Directory.GetFiles Method

Reference

Definition

Namespace: System.IO

Assembly: System.Runtime.dll

Returns the names of files that meet specified criteria.

Overloads

Expand table

GetFiles(String)	Returns the names of files (including their paths) in the specified directory.
GetFiles(String, String)	Returns the names of files (including their paths) that match the specified search pattern in the specified directory.
GetFiles(String, String, EnumerationOptions)	Returns the names of files (including their paths) that match the specified search pattern and enumeration options in the specified directory.
GetFiles(String, String, SearchOption)	Returns the names of files (including their paths) that match the specified search pattern in the specified directory, using a value to determine whether to search subdirectories.

GetFiles(String)

Source: Directory.cs

Returns the names of files (including their paths) in the specified directory.

C#

```
public static string[] GetFiles (string path);
```

Parameters

path String

The relative or absolute path to the directory to search. This string is not case-sensitive.

Returns

String[]

An array of the full names (including paths) for the files in the specified directory, or an empty array if no files are found.

Exceptions

IOException

`path` is a file name.

-or-

A network error has occurred.

UnauthorizedAccessException

The caller does not have the required permission.

ArgumentException

.NET Framework and .NET Core versions older than 2.1: `path` is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the GetInvalidPathChars() method.

ArgumentNullException

`path` is `null`.

PathTooLongException

The specified path, file name, or both exceed the system-defined maximum length.

DirectoryNotFoundException

The specified path is not found or is invalid (for example, it is on an unmapped drive).

Examples

The following example demonstrates how to use the GetFiles method to return file names from a user-specified location. The example is configured to catch all errors common to this method.

C#

```
// For Directory.GetFiles and Directory.GetDirectories
// For File.Exists, Directory.Exists
using System;
using System.IO;
using System.Collections;

public class RecursiveFileProcessor
{
    public static void Main(string[] args)
    {
        foreach(string path in args)
        {
            if(File.Exists(path))
            {
                // This path is a file
                ProcessFile(path);
            }
            else if(Directory.Exists(path))
            {
                // This path is a directory
                ProcessDirectory(path);
            }
            else
            {
                Console.WriteLine("{0} is not a valid file or di-
rectory.", path);
            }
        }
    }

    // Process all files in the directory passed in, recurse on
any directories
    // that are found, and process the files they contain.
    public static void ProcessDirectory(string targetDirectory)
    {
        // Process the list of files found in the directory.
        string [] fileEntries =
Directory.GetFiles(targetDirectory);
        foreach(string fileName in fileEntries)
            ProcessFile(fileName);

        // Recurse into subdirectories of this directory.
        string [] subdirectoryEntries =
Directory.GetDirectories(targetDirectory);
        foreach(string subdirectory in subdirectoryEntries)
            ProcessDirectory(subdirectory);
    }

    // Insert logic for processing found files here.
    public static void ProcessFile(string path)
    {
        Console.WriteLine("Processed file '{0}'.", path);
        }
    }
```

Remarks

The EnumerateFiles and GetFiles methods differ as follows: When you use EnumerateFiles, you can start enumerating the collection of names before the whole collection is returned; when you use GetFiles, you must wait for the whole array of names to be returned before you can access the array. Therefore, when you are working with many files and directories, EnumerateFiles can be more efficient.

The returned file names are appended to the supplied `path` parameter.

This method is identical to GetFiles(String, String) with the asterisk (*) specified as the search pattern.

The `path` parameter can specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see GetCurrentDirectory.

The order of the returned file names is not guaranteed; use the Sort method if a specific sort order is required.

The case-sensitivity of the `path` parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

For a list of common I/O tasks, see Common I/O Tasks.

See also

- File and Stream I/O
- How to: Read Text from a File
- How to: Write Text to a File

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

GetFiles(String, String)

Source: Directory.cs

Returns the names of files (including their paths) that match the specified search pattern in the specified directory.

```
C#

public static string[] GetFiles (string path, string
searchPattern);
```

Parameters

`path` String

The relative or absolute path to the directory to search. This string is not case-sensitive.

`searchPattern` String

The search string to match against the names of files in `path`. This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.

Returns

String[]

An array of the full names (including paths) for the files in the specified directory that match the specified search pattern, or an empty array if no files are found.

Exceptions

IOException

`path` is a file name.

-or-

A network error has occurred.

UnauthorizedAccessException

The caller does not have the required permission.

ArgumentException

.NET Framework and .NET Core versions older than 2.1: `path` is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using GetInvalidPathChars().

-or-

`searchPattern` doesn't contain a valid pattern.

ArgumentNullException

`path` or `searchPattern` is `null`.

PathTooLongException

The specified path, file name, or both exceed the system-defined maximum length.

DirectoryNotFoundException

The specified path is not found or is invalid (for example, it is on an unmapped drive).

Examples

The following example counts the number of files that begin with the specified letter.

C#

```
using System;
using System.IO;

class Test
{
    public static void Main()
    {
        try
        {
            // Only get files that begin with the letter "c".
            string[] dirs = Directory.GetFiles(@"c:\", "c*");
            Console.WriteLine("The number of files starting with c
is {0}.", dirs.Length);
```

```
                foreach (string dir in dirs)
                {
                    Console.WriteLine(dir);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("The process failed: {0}", e.To-
String());
            }
        }
    }
```

Remarks

The returned file names are appended to the supplied `path` parameter and the order of the returned file names is not guaranteed; use the Sort method if a specific sort order is required.

`searchPattern` can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in `searchPattern`.

Expand table

Wildcard specifier	Matches
* (asterisk)	Zero or more characters in that position.
? (question mark)	Exactly one character in that position.

Characters other than the wildcard are literal characters. For example, the `searchPattern` string "*t" searches for all names in `path` ending with the letter "t". The `searchPattern` string "s*" searches for all names in `path` beginning with the letter "s".

`searchPattern` cannot end in two periods ("..") or contain two periods ("..") followed by DirectorySeparatorChar or AltDirectorySeparatorChar, nor can it contain any invalid characters. You can query for invalid characters by using the GetInvalidPathChars method.

ⓘ Note

.NET Framework only: When you use the asterisk wildcard character in `searchPattern` and you specify a three-character file extension, for example, "*.txt", this method also returns files with extensions that *begin* with the specified extension. For example, the search pattern "*.xls" returns both "book.xls" and "book.xlsx". This behavior only occurs if an asterisk is used in the search pattern and the file extension provided is exactly three characters. If you use the question mark wildcard character somewhere in the search pattern, this method returns only files that match the specified file extension exactly. The following table depicts this anomaly in .NET Framework.

Expand table

Files in directory	Search pattern	.NET 5+ returns	.NET Framework returns
file.ai, file.aif	*.ai	file.ai	file.ai
book.xls, book.xlsx	*.xls	book.xls	**book.xls, book.xlsx**
ello.txt, hello.txt, hello.txtt	?ello.txt	hello.txt	hello.txt

ⓘ Note

Because this method checks against file names with both the 8.3 file name format and the long file name format, a search pattern similar to "*1*.txt" may return unexpected file names. For example, using a search pattern of "*1*.txt" returns "longfilename.txt" because the equivalent 8.3 file name format is "LONGFI~1.TXT".

The EnumerateFiles and GetFiles methods differ as follows: When you use EnumerateFiles, you can start enumerating the collection of names before the whole collection is returned; when you use GetFiles, you must wait for the whole array of names to be returned before you can access the array. Therefore, when you are working with many files and directories, EnumerateFiles can be more efficient.

The `path` parameter can specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see GetCurrentDirectory.

The case-sensitivity of the `path` parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default

Windows file system) and case-sensitive on Linux file systems.

For a list of common I/O tasks, see Common I/O Tasks.

See also

- File and Stream I/O
- How to: Read Text from a File
- How to: Write Text to a File

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

GetFiles(String, String, EnumerationOptions)

Source: Directory.cs

Returns the names of files (including their paths) that match the specified search pattern and enumeration options in the specified directory.

C#

```
public static string[] GetFiles (string path, string searchPat-
tern, System.IO.EnumerationOptions enumerationOptions);
```

Parameters

`path` String

The relative or absolute path to the directory to search. This string is not case-sensitive.

`searchPattern` String

The search string to match against the names of files in `path`. This parameter can contain a combination of valid literal and wildcard characters, but it doesn't support regular expressions.

`enumerationOptions` EnumerationOptions

An object that describes the search and enumeration configuration to use.

Returns

String[]

An array of the full names (including paths) for the files in the specified directory that match the specified search pattern and enumeration options, or an empty array if no files are found.

Exceptions

IOException

`path` is a file name.

-or-

A network error has occurred.

UnauthorizedAccessException

The caller does not have the required permission.

ArgumentException

.NET Framework and .NET Core versions older than 2.1: `path` is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using GetInvalidPathChars().

-or-

`searchPattern` doesn't contain a valid pattern.

ArgumentNullException

`path` or `searchPattern` is `null`.

PathTooLongException

The specified path, file name, or both exceed the system-defined maximum length.

DirectoryNotFoundException

The specified path is not found or is invalid (for example, it is on an unmapped drive).

Remarks

The returned file names are appended to the supplied `path` parameter and the order of the returned file names is not guaranteed; use the Sort method if a specific sort order is required.

`searchPattern` can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in `searchPattern`.

Expand table

Wildcard specifier	Matches
* (asterisk)	Zero or more characters in that position.
? (question mark)	Exactly one character in that position.

Characters other than the wildcard are literal characters. For example, the `searchPattern` string "*t" searches for all names in `path` ending with the letter "t". The `searchPattern` string "s*" searches for all names in `path` beginning with the letter "s".

`searchPattern` cannot end in two periods ("..") or contain two periods ("..") followed by DirectorySeparatorChar or AltDirectorySeparatorChar, nor can it contain any invalid characters. You can query for invalid characters by using the GetInvalidPathChars method.

ⓘ Note

.NET Framework only: When you use the asterisk wildcard character in `searchPattern` and you specify a three-character file extension, for example, "*.txt", this method also returns files with extensions that *begin* with the specified extension. For example, the search pattern "*.xls" returns both "book.xls" and "book.xlsx". This behavior only occurs if an asterisk is used in the search pattern and the file extension provided is exactly three characters. If you use the question mark wildcard character somewhere in the search pattern, this method returns only files that match the specified file extension exactly. The following table depicts this anomaly in .NET Framework.

Expand table

Files in directory	Search pattern	.NET 5+ returns	.NET Framework returns
file.ai, file.aif	*.ai	file.ai	file.ai
book.xls, book.xlsx	*.xls	book.xls	**book.xls, book.xlsx**
ello.txt, hello.txt, hello.txtt	?ello.txt	hello.txt	hello.txt

Note

Because this method checks against file names with both the 8.3 file name format and the long file name format, a search pattern similar to "*1*.txt" may return unexpected file names. For example, using a search pattern of "*1*.txt" returns "longfilename.txt" because the equivalent 8.3 file name format is "LONGFI~1.TXT".

The EnumerateFiles and GetFiles methods differ as follows: When you use EnumerateFiles, you can start enumerating the collection of names before the whole collection is returned; when you use GetFiles, you must wait for the whole array of names to be returned before you can access the array. Therefore, when you are working with many files and directories, EnumerateFiles can be more efficient.

The `path` parameter can specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see GetCurrentDirectory.

The case-sensitivity of the `path` parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

For a list of common I/O tasks, see Common I/O Tasks.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

GetFiles(String, String, SearchOption)

Source: Directory.cs

Returns the names of files (including their paths) that match the specified search pattern in the specified directory, using a value to determine whether to search subdirectories.

C#

```
public static string[] GetFiles (string path, string searchPat-
tern, System.IO.SearchOption searchOption);
```

Parameters

`path` String

The relative or absolute path to the directory to search. This string is not case-sensitive.

`searchPattern` String

The search string to match against the names of files in `path`. This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.

`searchOption` SearchOption

One of the enumeration values that specifies whether the search operation should include all subdirectories or only the current directory.

Returns

String[]

An array of the full names (including paths) for the files in the specified directory that match the specified search pattern and option, or an empty array if no files are found.

Exceptions

ArgumentException

.NET Framework and .NET Core versions older than 2.1: `path` is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters with the GetInvalidPathChars() method.

-or-

`searchPattern` does not contain a valid pattern.

ArgumentNullException

`path` or `searchpattern` is `null`.

ArgumentOutOfRangeException

`searchOption` is not a valid SearchOption value.

UnauthorizedAccessException

The caller does not have the required permission.

DirectoryNotFoundException

The specified path is not found or is invalid (for example, it is on an unmapped drive).

PathTooLongException

The specified path, file name, or both exceed the system-defined maximum length.

IOException

`path` is a file name.

-or-

A network error has occurred.

Remarks

The returned file names are appended to the supplied parameter `path` and the order of the returned file names is not guaranteed; use the Sort method if a specific sort order is required.

`searchPattern` can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in `searchPattern`.

Expand table

Wildcard specifier	Matches
* (asterisk)	Zero or more characters in that position.
? (question mark)	Exactly one character in that position.

Characters other than the wildcard are literal characters. For example, the `searchPattern` string "*t" searches for all names in `path` ending with the letter "t". The `searchPattern` string "s*" searches for all names in `path` beginning with the letter "s".

`searchPattern` cannot end in two periods ("..") or contain two periods ("..") followed by DirectorySeparatorChar or AltDirectorySeparatorChar, nor can it contain any invalid characters. You can query for invalid characters by using the GetInvalidPathChars method.

ⓘ Note

.NET Framework only: When you use the asterisk wildcard character in `searchPattern` and you specify a three-character file extension, for example, "*.txt", this method also returns files with extensions that *begin* with the specified extension. For example, the search pattern "*.xls" returns both "book.xls" and "book.xlsx". This behavior only occurs if an asterisk is used in the search pattern and the file extension provided is exactly three characters. If you use the question mark wildcard character somewhere in the search pattern, this method returns only files that match the specified file extension exactly. The following table depicts this anomaly in .NET Framework.

Expand table

Files in directory	Search pattern	.NET 5+ returns	.NET Framework returns
file.ai, file.aif	*.ai	file.ai	file.ai
book.xls, book.xlsx	*.xls	book.xls	**book.xls, book.xlsx**
ello.txt, hello.txt, hello.txtt	?ello.txt	hello.txt	hello.txt

ⓘ Note

Because this method checks against file names with both the 8.3 file name format and the long file name format, a search pattern similar to "*1*.txt" may return unexpected file names. For example, using a search pattern of "*1*.txt" returns "longfilename.txt" because the equivalent 8.3 file name format is "LONGFI~1.TXT".

The EnumerateFiles and GetFiles methods differ as follows: When you use EnumerateFiles, you can start enumerating the collection of names before the whole collection is returned; when you use GetFiles, you must wait for the whole array of names to be returned before you can access the array. Therefore, when you are working with many files and directories, EnumerateFiles can be more efficient.

The file names include the full path.

The `path` parameter can specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see GetCurrentDirectory.

The case-sensitivity of the `path` parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

For a list of common I/O tasks, see Common I/O Tasks.

See also

- File and Stream I/O
- How to: Read Text from a File
- How to: Write Text to a File

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
Framework	
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0