#-- 檔案排序 --#

Very easy with LINQ.

# -- To sort by name --#

1. var sorted = Directory.GetFiles(".").OrderBy(f => f);
2. Directory.GetFiles(folder, "\*.dll").Select(fn => new FileInfo(fn)). OrderBy(f => f.Name);
3. DirectoryInfo directoryInfo = new Directory(@"C:\");

List<FileInfo> list = directoryInfo.GetFiles().ToList();

list.Sort((a, b) => string.Compare(a.Name, b.Name));

# -- To sort by size --#

1. var sorted = Directory.GetFiles(".").OrderBy(f => new FileInfo(f).Length);
2. Directory.GetFiles(folder, "\*.dll").Select(fn => new FileInfo(fn)).

OrderBy(f => f.Length);

#-- To order by date --#

1. var orderedFiles = Directory.GetFiles("").Select(f=> new FileInfo(f)).OrderBy(f=> f.CreationTime)

|  |
| --- |
|  |

//-- 其他範例 --//

private static string formatFileNumberForSort(string inVal)

{

int o;

if (int.TryParse(Path.GetFileName(inVal), out o))

{

Console.WriteLine(string.Format("{0:0000000000}", o));

return string.Format("{0:0000000000}", o);

}

else

return inVal;

}

var fileNames = Directory.GetFiles("Spaceman").OrderBy(formatFileNumberForSort);

foreach(var s in fileNames)

Console.WriteLine("{0}", s);

using System;

using System.IO;

using System.Linq;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

// I have files called 1, 2, 10, 11, 20, 21 in my temp\test2 folder

var fileNames = Directory.GetFiles("c:\\temp\\test2").OrderBy(formatFileNumberForSort);

foreach(var s in fileNames)

Console.WriteLine("{0}", s);

}

private static string formatFileNumberForSort(string inVal)

{

int o;

if (int.TryParse(Path.GetFileName(inVal), out o))

{

Console.WriteLine(string.Format("{0:0000000000}", o));

return string.Format("{0:0000000000}", o);

}

else

return inVal;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

namespace ConsoleApplication7

{

class Program

{

static void Main(string[] args)

{

DirectoryInfo info = new DirectoryInfo("c:\\Snowman");

string[] fileNames = info.GetFiles().OrderBy(p => Path.GetFileNameWithoutExtension(p.Name),new ComparerN() )

.Select(fi => fi.Name).ToArray();

}

}

public class ComparerN : IComparer<string>

{

public int Compare(string sf, string ss)

{

if (IsNumer(sf) && IsNumer(ss))

{

if (Convert.ToInt32(sf) > Convert.ToInt32(ss))

{

return 1;

}

if (Convert.ToInt32(sf) < Convert.ToInt32(ss))

{

return -1;

}

if (Convert.ToInt32(sf) == Convert.ToInt32(ss))

{

return 0;

}

}

if (IsNumer(sf) && !IsNumer(ss))

return -1;

if (!IsNumer(sf) && IsNumer(ss))

return 1;

return string.Compare(sf, ss, true);

}

public static bool IsNumer(object v)

{

try

{

int it = Convert.ToInt32(v.ToString());

return true;

}

catch (FormatException){return false;}

}

}

}

* Jul 09 Sat 2011 03:49
* [**C# File類別與FileInfo類別**](http://byron0920.pixnet.net/blog/post/66998545)

**一.File類別與FileInfo類別:**  
      使用File類別進行檔案操作時，每次都會進行安全性檢查  
      FileInfo類別則只會於第一次進行安全性檢查，如果你需重複使用物件好幾次，請考慮使用 FileInfo 的執行個體方法，以取代 [File](http://msdn.microsoft.com/zh-tw/library/system.io.file.aspx) 類別的對應靜態 (Static) 方法，因為安全性檢查並不一定會需要。  
  
**二.指定路徑:**  
      C#語言下路徑指定方式有兩種:

1. 是使用兩個斜線，例如    "C:\\Test.txt"
2. 第二種是在路徑前加上@符號，例如    @"C:\Test.txt"

**三.FileInfo的檔案操作**  
    **EX：取得檔案資訊**  
        private void button1\_Click(object sender, EventArgs e)  
        {  
            //引用類別 FileInfo myInfo = new FileInfo(@"C:\newfile.txt");  
            //判斷檔案是否存在  
            if (myInfo.Exists)  
            {  
                //輸出檔名  
                Console.WriteLine("FileName:{0}", myInfo.Name);  
                //輸出路徑 Console.WriteLine("Path:{0}", myInfo.FullName);  
            }  
        }  
   
  
**EX：複製檔案**  
        private void button2\_Click(object sender, EventArgs e)  
        {  
            //引用類別  
            FileInfo myInfo = new FileInfo(@"C:\newfile.txt");  
            //判斷檔案是否存在  
            if (myInfo.Exists)  
            {  
                //複製檔案  
                myInfo.CopyTo(@"C:\newfile.bak");  
            }  
        }  
   
   
 **四.變更檔名**  
.net並沒有直接更名的函式所以必須以Copy或Move的方法處理，但是若採用Copy的方法，因為須在回頭刪除原始檔案，所以採用Move的方法會較好。  
   1.使用**File**(System.IO.File): **File.Move**(@"C:\From.txt", @"C:\TO.txt");  
   2.使用**FileInfo**(System.IO.FileInfo):**FileInfo.MoveTo**(@"C:new\_file.txt")

**C# FileInfo:**

You have learned how to perform different tasks on physical files using static File class in the previous section. Here, we will use FileInfo class to perform read/write operation on physical files.

The FileInfo class provides the same functionality as the static File class but you have more control on read/write operations on files by writing code manually for reading or writing bytes from a file.

**Important properties and methods of FileInfo:**

| **Property** | **Usage** | |
| --- | --- | --- |
| Directory | Gets an instance of the parent directory. | |
| DirectoryName | Gets a string representing the directory's full path. | |
| Exists | Gets a value indicating whether a file exists. | |
| Extension | Gets the string representing the extension part of the file. | |
| FullName | Gets the full path of the directory or file. | |
| IsReadOnly | Gets or sets a value that determines if the current file is read only. | |
| LastAccessTime | Gets or sets the time the current file or directory was last accessed | |
| LastWriteTime | Gets or sets the time when the current file or directory was last written to | |
| Length | Gets the size, in bytes, of the current file. | |
| Name | Gets the name of the file. | |
| **Method** | | **Usage** | |
| AppendText | | Creates a StreamWriter that appends text to the file represented by this instance of the FileInfo. | |
| CopyTo | | Copies an existing file to a new file, disallowing the overwriting of an existing file. | |
| Create | | Creates a file. | |
| CreateText | | Creates a StreamWriter that writes a new text file. | |
| Decrypt | | Decrypts a file that was encrypted by the current account using the Encrypt method. | |
| Delete | | Deletes the specified file. | |
| Encrypt | | Encrypts a file so that only the account used to encrypt the file can decrypt it. | |
| GetAccessControl | | Gets a FileSecurity object that encapsulates the access control list (ACL) entries for a specified file. | |
| MoveTo | | Moves a specified file to a new location, providing the option to specify a new file name. | |
| Open | | Opens a in the specified FileMode. | |
| OpenRead | | Creates a read-only FileStream. | |
| OpenText | | Creates a StreamReader with UTF8 encoding that reads from an existing text file. | |
| OpenWrite | | Creates a write-only FileStream. | |
| Replace | | Replaces the contents of a specified file with the file described by the current FileInfo object, deleting the original file, and creating a backup of the replaced file. | |
| ToString | | Returns a path as string. | |

The following example shows how to read bytes from a file manually and then convert them to a string using UTF8 encoding:

Example: Read file using FileInfo class

//Create object of FileInfo for specified path

FileInfo fi = new FileInfo(@"D:\DummyFile.txt");

//Open file for Read\Write

FileStream fs = fi.Open(FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.ReadWrite);

//create byte array of same size as FileStream length

byte[] fileBytes = new byte[fs.Length];

//define counter to check how much butes to read. Decrease the counter as you read each byte

int numBytesToRead = (int)fileBytes.Length;

//Counter to indicate number of bytes already read

int numBytesRead = 0;

//iterate till all the bytes read from FileStream

while (numBytesToRead > 0)

{

int n = fs.Read(fileBytes, numBytesRead, numBytesToRead);

if (n == 0)

break;

numBytesRead += n;

numBytesToRead -= n;

}

//Once you read all the bytes from FileStream, you can convert it into string using UTF8 encoding

string filestring = Encoding.UTF8.GetString(fileBytes);

As you have seen in the above code, you have to write lot of code for reading/writing a string from a FileSream. The same read/write operation can be done easily using StreamReader and StreamWriter.

The following example shows how StreamReader makes it easy to read strings from a file:

Example: Read file using StreamReader

//Create object of FileInfo for specified path

FileInfo fi = new FileInfo(@"D:\DummyFile.txt");

//Open file for Read\Write

FileStream fs = fi.Open(FileMode.OpenOrCreate, FileAccess.Read , FileShare.Read);

//Create object of StreamReader by passing FileStream object on which it needs to operates on

StreamReader sr = new StreamReader(fs);

//Use ReadToEnd method to read all the content from file

string fileContent = sr.ReadToEnd();

//Close StreamReader object after operation

sr.Close();

fs.Close();

Notice that fi.Open() has three parameters: The first parameter is FileMode for creating and opening a file if it does not exist; the second parameter, FileAccess, is to indicate a Read operation; and the third parameter is to share the file for reading with other users while the file is open.

The following example shows how StreamWriter makes it easy to write strings to a File:

Example: Write texts to file using StreamWriter

//Create object of FileInfo for specified path

FileInfo fi = new FileInfo(@"D:\DummyFile.txt");

//Open file for Read\Write

FileStream fs = fi.Open(FileMode.OpenOrCreate, FileAccess.Write, FileShare.Read );

//Create StreamWriter object to write string to FileSream

StreamWriter sw = new StreamWriter(fs);

sw.WriteLine("Another line from streamwriter");

sw.Close();

Read and Write operations are not possible on the same FileStream object simultaneously. If you are already reading from a file, create a separate FileStream object to write to the same file, as shown below:

Example: StreamReader & StreamWriter

//Create FileInfo object for DummyFile.txt

FileInfo fi = new FileInfo(@"D:\DummyFile.txt");

//open DummyFile.txt for read operation

FileStream fsToRead = fi.Open(FileMode.OpenOrCreate, FileAccess.ReadWrite , FileShare.ReadWrite);

//open DummyFile.txt for write operation

FileStream fsToWrite = fi.Open(FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.ReadWrite);

//get the StreamReader

StreamReader sr = new StreamReader(fsToRead);

//read all texts using StreamReader object

string fileContent = sr.ReadToEnd();

sr.Close();

//get the StreamWriter

StreamWriter sw = new StreamWriter(fsToWrite);

//write some text using StreamWriter

sw.WriteLine("Another line from streamwriter");

sw.Close();

//close all Stream objects

fsToRead.Close();

fsToWrite.Close();

Thus you can use FileInfo, StreamReader and StreamWriter class to read/write contents from physical file.

# Working with Files & Directories:

C# provides the following classes to work with the File system. They can be used to access directories, access files, open files for reading or writing, create a new file or move existing files from one location to another, etc.

| **Class Name** | **Usage** |
| --- | --- |
| File | File is a static class that provides different functionalities like copy, create, move, delete, open for reading or /writing, encrypt or decrypt, check if a file exists, append lines or text to a file’s content, get last access time, etc. |
| FileInfo | The FileInfo class provides the same functionality as a static File class. You have more control on how you do read/write operations on a file by writing code manually for reading or writing bytes from a file. |
| [Directory](http://msdn.microsoft.com/en-us/library/system.io.directory(v=vs.110).aspx) | Directory is a static class that provides functionality for creating, moving, deleting and accessing subdirectories. |
| [DirectoryInfo](http://msdn.microsoft.com/en-us/library/system.io.directoryinfo(v=vs.110).aspx) | DirectoryInfo provides instance methods for creating, moving, deleting and accessing subdirectories. |
| [Path](http://msdn.microsoft.com/en-us/library/system.io.path(v=vs.110).aspx) | Path is a static class that provides functionality such as retrieving the extension of a file, changing the extension of a file, retrieving the absolute physical path, and other path related functionalities. |

## File:

C# includes static **File** class to perform I/O operation on physical file system. The static File class includes various utility method to interact with physical file of any type e.g. binary, text etc.

Use this static File class to perform some quick operation on physical file. It is not recommended to use File class for multiple operations on multiple files at the same time due to performance reasons. Use FileInfo class in that scenario.

## Important Methods of Static File Class:

| **Method** | **Usage** |
| --- | --- |
| AppendAllLines | Appends lines to a file, and then closes the file. If the specified file does not exist, this method creates a file, writes the specified lines to the file, and then closes the file. |
| AppendAllText | Opens a file, appends the specified string to the file, and then closes the file. If the file does not exist, this method creates a file, writes the specified string to the file, then closes the 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. |
| Copy | Copies an existing file to a new file. Overwriting a file of the same name is not allowed. |
| Create | Creates or overwrites a file in the specified path. |
| CreateText | Creates or opens a file for writing UTF-8 encoded text. |
| Decrypt | Decrypts a file that was encrypted by the current account using the Encrypt method. |
| Delete | Deletes the specified file. |
| Encrypt | Encrypts a file so that only the account used to encrypt the file can decrypt it. |
| Exists | Determines whether the specified file exists. |
| GetAccessControl | Gets a FileSecurity object that encapsulates the access control list (ACL) entries for a specified file. |
| Move | Moves a specified file to a new location, providing the option to specify a new file name. |
| Open | Opens a FileStream on the specified path with read/write access. |
| ReadAllBytes | Opens a binary file, reads the contents of the file into a byte array, and then closes the file. |
| ReadAllLines | Opens a text file, reads all lines of the file, and then closes the file. |
| ReadAllText | Opens a text file, reads all lines of the file, and then closes the file. |
| Replace | Replaces the contents of a specified file with the contents of another file, deleting the original file, and creating a backup of the replaced file. |
| WriteAllBytes | Creates a new file, writes the specified byte array to the file, and then closes the file. If the target file already exists, it is overwritten. |
| WriteAllLines | Creates a new file, writes a collection of strings to the file, and then closes the file. |
| WriteAllText | Creates a new file, writes the specified string to the file, and then closes the file. If the target file already exists, it is overwritten. |

## Append text lines:

Use AppendAllLines() method to append multiple text lines to the specified file as shown below.

Example: Append all text lines to a file

string dummyLines = "This is first line." + Environment.NewLine +

"This is second line." + Environment.NewLine +

"This is third line.";

//Opens DummyFile.txt and append lines. If file is not exists then create and open.

File.AppendAllLines(@"C:\DummyFile.txt", dummyLines.Split(Environment.NewLine.ToCharArray()).ToList<string>());

## Append text string:

Use *File.AppendAllText()* method to append string to a file in single line of code as shown below.

Example: Append string to a file

//Opens DummyFile.txt and append Text. If file is not exists then create and open.

File.AppendAllText(@"C:\ DummyFile.txt", "This is File testing");

## Overwrite existing texts:

Use *File.WriteAllText()* method to write texts to the file. Please note that it will not append text but overwrite existing texts.

Example: Overwrite existing texts

//Opens DummyFile.txt and write texts. If file is not exists then create and open.

File.WriteAllText(@"C:\DummyFile.txt", "This is dummy text");

The following example shows how to perform different operations using static File class.

Example: Multiple File operations

//Check whether file is exists or not at particular location

bool isFileExists = File.Exists(@"C:\ DummyFile.txt"); // returns false

//Copy DummyFile.txt as new file DummyFileNew.txt

File.Copy(@"C:\DummyFile.txt", @"D:\NewDummyFile.txt");

//Get when the file was accessed last time

DateTime lastAccessTime = File.GetLastAccessTime(@"C:\DummyFile.txt");

//get when the file was written last time

DateTime lastWriteTime = File.GetLastWriteTime(@"C:\DummyFile.txt");

// Move file to new location

File.Move(@"C:\DummyFile.txt", @"D:\DummyFile.txt");

//Open file and returns FileStream for reading bytes from the file

FileStream fs = File.Open(@"D:\DummyFile.txt", FileMode.OpenOrCreate);

//Open file and return StreamReader for reading string from the file

StreamReader sr = File.OpenText(@"D:\DummyFile.txt");

//Delete file

File.Delete(@"C:\DummyFile.txt");

Thus, it is easy to work with physical file using static File class. However, if you want more flexibility then use FileInfo class. The same way, use static Directory class to work with physical directories.



### Points to Remember :

1. File is a static class to read\write from physical file with less coding.
2. Static File class provides functionalities such as create, read\write, copy, move, delete and others for physical files.
3. Static Directory class provides functionalities such as create, copy, move, delete etc for physical directories with less coding.
4. FileInfo and DirectoryInfo class provides same functionality as static File and Directory class.

# C# Stream:

C# includes following standard IO (Input/Output) classes to read/write from different sources like a file, memory, network, isolated storage, etc.

**Stream:** *System.IO.Stream* is an abstract class that provides standard methods to transfer bytes (read, write, etc.) to the source. It is like a wrapper class to transfer bytes. Classes that need to read/write bytes from a particular source must implement the Stream class.

The following classes inherits Stream class to provide functionality to Read/Write bytes from a particular source:

**FileStream** reads or writes bytes from/to a physical file whether it is a .txt, .exe, .jpg or any other file. FileStream is derived from the Stream class.

**MemoryStream:** MemoryStream reads or writes bytes that are stored in memory.

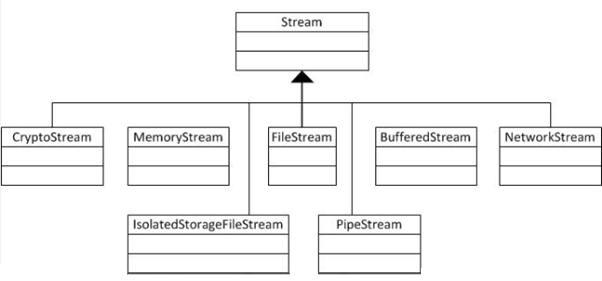
**BufferedStream:** BufferedStream reads or writes bytes from other Streams to improve the performance of certain I/O operations.

**NetworkStream:** NetworkStream reads or writes bytes from a network socket.

**PipeStream:** PipeStream reads or writes bytes from different processes.

**CryptoStream:** CryptoStream is for linking data streams to cryptographic transformations.

The following diagram shows the hierarchy of stream classes:

[](http://www.tutorialsteacher.com/Content/images/csharp/stream-heirarchy.png)Stream Classes Hierarchy

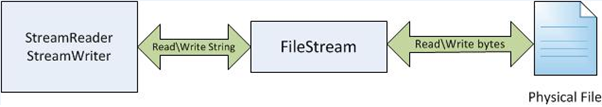
## Readers and Writers:

**StreamReader**: StreamReader is a helper class for reading characters from a Stream by converting bytes into characters using an encoded value. It can be used to read strings (characters) from different Streams like FileStream, MemoryStream, etc.

**StreamWriter**: StreamWriter is a helper class for writing a string to a Stream by converting characters into bytes. It can be used to write strings to different Streams such as FileStream, MemoryStream, etc.

**BinaryReader**: BinaryReader is a helper class for reading primitive datatype from bytes.

**BinaryWriter**: BinaryWriter writes primitive types in binary.

[](http://www.tutorialsteacher.com/Content/images/csharp/stream-relations.png)Stream IO

The above image shows that FileStream reads bytes from a physical file and then StreamReader reads strings by converting those bytes to strings. In the same way, StreamWriter takes a string and converts it into bytes and writes to FileStream and then FileStream writes the bytes to a physical file. So FileStream deals with bytes where as StreamReader & StreamWriter deals with strings.



### Points to Remember :

1. **Stream** is an abstract class for transfering bytes from different sources. It is base class for all other class that reads\writes bytes to different sources.
2. **FileStream** class provides reading and writing functionality of bytes to physical file.
3. Reader & writer classes provides functionality to read bytes from Stream classes (FileStream, MemoryStream etc) and converts bytes into appropriate encoding.
4. StreamReader provides a helper method to read string from FileStream by converting bytes into strings. StreamWriter provides a helper method to write string to FileStream by converting strings into bytes.

Learn how to read/write to the File system in the next section.

**Path 類別**

在含有檔案或目錄路徑資訊的 [String](https://msdn.microsoft.com/zh-tw/library/system.string(v=vs.110).aspx) 執行個體上執行作業。 這些作業是以跨平台方式來執行的。

若要瀏覽此類型的.NET Framework 原始程式碼，請參閱 [Reference Source](http://referencesource.microsoft.com/#mscorlib/system/io/path.cs#090eca8621a248ee)。

**命名空間:**   [System.IO](https://msdn.microsoft.com/zh-tw/library/system.io(v=vs.110).aspx)  
**組件:**  mscorlib (於 mscorlib.dll)

**繼承階層**

[System.Object](https://msdn.microsoft.com/zh-tw/library/system.object(v=vs.110).aspx)  
  System.IO.Path

**語法**

C#

[ComVisibleAttribute(true)]

public static class Path

**方法**

|  |  |  |
| --- | --- | --- |
|  | **名稱** | **描述** |
|  | [ChangeExtension(String, String)](https://msdn.microsoft.com/zh-tw/library/system.io.path.changeextension(v=vs.110).aspx) | 變更路徑字串的副檔名。 |
|  | [Combine(String, String)](https://msdn.microsoft.com/zh-tw/library/fyy7a5kt(v=vs.110).aspx) | 將兩個字串合併為一個路徑。 |
|  | [Combine(String, String, String)](https://msdn.microsoft.com/zh-tw/library/dd784047(v=vs.110).aspx) | 將三個字串合併為一個路徑。 |
|  | [Combine(String, String, String, String)](https://msdn.microsoft.com/zh-tw/library/dd782933(v=vs.110).aspx) | 將四個字串合併為一個路徑。 |
|  | [Combine(String[])](https://msdn.microsoft.com/zh-tw/library/dd991142(v=vs.110).aspx) | 將一個字串陣列合併為單一路徑。 |
|  | [GetDirectoryName(String)](https://msdn.microsoft.com/zh-tw/library/system.io.path.getdirectoryname(v=vs.110).aspx) | 傳回指定路徑字串的目錄資訊。 |
|  | [GetExtension(String)](https://msdn.microsoft.com/zh-tw/library/system.io.path.getextension(v=vs.110).aspx) | 傳回指定路徑字串的副檔名。 |
|  | [GetFileName(String)](https://msdn.microsoft.com/zh-tw/library/system.io.path.getfilename(v=vs.110).aspx) | 傳回指定路徑字串的檔案名稱和副檔名。 |
|  | [GetFileNameWithoutExtension(String)](https://msdn.microsoft.com/zh-tw/library/system.io.path.getfilenamewithoutextension(v=vs.110).aspx) | 傳回沒有副檔名的指定路徑字串的檔案名稱。 |
|  | [GetFullPath(String)](https://msdn.microsoft.com/zh-tw/library/system.io.path.getfullpath(v=vs.110).aspx) | 傳回指定路徑字串的絕對路徑。 |
|  | [GetInvalidFileNameChars()](https://msdn.microsoft.com/zh-tw/library/system.io.path.getinvalidfilenamechars(v=vs.110).aspx) | 取得陣列，該陣列包含檔案名稱中不允許的字元。 |
|  | [GetInvalidPathChars()](https://msdn.microsoft.com/zh-tw/library/system.io.path.getinvalidpathchars(v=vs.110).aspx) | 取得陣列，該陣列包含路徑名稱中不允許的字元。 |
|  | [GetPathRoot(String)](https://msdn.microsoft.com/zh-tw/library/system.io.path.getpathroot(v=vs.110).aspx) | 取得指定路徑的根目錄資訊。 |
|  | [GetRandomFileName()](https://msdn.microsoft.com/zh-tw/library/system.io.path.getrandomfilename(v=vs.110).aspx) | 傳回隨機資料夾名稱或檔案名稱。 |
|  | [GetTempFileName()](https://msdn.microsoft.com/zh-tw/library/system.io.path.gettempfilename(v=vs.110).aspx) | 在磁碟上建立具名之零位元組的唯一暫存檔案，然後傳回該檔案的完整路徑。 |
|  | [GetTempPath()](https://msdn.microsoft.com/zh-tw/library/system.io.path.gettemppath(v=vs.110).aspx) | 傳回目前使用者的暫存資料夾的路徑。 |
|  | [HasExtension(String)](https://msdn.microsoft.com/zh-tw/library/system.io.path.hasextension(v=vs.110).aspx) | 判斷路徑是否包括副檔名。 |
|  | [IsPathRooted(String)](https://msdn.microsoft.com/zh-tw/library/system.io.path.ispathrooted(v=vs.110).aspx) | 取得值，該值指出指定的路徑字串是否包含根目錄。 |

**欄位**

|  |  |  |
| --- | --- | --- |
|  | **名稱** | **描述** |
|  | [AltDirectorySeparatorChar](https://msdn.microsoft.com/zh-tw/library/system.io.path.altdirectoryseparatorchar(v=vs.110).aspx) | 提供平台特定替代字元，用以在反映階層式檔案系統組織的路徑字串中分隔目錄層級。 |
|  | [DirectorySeparatorChar](https://msdn.microsoft.com/zh-tw/library/system.io.path.directoryseparatorchar(v=vs.110).aspx) | 提供平台特定字元，用以在反映階層式檔案系統組織的路徑字串中分隔目錄層級。 |
|  | [InvalidPathChars](https://msdn.microsoft.com/zh-tw/library/system.io.path.invalidpathchars(v=vs.110).aspx) | **已淘汰。** 提供字元的平台專屬陣列，這些字元不可以在傳遞給 Path 類別成員的路徑字串引數中指定。 |
|  | [PathSeparator](https://msdn.microsoft.com/zh-tw/library/system.io.path.pathseparator(v=vs.110).aspx) | 平台特定分隔符號字元，用來分隔環境變數中的路徑字串。 |
|  | [VolumeSeparatorChar](https://msdn.microsoft.com/zh-tw/library/system.io.path.volumeseparatorchar(v=vs.110).aspx) | 提供平台特定磁碟區分隔符號字元。 |

**註解**

|  |
| --- |
| **注意** |
| 若要檢視此類型的.NET Framework 原始程式碼，請參閱 [Reference Source](http://referencesource.microsoft.com/#mscorlib/system/io/path.cs#090eca8621a248ee)。 您可以瀏覽原始碼、 下載的離線，參考並逐步執行 （包含修補程式和更新），在偵錯。see [instructions](http://referencesource.microsoft.com/). |

.NET Framework 不支援直接存取實體磁碟，透過裝置的名稱，例如 「 \\.\PHYSICALDRIVE0 」 的路徑。

路徑是字串，提供檔案或目錄的位置。 路徑不一定是指向磁碟; 上的位置例如，路徑可能對應至記憶體中或在裝置上的位置。 路徑的確切的格式取決於目前的平台。 例如，某些系統路徑可以開始磁碟機或磁碟區的代號，而這個項目不會在其他系統。 某些系統檔案的路徑可以包含擴充功能，可指示儲存在檔案中的資訊類型。 副檔名的檔案名稱的格式是平台相依。比方說，有些系統限制為三個字元，擴充功能，部分則否。 目前的平台也會決定一組字元，用來分隔的路徑，項目和一組指定的路徑時，無法使用的字元。 由於這些差異，欄位的 **Path** 類別的某些成員的實際行為及 **Path** 類別會依平台而異。

路徑可以包含絕對或相對位置資訊。 絕對路徑完整指定的位置︰ 檔案或目錄可以唯一識別目前的位置無關。 相對路徑指定部分位置︰ 尋找指定的相對路徑的檔案時，使用目前的位置做為起點。 若要判斷目前的目錄，請呼叫 [Directory.GetCurrentDirectory](https://msdn.microsoft.com/zh-tw/library/system.io.directory.getcurrentdirectory(v=vs.110).aspx)。

大部分 **Path** 類別不會使用檔案系統互動，而不會驗證指定的路徑字串的檔案是否存在。 **Path** 類別成員，以修改路徑的字串，例如 [ChangeExtension](https://msdn.microsoft.com/zh-tw/library/system.io.path.changeextension(v=vs.110).aspx), ，在檔案系統中有不會影響檔案的名稱。 **Path** 成員執行，不過，驗證指定的路徑字串的內容，則擲回 [ArgumentException](https://msdn.microsoft.com/zh-tw/library/system.argumentexception(v=vs.110).aspx) 例外狀況，如果字串包含無效的路徑字串中所定義的字元從傳回的字元 [GetInvalidPathChars](https://msdn.microsoft.com/zh-tw/library/system.io.path.getinvalidpathchars(v=vs.110).aspx) 方法。 例如，在 Windows 架構桌面平台無效路徑字元可能包含引號 （"）、 小於 (<)、 大於 (>)、 垂直線 (|)、 退格鍵 (\b)、 null (\0) 和 16-18 到 20 到 25 的 Unicode 字元。

成員 **Path** 類別可讓您快速且輕鬆地執行一般作業，例如判斷副檔名是否為路徑的一部分，並將兩個字串結合成一個路徑名稱。

所有成員 **Path** 類別都是靜態的因此不需要路徑的執行個體呼叫。

|  |
| --- |
| **注意** |
| 在接受路徑，做為輸入字串的成員，該路徑必須是格式正確或發生例外狀況。 比方說，如果路徑是完整名稱，但開頭有空格，路徑中類別的方法不修剪。 因此，路徑的格式不正確，而且會引發例外狀況。 同樣地，路徑的組合無法完整限定兩次。 例如，"c:\temp c:\windows 」 也會引發例外狀況在大部分情況下。 請確定您的路徑是語式正確的當您使用接受路徑字串的方法。 |

在接受路徑的成員，路徑可以參考到檔案或只是一個目錄。 指定的路徑也可以參考的相對路徑或通用命名慣例 (UNC) 路徑的伺服器和共用名稱。 例如，下列所有條件都是可接受的路徑︰

* 在 C# 中的 「 c:\\MyDir\\MyFile.txt 」 或者 「 c:\MyDir\MyFile.txt 」 在 Visual Basic 中。
* 在 C# 中的 「 c:\\MyDir 」 或者 「 c:\MyDir 」 在 Visual Basic 中。
* 「 MyDir\\MySubdir 」 以 C# 或 Visual Basic 中的 「 MyDir\MySubDir 」。
* 在 C# 中的 「 \\\MyServer\\MyShare 」 或者 「 \\MyServer\MyShare 」 在 Visual Basic 中。

因為所有這些作業會執行字串，就無法驗證結果在所有案例中有效。 例如， [GetExtension](https://msdn.microsoft.com/zh-tw/library/system.io.path.getextension(v=vs.110).aspx) 方法會剖析字串傳遞給它，並傳回該字串的副檔名。 不過，這不表示該副檔名的檔案存在磁碟上。

如需一般 I/O 工作的清單，請參閱 [一般 I/O 工作](https://msdn.microsoft.com/zh-tw/library/ms404278(v=vs.110).aspx)。

**範例**

下列程式碼範例示範一些主要成員 **Path** 類別。

C#

using System;

using System.IO;

class Test

{

public static void Main()

{

string path1 = @"c:\temp\MyTest.txt";

string path2 = @"c:\temp\MyTest";

string path3 = @"temp";

if (Path.HasExtension(path1))

{

Console.WriteLine("{0} has an extension.", path1);

}

if (!Path.HasExtension(path2))

{

Console.WriteLine("{0} has no extension.", path2);

}

if (!Path.IsPathRooted(path3))

{

Console.WriteLine("The string {0} contains no root information.", path3);

}

Console.WriteLine("The full path of {0} is {1}.", path3, Path.GetFullPath(path3));

Console.WriteLine("{0} is the location for temporary files.", Path.GetTempPath());

Console.WriteLine("{0} is a file available for use.", Path.GetTempFileName());

/\* This code produces output similar to the following:

\* c:\temp\MyTest.txt has an extension.

\* c:\temp\MyTest has no extension.

\* The string temp contains no root information.

\* The full path of temp is D:\Documents and Settings\cliffc\My Documents\Visual Studio 2005\Projects\ConsoleApplication2\ConsoleApplication2\bin\Debug\temp.

\* D:\Documents and Settings\cliffc\Local Settings\Temp\8\ is the location for temporary files.

\* D:\Documents and Settings\cliffc\Local Settings\Temp\8\tmp3D.tmp is a file available for use.

\*/

}

}