 The first argument to File. Append and then write it out completely again. That is slowit's more efficient to use an append. Argument 1 The first argument to File. Append and then write it out completely again. That is slowit's more efficient to use an append text to. Argument 1 The first argument to File. Append and then write it out completely again. That is slowit's more efficient to use an append text to. Argument 1 The first argument to File. Append and then write it out completely again. That is slowit's more efficient to use an append text to. Argument 1 The first argument to File. Append and then write it out completely again. in IL Disassembler: AppendAllText internally calls StreamWriter, with the second parameter "appended to. Otherwise, a new file is created.using System.IO; class Program { static void Main() { // Use AppendAllText to write one line to the text file. File.AppendAllText("C:\\perls.txt", "first part"); // Write a third line. string third = "third partfirst partsecond partthird partfirst partsecond partf partReadAllBytesWe use File.ReadAllBytes to read an image into memory. Here we read the bytes from a WEBP file into a byte array, and print its length and the first byte.using System;using System;usin Console.WriteLine("Length: {0}", webpFile.Length: {0}", webpFile.Length: {0}", webpFile.Length: 822First byte: {0}", webpFile.Cength: 822First byte: 82Benchmark, ReadLineSuppose we have a text file with many linespossibly 1 million lines. We could use a method like ReadAllLines, or StreamReader on the file.Version 1 In this version of the code, we read in the file with StreamReader line-by-line. Version 2 Here we read in the file with File. ReadAllLines. The code is more complex and longer. Result Using System. Diagnostics; using System. Diagnostics; using System. For files with many lines, it is worth reading in the lines iteratively. Using System. Diagnostics; using System. Diagnostics; using System. Diagnostics and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. Result Using ReadLine with File. ReadAllLines. The code is more complex and longer. ReadLine with File. ReadAllLines. The code is more complex and longer. ReadLine with File. ReadAllLines. The code is more complex and longer. ReadLine with File. ReadAllLines. The code is more complex and longer. ReadLine with File. ReadAllLines. The code is more complex and longer. ReadLine with File. ReadLine with Fil CreateFileWithManyLines() { // Create temporary file for benchmark. using (StreamWriter writer = new StreamWriter(@"C:\programs\file.txt")) { for (int i = 0; i < 1000000; i++) { writer.WriteLine("x"); } } } const int \_max = 10; static void Main() { CreateFileWithManyLines(); // Version 1: use StreamReader and read in each line. var s1 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method1() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } } s1.Stop(); // Version 2: use File.ReadAllLines to get entire string array. var s2 = Stopwatch.StartNew(); for (int i = 0; i < \_max; i++) { if (Method2() == 0) { return; } s1.Stop(); // Version 2: use File.ReadAllLines to get e Console.WriteLine(s2.Elapsed.TotalMilliseconds.ToString("0.00 ms")); } static int Method1() { int count = 0; using (StreamReader reader = new StreamReader(@"C:\programs\file.txt")) { while (true) { string line = reader.ReadLine(); if (line == null) { break; } count++; } } return count; } static int Method2() { string[] array = new StreamReader(@"C:\programs\file.txt")) { while (true) { string line = reader.ReadLine(); if (line == null) { break; } count++; } } File.ReadAllLines(@"C:\programs\file.txt"); return array.Length; }}219.53 ms StreamReader, ReadLine1212.43 ms File.ReadAllLinesEven with the helpful types provided in .NET, file handling involves many errors. We must account for disk problems and invalid data. File handling is a fundamental aspect of programming, and C# provides various methods to read from and write to files effectively. In this article, we will explore the best practices and efficient methods for file operations in C#, covering both reading and writing text content. using System; using System. IO; string filePath = "example.txt";string content = File.ReadAllText(filePath);Console.WriteLine("File written successfully."); File.ReadAllText reads the entire content of a file into a single string. This method is suitable for smaller files, as it loads the entire file into memory. File.WriteAllLines are efficient and easy to use. using System; using System.IO; string filePath = "example.txt"; string[] lines = File.ReadAllLines(filePath); Console.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = "example.txt"; string[] lines = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = { "Line 1", "Line 2", "Line 3" }; File.WriteLine(line); Using System.IO; string filePath = { "Line 1", "Line 3" }; File.WriteLine(line); Using System.IO; written with multiple lines successfully."); File.ReadAllLines reads each line of the file into a string array, making it easy to iterate over lines. File.WriteAllLines writes an array of strings to a file, with each element representing a line. For large files, StreamReader and StreamWriter allow efficient reading and writing by handling data in streams rather than loading the entire file into memory. using System; using Sys new StreamWriter(filePath)) { writer.WriteLine("This is the first line."); Writer.WriteLine("This is the second line."); }Console.WriteLine("This is the file, which is suitable for handling large volumes of data. Using using statements with StreamReader and StreamWriter ensures that file resources are properly disposed of when the operations or are working with binary data, FileStream is the most efficient approach. It provides low-level access to files. using System; using System.IO; string filePath = "binaryFile.dat"; using (FileStream fs = new FileStream(filePath, FileMode.Open, FileAccess.Read)) { byte[] buffer = new byte[fs.Length]; fs.Read(buffer, 0, buffer.Length); Console.WriteLine("Binary file content:"); foreach (byte b in buffer) { Console.Write(b + " "); }} using System; using System; using System.IO; string filePath = "binaryFile.dat"; using System; using System; using System.IO; string filePath = "binaryFile.dat"; using System; u "binaryFile.dat";byte[] data = { 0x01, 0x02, 0x03, 0x04 };using (FileStream fs = new FileStream fs = new F structure. FileMode and FileAccess parameters allow for flexibility in how the file is accessed (e.g., FileMode.Create creates a new file). BufferedStream can be layered over FileStream to improve performance in cases where you perform many small read or write operations. using System; using Sys (FileStream fs = new FileStream(filePath, FileMode.OpenOrCreate))using (BufferedStream (s)) { bytesRead } bytesRead bytesRead } bytesRead by reducing the number of actual read or write operations to the disk. This can be beneficial for files with frequent small files with text contentModerateNoFile.ReadAllLines / File.WriteAllLinesSmall files with frequent small reads or writes. MethodBest ForMemory UsageCreates StreamFile.ReadAllLines / File.WriteAllLinesSmall files with frequent small reads or writes. StreamWriterLarge files with text contentLowYesFileStreamBinary data or byte-level controlLowYesBufferedStreamHigh-frequency small reads/writesLowYes In C#, there are multiple ways to efficiently read and write files depending on your requirements. For small files, File.ReadAllText or File.WriteAllText are convenient and fast, while StreamReader and StreamWriter are better suited for larger files. For binary data or byte-level control, FileStream and BufferedStream provide optimized handling in C#. Download "C# Essentials: A Developer's Cheat Sheet" for key syntax, tips, and quick references namespace File.SomeFileMethod(); // use the file class with methodsThe File class with methods for creating and getting information about files. For example:MethodOescriptionAppendText()Appends text at the end of an existing fileCopy()Copies a fileExists()Tests whether the file existsReadAllText()Reads the contents of a file Replace()Replaces the contents of a file with the contents of a fi we use the WriteAllText() method to create a file named "filename.txt" and write System.IO; // create a file and write a file the content of writeText to itstring readText = File.ReadAllText("filename.txt"); // Read the contents of the fileConsole.WriteLine(readText); // Output the contentThe output will be:Hello World!Run example There are a few ways to create a file and write to it using the .NET File API (in System.IO). The simplest way is to use high-level methods like File.WriteAllText() and File.WriteAllLines(), specifying the file path and string(s) to write to the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file path and string(s) to write to the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file path and string(s) to write to the file. Heres an example of using these (and their async equivalents): using System.IO; File.WriteAllLines(), specifying the file path and string(s) to write to the file. Heres an example of using these (and their async equivalents): using System.IO; File.WriteAllLines(), specifying the file path and string(s) to write to the file. Heres an example of using these (and their async equivalents): using System.IO; File.WriteAllLines(), specifying the file path and string(s) to write to the file. Heres an example of using these (and their async equivalents): using System.IO; File.WriteAllLines(), specifying the file path and string(s) to write the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file path and string(s) to write the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file path and string(s) to write the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file. Heres an example of using System.IO; File.WriteAllLines(), specifying the file. Heres an example of using System.IO; File.WriteAllLines(), specifying System.IO; File.WriteAllLi File.WriteAllTextAsync(@"C:\temp\a async.txt", "Hello World Async"); await File.WriteAllLinesAsync(@"C:\temp\a async.txt", lines); Code language: C# (cs) These high-level methods abstract away the details. They create the file, open it for writing, write the content, and then close the file. If the file already exists, they overwrite it. The next best option is to use File.CreateText(). This returns a StreamWriter, which you can use to write to the file. Heres an example: using System.IO; using (StreamWriter.WriteLine(Tue); Code language: C# (cs) This creates the text file, opens it for writing, writes each line, and then closes the file (when it falls out of the using block). If the file already exists, this overwrites it. Heres what the file content looks like: Hello World7TrueCode language: plaintext (plaintext) Ill show examples of less common scenarios (like writing to a binary file), and also discuss some problems to avoid. If you try to create a file in a non-existing directory, youll get an exception like this: System.IO.DirectoryNotFoundException: Could not find a part of the path C:\data\en\hello.txt. You can use Directory, youll get an exception like this: System.IO.Directory() to create the missing directories in the path. Heres an example (C:\data\ and C:\data\ and C:\data\en\ terestory); File.WriteAllText(filePath, "Hello new directory"); Code language: C# (cs) This first creates the C:\data\ and C:\d C:\data\en\ and writes to it. Note: If the directory already exists, Directory. CreateDirectory() does nothing. You can specify a relative or absolute file path when creating a file. If you specify just a file name, itll create the file in the current directory (whatever Environment. Current Directory is set to). Heres an example: using System.IO; Console. WriteLine ("Current directory:"); Console. WriteLine (Environment. Current Directory); File. WriteLine (Environment. Current Directory); File. WriteLine (Environment. Current Directory); File. WriteLine (Environment. Current Directory); Tonsole. WriteLine (Environment. Current Directory); File. WriteLine (Environment. Current Directory); Tonsole. WriteLine (Environment. Current Di Current directory:D:\Project\bin\Debuget6.0Does the file exist? TrueCode language: plaintext (plaintext) Most of the time youll be dealing with text files, but sometimes you may need to create and write to binary files. The two simplest ways to do this are to use File.WriteAllBytes() or use File.Create() and BinaryWriter. These are equivalent to the text file methods I showed up above. Heres an example of writing a byte array to a binary file with File. WriteAllBytes(): using System. IO; byte[] data = new byte[] { 0b0110\_1100, 0b0110\_1100, 0b0110\_1111}; File. WriteAllBytes(@"C:\temp\data.bin", data); Code language: C# (cs) This creates a binary file with the ASCII bytes for hello in it. The other option is to use File.Create(@"C:\temp\hello.bin"))){ writer.Write("Hello World"); writer.Write(true); writer.Write(10); writer.Write(7.5m); Code language: C# (cs) This creates a binary file and writes bytes for whatever objects you pass it. Note: If you open it up in Notepad++, you can see the unprintable characters, like [NUL]. last modified May 12, 2025This tutorial explores file handling in C# and covers essential operations such as creating, reading, writing, deleting, and appending files. In C#, file management is handled using the System. IO and System. Text namespaces, providing efficient tools for working withfiles. The File class in System. IO and System. Text namespaces, providing efficient tools for working withfiles. The File class in System. IO and System. Text namespaces, providing efficient tools for working withfiles. The File class in System. IO and objects foradvanced file manipulation. For object-oriented file handling, .NET provides the FileInfoclass. While FileInfoclass. While FileInfoclass are utility class for managingfiles across applications. The words.txt file This is a text file that we work with in some examples. words.txt skybluecloudraisintreefalconowleaglerockwaterlakeThe File.create creates or overwrites a file in the specified in path. Program.cs using System.Text;var path = "words.txt";using FileStream fs = File.Create(path);byte[] data = Encoding.UTF8.GetBytes("falconribboncloud");fs.Write(data, 0, data.Length);Console.WriteLine("data written to file");The example creates a file and writes three words into it. This demonstrates howto use a FileStream to write raw bytes to a file, which is useful when you needprecise control over file content or are working with binary data.using FileStream fs = File.Create(path); We create a file and retrieve a file stream to the file's bytes, allowing for efficient reading.UTF8.GetBytes. This step is necessary because files storedata as bytes, and converting strings to bytes ensures the correct encoding issued file. Create Text The File. Create Text creates or opens a file for writing UTF-8encoded text. If the file already exists, its contents are overwritten. Itreturns a StreamWriter that writes to the specified file using UTF-8 encoding. Program.cs var path = "words.txt"; using StreamWriter sw = File.CreateText(path);sw.WriteLine("falcon");sw.WriteLine("cloud");Console.WriteLine("cloud");Sw.WriteLine("cloud");Sw.WriteLine("cloud");Sw.WriteLine("cloud");Sw.WriteLine("cloud");Sw.WriteLine("cloud");Console.WriteLine("cloud");Console.WriteLine("cloud");Sw.WriteLine("cloud");Sw.WriteLine("cloud");Console.WriteLine("cloud");Console.WriteLine("cloud");Sw.WriteLine("clou C# File.Copy The File.copy copies an existing file to a new file. It takes the source file and the destination file as parameters. Program.cs var sourcePath, destPath); Console.WriteLine("file copied"); The example copies a text file. This operation is useful for creating backups orduplicating files for further processing without altering the original. C# File.Move moves a specified file to a new location or to rename a file to a new location. With thismethod, we can move a file to a new location or to rename a file. Program.cs var sourcePath = "words.txt"; File.Move(sourcePath, destPath); Console.WriteLine("file moved"); The example renames words.txt to data.txt. Thisdemonstrates how to move a file to a new location or change its name within thesame directory. C# File.Exists (path)) { Console.WriteLine("the file exists");} else { Console.WriteLine("the file exists");} else { Console.WriteLine("the file exists");} not exist");}In the code example we check if the words2.txt file exists. This is a common operation to prevent errors when attempting to access or modify filesthat may not be present. C# File.Delete deleted the specified file. Program.cs var path = "words.txt";File.Delete (path);Console.WriteLine("file deleted");The example deletes a text file. Deleting files is essential for managing diskspace and removing outdated or unnecessary data from your application. C# File.GetCreationTime returns the creationTime of thespecified file or directory. Program.cs var path = "words.txt";DateTime dt = File.GetCreationTime (path);Console.WriteLine(\$"CreationTime The File.GetCreationTime The File.GetCreationTime The File.GetCreationTime (path);Console.WriteLine(\$"CreationTime The File.GetCreationTime The File.GetCrea time: {dt}"); The example prints the creation time of the words.txt file. Knowing when a file was created can be useful for logging, auditing, or managing file lifecycles.\$ dotnet runCreation time of thespecified file or directory was last written to. Program.cs var path = "words.txt"; DateTime dt = File. GetLastWriteTime(path); Console. Write time: {dt}"); The example prints the last write time of the text file. Open The File. Open opens a FileStream on the specified path. The overloaded methods of File. Open allow to specify the file mode (Open, Create, CreateNew, Truncate, Append, or OpenOrCreate), the file access (Read, Write, ReadWrite, Write, or Inheritable. Program.cs using System.Text;var path = "words.txt";using FileStream fs = File.Open(path, FileMode.Open, FileAccess.Read);byte[] buf = new byte[1024];int c;while ((c = fs.Read(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads the contents of the words.txt file. Thisdemonstrates how to use a FileStream to read raw bytes from a file and convertthem into a string using a specific encoding. This ensures that the file for reading. This ensures that the file is accessed in a waythat prevents accidental modification while allowing data to be readefficiently byte[] buf = new byte[1024]; int c; We create a buffer of bytes and an auxiliary variable. The buffer temporarilyholds the data read from the file, and the variable tracks how many bytes were actually read. While ((c = fs.Read(buf, 0, c)); The Read method reads a block of bytes from the stream and writesthe data to the given buffer. The Encoding UTF8.GetString decodes asequence of bytes into a string. This approach is efficient for reading largefiles in chunks.\$ dotnet runskybluecloudraisintreefalconowleaglerockwaterlake C# File.OpenRead The File.Op conveniencemethod to the File.Open. Program.cs using System. Text; var path = "words.txt"; using FileStream fs = File.OpenRead(path); byte[] buf = new byte[1024]; int c; while ((c = fs.Read(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead. This method simplifies opening a file with File.OpenRead(path); byte[] buf = new byte[1024]; int c; while ((c = fs.Read(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} The example reads a file with File.OpenRead(buf, 0, buf.Length)) > 0){ Console.WriteLine(Encoding.UTF8.GetString(buf, 0, c));} file for reading by returning a read-only FileStream, making it easyto process file data. C# File.OpenText opens an existing UTF-8 encoded text file forreading. It returns a StreamReader on the specified path. Program.cs var path = "words.txt"; using StreamReader on the specified path. Program.cs var path = "words.txt"; using StreamReader on the specified path. sr.ReadLine()) != null){ Console.WriteLine(s);} The example opens a text file and reads its contents. Using aStreamReader allows you to read text files such as logs or CSVs.while ((s = sr.ReadLine()) != null){ Console.WriteLine(s);} We read the text file line by line. We do not have to decode the bytes into thestring ourselves. The StreamReader handles the conversion from bytes to strings, making text processing straightforward. C# File.OpenWrite opens an existing file or creates a new file forwriting. It returns a FileStream object on the specified path withwrite access. Program.cs var path = "langs.txt"; using FileStream fs = File.OpenWrite(path);using StreamWriter sr = new StreamWriter(fs);sr.WriteLine("PHPDartJavaC#");Console.WriteLine("data written");The example opens a file in write mode and writes a line to the file. Thisdemonstrates how to use a FileStream and StreamWriter for write text data efficiently.using FileStream fs = File.OpenWrite(path); using StreamWriter sr = new StreamWriter sr = new StreamWriter sr = new StreamWriter, which is used to write characters in a particular encoding (UTF8 by default). This setup is useful for writing large amounts of text data to a file.sr.WriteLine("PHPDartJavaC#"); A line of text is written to the file with WriteLine. TheStreamWriter automatically adds a newline character, making it easy to writemultiple lines of text. C# File.ReadLines, it does not load all lines into memory at once, making it more efficient for large files. Program.cs var path = "words.txt"; var lines = File.ReadLines (path); foreach (var line in lines) { Console.WriteLine(line); } This example reads lines from a file one at a time using File.ReadLines, reducing memory usage compared to File.ReadLines, reducing memory usage compared to File.ReadLines (path); foreach loop, processing each line as needed. This is especially useful for processing large files without loading them entirely into memory. C# File.ReadAllLines method reads all lines of a text file into astring[] array in a single operation. Since the entire file isloaded into memory at once, this approach is better suited for small tomoderately-sized files. Program.cs var path = "words.txt";string[] lines = File.ReadAllLines(path);foreach (var line in lines){ Console.WriteLine(line);}This example reads all lines from a file at once, storing them in a string[] array. The entire array is then iterated through using a foreach loop. This approach is suitable for small to moderately-sized files where memory usage is not a concern.If working with large files, consider using File. ReadAllText opens a text file, reads all the text in thefile into a string, and then closes the file. Note that this method should notbe used for very large files. Program.cs var path = "words.txt";string readText = File.ReadAllText(path);Console.WriteLine(readText);The example reads the whole text into a string in one go. This is convenient for quickly loading the entire contents of a file, but should be avoided for very large files to prevent excessive memory usage. C# File.ReadAllBytes The File.ReadAllBytes opens a binary file, reads the contents ofthe file into a byte array, and then closes the file. Program.cs var path = "favicon.ico"; byte[] data = File.ReadAllBytes(path); int i = 0; foreach (byte c in data) { Console.WriteLine(); }} The example reads a favicon.ico binary file. The data is the example, we write four words to a file withFile.WriteAllLines The File.WriteAllLines creates a new file, overwriting any existing content. C# File.WriteAllLines The File.WriteAllLines The File.WriteAllLines creates a new file, writes one or morestrings to the file, writes one or morestrings to the file. WriteAllLines The File.WriteAllLines The File.WriteAllLine { "sky", "cloud", "falcon", "hawk" }; File.WriteAllLines(path, data, Encoding.UTF8); Console.WriteAllBytes The File.WriteAllBytes creates a new file, writes the specifiedbyte array to the file, and then closes the file. Program.cs using System.Text;var text = "falconhawkforestcloudsky";byte[] data = Encoding.UTF8.GetBytes(text);File.WriteAllBytes. Thismethod is ideal for saving binary data, such as images or serialized objects, directly to a file.var text = "falconhawkforestcloudsky"; byte[] data = Encoding.UTF8.GetBytes. This conversion ensures that the text isproperly encoded for storage as binary data. File. WriteAllBytes (path, data); Then we write the array to the file with File. AppendText Creates a StreamWriter that appends UTF-8 encoded text to an existing file, or to a new file if the specified file does not exist. Program.cs var path = "words.txt";using StreamWriter sw = File.AppendText(path);sw.WriteLine("sky");sw.WriteLine("lake");The example appends two words to the words.txt file. AppendAllText The File.AppendAllText appends the specified string to the file.It creates the file if it does not exist. Program.cs var path = "words.txt"; var contents = "armourswordarrow"; File.AppendAllText(path, contents); Console.WriteLine("text appended to file"); The example appends three words to the specified text file. This method isconvenient for adding text to a file, automatically creating the file if it does not exist. C# File.AppendAllLines The File.AppendAllLines appends lines to a file, and then closesthe file. Program.cs var path = "words.txt";List data = ["brown", "blue", "khaki"];File.AppendAllLines appends lines to a file, and then closesthe file. Program.cs var path = "words.txt";List data = ["brown", "blue", "khaki"];File.AppendAllLines appends lines to a file, and then closesthe file. Program.cs var path = "words.txt";List data = ["brown", "blue", "khaki"];File.AppendAllLines appends lines to a file, and then closesthe file. Program.cs var path = "words.txt";List data = ["brown", "blue", "khaki"];File.AppendAllLines appends lines to a file, and then closesthe file. Program.cs var path = "words.txt";List data = ["brown", "blue", "khaki"];File.AppendAllLines appends lines to a file, and then closesthe file. Program.cs var path = "words.txt";List data = ["brown", "blue", "khaki"];File.AppendAllLines appends lines to a file, and then closesthe file. Program.cs var path = "words.txt";List data = ["brown", "blue", "khaki"];File.AppendAllLines appends lines to a file, and then closesthe file. Program.cs var path = "words.txt";List data = ["brown", "blue", "khaki"];File.AppendAllLines appends lines to a file, and then closesthe file. Program.cs var path = "words.txt", "blue", "khaki"];File.AppendAllLines appends lines to a file of the file. Program appends lines appen oftext to a file in a single operation. C# File.Replace The File.Replace method replaces the contents of a specified filewith the contents of another file, optionally creating a backup file = "words backup.txt"; File.Replace (sourceFile, destFile, destFile, destFile, optionally creating a backup file = "words backup.txt"; File.Replace method replaces the contents of another file, optionally creating a backup file = "words.txt"; var backup.file = "words.txt"; var backup.file = "words.txt"; of another file, optionally creating a backup of the replaces file = "words.txt"; of another file, optionally creating a backup of the replaces file = "words.txt"; of another file, optionally creating a backup.txt"; of another file, optionally creating a backup of the replaces file = "words.txt"; of another file, optionally creating a backup.txt"; of ano backupFile); Console. WriteLine ("file replaced"); The example replaces the contents of words 2.txt withwords.txt and creates a backup. This operation is useful forugating files while preserving the previous version for recovery if needed. C# File. SetAttributes and File. GetAttributes and Fil and retrieve file attributes such as read-only or hidden. Program.cs var path = "words.txt"; File.SetAttributes (path); Console.WriteLine(\$"File attributes and prints itsattributes. This demonstrates how you can programmatically change fileproperties and verify the changes, which is especially useful for automatingfile management tasks or enforcing certain file access policies in your C#programs. In this article we have worked with files in C#. We have utilized the File class of the System. IO. The provided examples cover a wide range of file operations, giving you practical knowledge to handlefiles efficiently and securely in your own C# projects. Source C# File class - language referenceMy name is Jan Bodnar, and I am a passionate programmer with extensive programming experience. I have been writing programming articles since 2007. To date, I have authored over 1,400 articles and 8 e-books. possess morethan ten years of experience in teaching programming.List all C# tutorials

Simple resume format for teacher job. How to write resume for teacher job fresher with no experience. Format of resume for teacher job. Fresher teacher job resume. Resume format for freshers for teacher job. Teaching job resume for freshers. Write a resume for teacher job. Resume for teacher fresh graduate.

• http://ydfthjx.com/userfiles/files/1756168989.pdf • https://ru-septik.ru/media-temp/img/uploads/files/6559482958.pdf http://ledson.ru/upload\_picture/61391088476.pdf
fiho
kamailio call forwarding
http://savranoglugroup.com/dosyalar/files/9541552301.pdf