How do I determine whether the filesystem is case-sensitive in .net?

Asked 15 years, 11 months ago Modified 1 month ago Viewed 7k times

15

Does .net have a way to determine whether the local filesystem is case-sensitive?

.net linux mono filesystems case-sensitive

Share
Improve this question
Follow

edited Oct 21, 2009 at 19:34
Kip
109k ● 87 ● 238 ● 268

asked Jan 10, 2009 at 0:30
Matt
32.3k ● 4 ● 35 ● 33

10 Answers

Sorted by: Highest score (default)

16

You can create a file in the temp folder (using lowercase filename), then check if the file exists (using uppercase filename), e.g:

```
string file = Path.GetTempPath() + Guid.NewGuid().ToSt
File.CreateText(file).Close();
```

```
bool isCaseInsensitive = File.Exists(file.ToUpper());
File.Delete(file);
```

Share Improve this answer Follow

edited Jan 10, 2009 at 0:50

answered Jan 10, 2009 at 0:37

M4N
96.4k ● 45 ● 224 ● 264

3 Good answer. There is an extremely small chance the GUID created contains none of 'abcdef'. It's not very likely but it could happen! – Mitch Wheat Jan 10, 2009 at 1:01

2 How about similarly a creating a temp folder, creating a known lowercase filename in it, and then testing with the uppercase name? – Mitch Wheat Jan 10, 2009 at 1:03

1 Good comment, Mitch (the first one). You could add "a" to the filename to be sure it contains at least one letter. – M4N Jan 10, 2009 at 1:13

16 How do you know the temp folder is in the same filesystem for which you wish to find out the case sensitivity? – Dan Gravell Aug 26, 2011 at 12:58

2 Why not use `typeof(CurrentClass).Assembly.Location`? Double-check that the current assembly file exists, and then see if it exists with its casing changed. *Extremely* likely to answer correctly, without the need to create any files. – Timo Nov 13, 2018 at 12:25

7

Keep in mind that you might have multiple file systems with different casing rules. For example, the root filesystem could be case-sensitive, but you can have a case-insensitive filesystem (e.g. an USB stick with a FAT filesystem on it) mounted somewhere. So if you do such checks, make sure that you make them in the directory that you are going to access.

Also, what if the user copies the data from say a case-sensitive to a case-insensitive file system? If you have files that differ only by case, one of them will overwrite the other, causing data loss. When copying in the other direction, you might also run into problems, for example, if file A contains a reference to file "b", but the file is actually named "B". This works on the original case-insensitive file system, but not on the case-sensitive system.

Thus I would suggest that you avoid depending on whether the file system is case-sensitive or not if you can. Do not generate file names that differ only by case, use the standard file picker dialogs, be prepared that the case might change, etc.

Share Improve this answer Follow

answered Jan 10, 2009 at 9:50

oefe
19.9k ● 7 ● 48 ● 66

There is no such a function in the .NET Class Library.

6

You can, however, roll out your own: Try creating a file with a lowercase name and then try to open it with the upparcase version of its name. Probably it is possible to improve this method, but you get the idea.

EDIT: You could actually just take the first file in the root directory and then check if both filename.ToLower() and filename.ToUpper() exist. Unfortunately it is quite possible that both uppercase and lowercase variants of the same file exist, so you should compare the FileInfo.Name properties of both the lowercase and uppercase variants to see if they are indeed the same or not. This will not require writing to the disk.

Obviously, this will fail if there are no files at all on the volume. In this case, just fall back to the first option (see Martin's answer for the implementation).

Share Improve this answer Follow

edited Jan 10, 2009 at 15:10

answered Jan 10, 2009 at 0:34

Tamas Czinege
121k ● 40 ● 153 ● 177

This could fail, if the first existing file, exists in both lower- and uppercase variants on a case-sensitive filesystem.
– M4N Jan 10, 2009 at 0:38

Martin: true. I will modify my answer accordingly.
– Tamas Czinege Jan 10, 2009 at 0:46

2 This will not work as fileInfo.Name and fileInfo.FullName will retain the case that was sent into the constructor it will not get the case of the actual filename!! – Peter Jul 29, 2012 at 22:39

4

It's not a .NET function, but the GetVolumeInformation and GetVolumeInformationByHandleW functions from the Windows API will do what you want (see yje lpFileSystemFlags parameter.

Share Improve this answer Follow

answered Apr 20, 2010 at 22:31
Paul Moore
7,189 ● 6 ● 45 ● 49

1 Sorry, old answer, but my tests seem to indicate then NTFS is case sensitive (the resulting FileSystemFlags has FILE_CASE_SENSITIVE_SEARCH). So am I doing something wrong? – Mr Lister Jun 12, 2013 at 14:53

@MrLister Sorry, old comment, but check this answer - NTFS is indeed case sensitive but Windows itself is not. – Bip901 Jan 29, 2022 at 15:14

3

How about this heuristic?

```
public static bool IsCaseSensitiveFileSystem() {
   var tmp = Path.GetTempPath();
   return !Directory.Exists(tmp.ToUpper()) || !Directo
}
```

Share Improve this answer

Follow

edited Jan 23, 2019 at 19:39

answered Jan 23, 2019 at 19:30

eirik

708 ● 3 ● 11

2

There are actually two ways to interpret the original question.

1. How to determine whether a specific file system is able to preserve case-sensitivity in file names?
2. How to determine whether the current operating system interprets file names case-sensitively when working with a specific file system.

This answer is based on the second interpretation, because I think that is what the OP wanted to know and also what matters to most people.

The following code is loosely based on M4N's and Nicolas Raoul's answer and attempts to create a really robust implementation that is able to determine whether the operating system handles file names case-sensitive inside of a specified directory (excluding sub-directories, since these could be mounted from another file system).

It works by creating two new files in succession, one with lower-case, the other one with upper-case characters. The files are locked exclusively and are deleted automatically when closed. This should avert any negative side effects caused by creating files. Of course, this implementation only works if the specified directory exists and the current user is able to create files inside of it.

The code is written for .NET Framework 4.0 and C# 7.2 (or later).

```
using System;
using System.IO;
using System.Reflection;

/// <summary>
/// Check whether the operating system handles file na
specified directory.
/// </summary>
/// <param name="directoryPath">The path to the direct
/// <returns>A value indicating whether the operating
case-sensitive in the specified directory.</returns>
/// <exception cref="ArgumentNullException"><paramref
null.</exception>
/// <exception cref="ArgumentException"><paramref name
contains one or more invalid characters.</exception>
/// <exception cref="DirectoryNotFoundException">The s
not exist.</exception>
/// <exception cref="UnauthorizedAccessException">The
permission to the specified directory.</exception>
private static bool IsFileSystemCaseSensitive(string d
{
    if (directoryPath == null)
    {
        throw new ArgumentNullException(nameof(directo
    }

    while (true)
    {
        string fileNameLower = ".cstest." + Guid.NewGu
        string fileNameUpper = fileNameLower.ToUpperIn
        string filePathLower = Path.Combine(directoryP
        string filePathUpper = Path.Combine(directoryP

        FileStream fileStreamLower = null;
        FileStream fileStreamUpper = null;
        try
        {
            try
            {
                // Try to create filePathUpper to ensu
file.
                fileStreamUpper = new FileStream(fileP
FileMode.CreateNew, FileAccess.Write, FileShare.None,
FileOptions.DeleteOnClose);

                // After ensuring that it didn't exist
must be closed/deleted again to ensure correct opening
regardless of the case-sensitivity of the file system.
                // On case-sensitive file systems ther
race condition, where another process could create fil
closing/deleting it here and newly creating it after f
                // This method would then incorrectly
insensitive file system.
                fileStreamUpper.Dispose();
            }
            catch (IOException ioException) when
(IsErrorFileExists(ioException))
            {
                // filePathUpper already exists, try a
                continue;
            }

            try
            {
                fileStreamLower = new FileStream(fileP
FileMode.CreateNew, FileAccess.Write, FileShare.None,
FileOptions.DeleteOnClose);
            }
            catch (IOException ioException) when
(IsErrorFileExists(ioException))
            {
                // filePathLower already exists, try a
                continue;
            }
            try
            {
                fileStreamUpper = new FileStream(fileP
FileMode.CreateNew, FileAccess.Write, FileShare.None,
FileOptions.DeleteOnClose);

                // filePathUpper does not exist, this
sensitivity
                return true;
            }
            catch (IOException ioException) when
(IsErrorFileExists(ioException))
            {
                // fileNameUpper already exists, this
insensitivity
                return false;
            }
        }
        finally
        {
            fileStreamLower?.Dispose();
            fileStreamUpper?.Dispose();
        }
    }
}

/// <summary>
/// Determines whether the specified <see cref="IOExce
"file exists" error.
/// </summary>
/// <param name="ioException">The <see cref="IOExcepti
/// <returns>A value indicating whether the specified
indicates a "file exists" error.</returns>
private static bool IsErrorFileExists(IOException ioEx
{
    //
https://referencesource.microsoft.com/mscorlib/microso
    const int ERROR_FILE_EXISTS = 0x50;

    // The Exception.HResult property's get accessor i
4.5, need to get its value via reflection.
    int hresult = (int)typeof(Exception)
        .GetProperty("HResult", BindingFlags.Instance
BindingFlags.NonPublic)
        .GetValue(ioException, index: null);

    //
https://referencesource.microsoft.com/mscorlib/microso
    return hresult == unchecked((int)0x80070000 | ERRO
}
```

As you can see there is a tiny possibility for a race condition which can cause a false-negative. If this race condition is something you really worry about I suggest you do the check a second time when the result is false, either inside the `IsFileSystemCaseSensitive` method or outside of it. However, in my opinion, the probability of encountering this race condition once, let alone two times in a row, is astronomically small.

Share Improve this answer

Follow

answered Nov 9, 2018 at 14:53

Stefan Podskubka

556 ●1 ●3 ●10

2

Here is the approach that does not use temporary files:

```
using System;
using System.Runtime.InteropServices;

static bool IsCaseSensitive()
{
    if (RuntimeInformation.IsOSPlatform(OSPlatform.Win
        RuntimeInformation.IsOSPlatform(OSPlatform.OSX
file-system) is usually configured to be case insensit
    {
        return false;
    }
```

```
        else if (RuntimeInformation.IsOSPlatform(OSPlatfor
        {
            return true;
        }
        else if (Environment.OSVersion.Platform == Platfor
        {
            return true;
        }
        else
        {
           // A default.
           return false;
        }
}
```

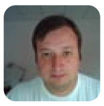
Instead, it contains an ingrained knowledge about operating environments.

Readily available as NuGet package, works on everything .NET and is updated on a regular basis: https://github.com/gapotchenko/Gapotchenko.FX/tree/main/Source/Modules/Catalog/Gapotchenko.FX.IO#iscasesensitive

Share Improve this answer Follow

edited Nov 13 at 10:48

answered Jun 26, 2019 at 13:25



ogggre
2,266 ● 1 ● 23 ● 19

1 Note: IIRC, HFS+ has the option to enable case sensitivity when the partition is created. Additionally, Linux can mount ntfs and fat partitions. This would cause a false positive in those cases. – Sir Intellegence-BrainStormexe Sep 13, 2021 at 15:28

That also apply to APFS (the HFS+ part). Stack overflow isn't letting me edit my original comment after confirming that. – Sir Intellegence-BrainStormexe Sep 13, 2021 at 15:36

Under Windows one can mount virtual drives created by utilities which make available cloud filesystems like FTP, Google Drive, etc. These filesystems are again case-sensitive. – Roland Pihlakas Jan 10, 2022 at 14:21

1

Try creating a temporary file in all lowercase, and then check if it exists using uppercase.

Share Improve this answer Follow

answered Jan 10, 2009 at 0:33

ScottS
8,543 ● 5 ● 31 ● 50

what about abc.xyz and aBc.xyz – masfenix Jan 10, 2009 at 0:47

0

```
/// <summary>
/// Check whether the operating system is case-sensiti
/// For instance on Linux you can have two files/folde
//// "test" and "TEST", but on Windows the two can not
/// This method does not extend to mounted filesystems
different properties.
/// </summary>
/// <returns>true if the operating system is case-sens
public static bool IsFileSystemCaseSensitive()
{
    // Actually try.
    string file = Path.GetTempPath() + Guid.NewGuid().
"test";
    File.CreateText(file).Close();
    bool result = ! File.Exists(file.ToUpper());
    File.Delete(file);

    return result;
}
```

Based on M4N's answer, with the following changes:

- Static names so that we are sure it contains a letter and not only numbers.
- Maybe more readable?
- Wrapped in a method.
- Documentation.

A better strategy would be to take a path as an argument, and create the file on the same filesystem, but writing there might have unexpected consequences.

Share Improve this answer Follow

edited Nov 8, 2018 at 9:07

answered Jul 25, 2016 at 3:46

Nicolas Raoul
60.1k ● 63 ● 235 ● 389

-2

I invoke The Cheat:

```
Path.DirectorySeparatorChar == '\\' ? "I'm insensitive
sensitive"
```

Share Improve this answer
Follow

answered Dec 5, 2017 at 17:32

Jeremy Cook
22k ● 11 ● 73 ● 78

1 **Caution**: This is not true on macOS for HFS+ nor for APFS (unless explicitly requested when creating the file system).
– piksel bitworks Nov 17, 2018 at 15:49