Directory.GetParent(String) Method

Reference

Definition

Namespace: System.IO

Assembly: System.Runtime.dll

Source: Directory.cs

Retrieves the parent directory of the specified path, including both absolute and relative paths.

C#

```
public static System.IO.DirectoryInfo? GetParent (string path);
```

Parameters

`path` String

The path for which to retrieve the parent directory.

Returns

DirectoryInfo

The parent directory, or `null` if `path` is the root directory, including the root of a UNC server or share name.

Exceptions

IOException

The directory specified by `path` is read-only.

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 with the GetInvalidPathChars() method.

ArgumentNullException

`path` is `null`.

PathTooLongException

The specified path, file name, or both exceed the system-defined maximum length. For more information, see the PathTooLongException topic.

DirectoryNotFoundException

The specified path was not found.

NotSupportedException

`path` is in an invalid format.

SecurityException

.NET Framework only: The caller does not have the required permissions.

Examples

The following example demonstrates how to use the GetParent method to retrieve the parent directory of a user-specified location, "path". The value returned by the GetParent method is then printed to the console. The example is configured to catch all errors common to this method.

C#

```
using System;

namespace GetFileSystemEntries
{
    class Class1
    {
        static void Main(string[] args)
        {
            Class1 snippets = new Class1();

            string path = System.IO.Directory.GetCurrentDirectory();
            string filter = "*.exe";

            snippets.PrintFileSystemEntries(path);
            snippets.PrintFileSystemEntries(path, filter);
            snippets.GetLogicalDrives();
            snippets.GetParent(path);
            snippets.Move("C:\\proof", "C:\\Temp");
        }

        void PrintFileSystemEntries(string path)
```

```
        {

            try
            {
                // Obtain the file system entries in the directory
path.
                string[] directoryEntries =
                    System.IO.Directory.GetFileSystemEntries(path);

                foreach (string str in directoryEntries)
                {
                    System.Console.WriteLine(str);
                }
            }
            catch (ArgumentNullException)
            {
                System.Console.WriteLine("Path is a null
reference.");
            }
            catch (System.Security.SecurityException)
            {
                System.Console.WriteLine("The caller does not have
the " +
                    "required permission.");
            }
            catch (ArgumentException)
            {
                System.Console.WriteLine("Path is an empty string, "
+
                    "contains only white spaces, " +
                    "or contains invalid characters.");
            }
            catch (System.IO.DirectoryNotFoundException)
            {
                System.Console.WriteLine("The path encapsulated in
the " +
                    "Directory object does not exist.");
            }
        }
        void PrintFileSystemEntries(string path, string pattern)
        {
            try
            {
                // Obtain the file system entries in the directory
                // path that match the pattern.
                string[] directoryEntries =
                    System.IO.Directory.GetFileSystemEntries(path,
pattern);

                foreach (string str in directoryEntries)
                {
                    System.Console.WriteLine(str);
                }
            }
            catch (ArgumentNullException)
```

```
            {
                System.Console.WriteLine("Path is a null
reference.");
            }
            catch (System.Security.SecurityException)
            {
                System.Console.WriteLine("The caller does not have
the " +
                    "required permission.");
            }
            catch (ArgumentException)
            {
                System.Console.WriteLine("Path is an empty string, "
+
                    "contains only white spaces, " +
                    "or contains invalid characters.");
            }
            catch (System.IO.DirectoryNotFoundException)
            {
                System.Console.WriteLine("The path encapsulated in
the " +
                    "Directory object does not exist.");
            }
        }

        // Print out all logical drives on the system.
        void GetLogicalDrives()
        {
            try
            {
                string[] drives =
System.IO.Directory.GetLogicalDrives();

                foreach (string str in drives)
                {
                    System.Console.WriteLine(str);
                }
            }
            catch (System.IO.IOException)
            {
                System.Console.WriteLine("An I/O error occurs.");
            }
            catch (System.Security.SecurityException)
            {
                System.Console.WriteLine("The caller does not have
the " +
                    "required permission.");
            }
        }
        void GetParent(string path)
        {
            try
            {
                System.IO.DirectoryInfo directoryInfo =
                    System.IO.Directory.GetParent(path);
```

```
                System.Console.WriteLine(directoryInfo.FullName);
            }
            catch (ArgumentNullException)
            {
                System.Console.WriteLine("Path is a null
reference.");
            }
            catch (ArgumentException)
            {
                System.Console.WriteLine("Path is an empty string, "
+
                    "contains only white spaces, or " +
                    "contains invalid characters.");
            }
        }
        void Move(string sourcePath, string destinationPath)
        {
            try
            {
                System.IO.Directory.Move(sourcePath,
destinationPath);
                System.Console.WriteLine("The directory move is com-
plete.");
            }
            catch (ArgumentNullException)
            {
                System.Console.WriteLine("Path is a null
reference.");
            }
            catch (System.Security.SecurityException)
            {
                System.Console.WriteLine("The caller does not have
the " +
                    "required permission.");
            }
            catch (ArgumentException)
            {
                System.Console.WriteLine("Path is an empty string, "
+
                    "contains only white spaces, " +
                    "or contains invalid characters.");
            }
            catch (System.IO.IOException)
            {
                System.Console.WriteLine("An attempt was made to move
a " +
                    "directory to a different " +
                    "volume, or destDirName " +
                    "already exists.");
            }
        }
    }
}
```

Remarks

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.

Trailing spaces are removed from the end of the `path` parameter before getting the directory.

The string returned by this method consists of all characters in the path up to, but not including, the last DirectorySeparatorChar or AltDirectorySeparatorChar. For example, passing the path "C:\Directory\SubDirectory\test.txt" to GetParent returns "C:\Directory\SubDirectory". Passing "C:\Directory\SubDirectory" returns "C:\Directory". However, passing "C:\Directory\SubDirectory\" returns "C:\Directory\SubDirectory", because the ending directory separator is after "SubDirectory".

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

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

See also

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

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback