How can I get the application's path in a .NET console application?

Ask Question

866

How do I find the application's path in a console application?

In Windows Forms, I can use `Application.StartupPath` to find the current path, but this doesn't seem to be available in a console application.

168

c# .net console console-application

edited Nov 20 '13 at 15:40

Peter Mortensen
13.9k 19 87 113

asked May 7 '09 at 23:05

JSmyth
4,860 3 19 18

5 Do you install .NET Framework on target (Client, Development) machine? if your answer is true; So, you can add a reference to System.Windows.Forms.dll and use Application.StartupPath! This is the best way if you want to drop away further future exceptions! – Ehsan Mohammadi Mar 13 '15 at 2:23

AppDomain.BaseDirectory is app directory. Be aware that application can behave different in VS env and Win env. But AppDomain should be same not as application.path but i hope that this is not only for IIS. – Mertuarez Apr 29 '16 at 13:11

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

26 Answers

1090

`System.Reflection.Assembly.GetExecutingAssembly()` . `Location` [1]

Combine that with `System.IO.Path.GetDirectoryName` if all you want is the directory.

> [1] *As per Mr.Mindor's comment:*
> `System.Reflection.Assembly.GetExecutingAssembly().Location` returns where the executing assembly is currently located, which may or may not be where the assembly is located when not executing. In the case of shadow copying assemblies, you will get a path in a temp directory.
> `System.Reflection.Assembly.GetExecutingAssembly().CodeBase` will return the 'permanent' path of the assembly.

edited Mar 14 at 8:48

Sebastian Brosch
25.8k 12 42 55

answered May 7 '09 at 23:09

Sam Axe
27.3k 7 45 73

233 System.Reflection.Assembly.GetExecutingAssembly().Location returns where the executing assembly is **currently** located, which may or may not be where the assembly is located when not executing. In the case of shadow copying assemblies, you will get a path in a temp directory. System.Reflection.Assembly.GetExecutingAssembly().CodeBase will return the '**permenant**' path of the assembly. – Mr.Mindor Oct 14 '11 at 18:13

12 @SamGoldberg: That depend on how it is used: stackoverflow.com/q/1068420/391656 . Or you can ... new

Uri(System.Reflection.Assembly.GetExecutingAssembly().CodeBase).LocalPath – Mr.Mindor Aug 30 '12 at 21:31

2 For whatever reason, in VS2013 (at least the copy I have) intellisense does not work past System.Reflection.Assembly. GetExeuctingAssembly() is there but you cannot see it. – PseudoToad Apr 25 '14 at 21:13

23 `GetExecutingAssembly` returns assembly that **contains the code that is currently executing**. This may not necessarily be the console **.exe** assembly. It may be an assembly that has been loaded from a totally different location. You will have to use `GetEntryAssembly` ! Also note that `CodeBase` might not be set when the assembly is in the GAC. The better alternative is `AppDomain.CurrentDomain.BaseDirectory` . – bitbonk Aug 6 '15 at 14:41

2 Please write code in 4 spaces so it is comfortable to copy – fnc12 Nov 27 '16 at 16:06

0

You can use following code, you will get application full path:

```
 class Program
{
    static void Main(string[] args)
    {
        string AppPath = System.Reflection.Assembly.GetExecutingAsse
        WriteLine($"ApplicationPath ---{AppPath}");
        //OutPut// //ApplicationPath---
C:\Users\TestUser\source\repos\ThreadStack\ThreadStack\bin\Debug\Thr
        ReadLine();
    }
}
```

edited Oct 24 '18 at 8:49

ayaio
59k 20 135 189

answered Oct 24 '18 at 8:48

Sunil Dhappadhule


91 6

5

in VB.net

```
My.Application.Info.DirectoryPath
```

works for me (Application Type: Class Library). Not sure about C#...
Returns the path w/o Filename as string

edited May 16 '18 at 2:45


Nicolas Raoul
34.3k 47 163 292

answered Apr 12 '15 at 12:08


dba
381 1 5 14

0

There are many ways to get executable path, which one we should use it depends on our needs here is a link which discuss different methods.

Different ways to get Application Executable Path

answered Apr 3 '18 at 7:26


Nasir Mahmood
1,057 1 10 17

Following line will give you an application path:

4

```
var applicationPath =
Path.GetDirectoryName(Process.GetCurrentProcess().MainModule.FileNam
```

Above solution is working properly in following situations:

- simple app
- in another domain where Assembly.GetEntryAssembly() would return null
- DLL is loaded from Embedded resources as byte array and loaded to AppDomain as Assembly.Load(byteArrayOfEmbeddedDll)

edited Feb 23 '18 at 20:32

answered Feb 23 '18 at 19:24

user2126375

912 8 17

0

I didn't see anyone convert the LocalPath provided by .Net Core reflection into a usable System.IO path so here's my version.

```
public static string GetApplicationRoot()
{
   var exePath = new Uri(System.Reflection.
   Assembly.GetExecutingAssembly().CodeBase).LocalPath;

   return new FileInfo(exePath).DirectoryName;

}
```

This will return the full "C:\xxx\xxx" formatted path to where your code is.

answered Feb 7 '18 at 19:03

mark gamache

152 1 9

12

If you are looking for a .NET Core compatible way, use

```
System.AppContext.BaseDirectory
```

This was introduced in .NET Framework 4.6 and .NET Core 1.0 (and .NET Standard 1.3). See: AppContext.BaseDirectory Property.

According to this page,

> This is the prefered replacement for AppDomain.CurrentDomain.BaseDirectory in .NET Core

answered Feb 2 '18 at 2:14

Dejan

3,227 1 31 60

2

Try this simple line of code:

```
string exePath = Path.GetDirectoryName( Application.ExecutablePath)
```

answered Nov 20 '17 at 9:42

daniele3004

6,073 7 40 52

2

None of these methods work in special cases like using a symbolic link to the exe, they will return the location of the link not the actual exe.

So can use QueryFullProcessImageName to get around that:

```
using System;
using System.IO;
using System.Runtime.InteropServices;
using System.Text;
using System.Diagnostics;

internal static class NativeMethods
{
    [DllImport("kernel32.dll", SetLastError = true)]
    internal static extern bool QueryFullProcessImageName([In]IntPtr
dwFlags, [Out]StringBuilder lpExeName, ref int lpdwSize);

    [DllImport("kernel32.dll", SetLastError = true)]
    internal static extern IntPtr OpenProcess(
        UInt32 dwDesiredAccess,
        [MarshalAs(UnmanagedType.Bool)]
        Boolean bInheritHandle,
        Int32 dwProcessId
    );
}

public static class utils
{

    private const UInt32 PROCESS_QUERY_INFORMATION = 0x400;
    private const UInt32 PROCESS_VM_READ = 0x010;

    public static string getfolder()
    {
        Int32 pid = Process.GetCurrentProcess().Id;
        int capacity = 2000;
        StringBuilder sb = new StringBuilder(capacity);
        IntPtr proc;

        if ((proc = NativeMethods.OpenProcess(PROCESS_QUERY_INFORMAT
PROCESS_VM_READ, false, pid)) == IntPtr.Zero)
            return "";
```

```
            NativeMethods.QueryFullProcessImageName(proc, 0, sb, ref cap
            string fullPath = sb.ToString(0, capacity);
            return Path.GetDirectoryName(fullPath) + @"\";
        }
    }
```

answered Feb 5 '17 at 8:54

colin lamarre

1,363 12 20

15

I have used this code and get the solution.

```
AppDomain.CurrentDomain.BaseDirectory
```

answered Jan 31 '17 at 8:17

Trimantra Software Solution

2,558 1 14 30

8

You can simply add to your project references `System.Windows.Forms` and then use the `System.Windows.Forms.Application.StartupPath` as usual .

So, not need for more complicated methods or using the reflection.

edited May 9 '16 at 12:27

answered Apr 26 '16 at 10:26

fruggiero
735 8 18

I used that one, and it works well. But one time I used the method having it in my unit test project. And of course, it failed because it was looking for my file in C:\PROGRAM FILES (X86)\MICROSOFT VISUAL STUDIO 14.0\COMMON7\IDE\COMMONEXTENSIONS\MICROSOFT\TESTWINDOW – ainasiart Aug 31 '17 at 19:09

-1

Here is a reliable solution that works with **32bit** and **64bit** applications.

Add these references:

> using System.Diagnostics;
>
> using System.Management;

Add this method to your project:

```
public static string GetProcessPath(int processId)
{
    string MethodResult = "";
    try
    {
        string Query = "SELECT ExecutablePath FROM Win32_Process WHEI
processId;

        using (ManagementObjectSearcher mos = new ManagementObjectSe
        {
            using (ManagementObjectCollection moc = mos.Get())
            {
                string ExecutablePath = (from mo in moc.Cast<Managem
mo["ExecutablePath"]).First().ToString();

                MethodResult = ExecutablePath;

            }
```

```
            }

        }
        catch //(Exception ex)
        {
            //ex.HandleException();
        }
        return MethodResult;
    }
```

Now use it like so:

```
int RootProcessId = Process.GetCurrentProcess().Id;

GetProcessPath(RootProcessId);
```

Notice that if you know the id of the process, then this method will return the corresponding ExecutePath.

Extra, for those interested:

```
Process.GetProcesses()
```

...will give you an array of all the currently running processes, and...

```
Process.GetCurrentProcess()
```

...will give you the current process, along with their information e.g. Id, etc. and also limited control e.g. Kill, etc.*

edited Mar 9 '16 at 10:56

answered Mar 9 '16 at 10:35

Knickerless-Noggins

5,331 3 47 59

you can use this one instead.

23

```
System.Environment.CurrentDirectory
```

edited Sep 16 '15 at 13:34

Matas Vaitkevicius

34.4k 16 169 176

answered Nov 15 '12 at 20:06

ButtShock

279 2 2

This will get the folder of the executable though – Iain Nov 15 '12 at 20:27

21 Unless something has called Directory.SetCurrentDirectory()... – Matthew Watson Nov 21 '12 at 14:21

4 @MatthewWatson or even simpler, the program was executed from another folder... – Ohad Schneider Aug 29 '17 at 16:58

Probably a bit late but this is worth a mention:

76

```
Environment.GetCommandLineArgs()[0];
```

Or more correctly to get just the directory path:

```
System.IO.Path.GetDirectoryName(Environment.GetCommandLineArgs()[0])
```

Edit:

Quite a few people have pointed out that `GetCommandLineArgs` is not guaranteed to return the program name. See [The first word on the command line is the program name only by convention](). The article does state that "Although extremely few Windows programs use this quirk (I am not aware of any myself)". So it is possible to 'spoof' `GetCommandLineArgs` , but we are talking about a console application. Console apps are usually quick and dirty. So this fits in with my KISS philosophy.

edited Aug 27 '15 at 9:17

shA.t
13.1k 4 38 71

answered May 21 '11 at 13:27

Steve Mc
2,747 18 32

And maybe this: path = Environment.GetCommandLineArgs()[0].Substring(0, iniFilePath.LastIndexOf("\\") + 1); – fabspro Oct 9 '11 at 14:00

1 @usr the situation you allude to is highly theoretical. In the context of a console application, it doesn't really make sense to use any other method. Keep it simple! – Steve Mc Jul 21 '12 at 10:32

1 @usr mmm - looking at the taskmgr cmdline column sort of backs up what I'm saying. A few system services with just the exe name. Never mind. What I'm trying to say is that when developing a console application there is no need to make things more complicated than they need to be. Especially when we already have the information available. Now, if you are running a console application in such a way as to trick GetCommandLineArgs then you are already jumping through hoops and you would probably need to ask yourself if a console app is the right way to go. – Steve Mc Jul 22 '12 at 8:54

5 Your "simple" solution involves two method calls. The "complicated" solution involves two method calls. No practical difference - except that the "simple" solution can give you the wrong answer under certain circumstances which aren't under your control when you're writing the program. Why take the risk? Use the other two method calls, and your

program will be no more complicated but will be more reliable. – Chris Feb 22 '13 at 14:53

3 Worked for my scenario, the other solutions did not, so thanks for providing another alternative :-) I was using ReSharper test runner to run an MS Unit test and the code I was testing needed a specific .dll to be in the executing directory...and Assembly.GetExecutingDirectory() weirdly returns a different result. – wallismark Mar 4 '15 at 6:27

141

+50

You have two options for finding the directory of the application, which you chose will depend on your purpose.

```
// to get the location the assembly is executing from
//(not necessarily where the it normally resides on disk)
// in the case of the using shadow copies, for instance in NUnit tes
// this will be in a temp directory.
string path = System.Reflection.Assembly.GetExecutingAssembly().Loca

//To get the location the assembly normally resides on disk or the i
string path = System.Reflection.Assembly.GetExecutingAssembly().Codel

//once you have the path you get the directory with:
var directory = System.IO.Path.GetDirectoryName(path);
```

edited Aug 27 '15 at 9:16

shA.t

13.1k 4 38 71

answered Oct 14 '11 at 18:27

Mr.Mindor

3,244 1 14 21

1 Just wanted to say, obviously there are many more than 2 options by how many other choices are posted... – vapcguy Jun 27 '16 at 17:23

13 If whatever you're trying to do with said path doesn't support URI format, use `var localDirectory = new Uri(directory).LocalPath;` –

Okuma.Scott Sep 27 '16 at 13:25

This is just wrong. What is the executable is not a .NET assembly at all? The right answer is to check the environment and inspect the command line. – mark Apr 13 '18 at 13:17

@Ukuma.Scott This doesn't work if the path contains & or # – MatsW Feb 7 at 9:12

7

I mean, why not a p/invoke method?

```
    using System;
    using System.IO;
    using System.Runtime.InteropServices;
    using System.Text;
    public class AppInfo
    {
            [DllImport("kernel32.dll", CharSet = CharSet.Auto, Exact
            private static extern int GetModuleFileName(HandleRef hM
buffer, int length);
            private static HandleRef NullHandleRef = new HandleRef(n
            public static string StartupPath
            {
                get
                {
                    StringBuilder stringBuilder = new StringBuilder(
                    GetModuleFileName(NullHandleRef, stringBuilder,
stringBuilder.Capacity);
                    return Path.GetDirectoryName(stringBuilder.ToStr
                }
            }
    }
```

You would use it just like the Application.StartupPath:

```
    Console.WriteLine("The path to this executable is: " + AppInfo.S
System.Diagnostics.Process.GetCurrentProcess().ProcessName + ".exe")
```

answered Jul 24 '14 at 20:56

user3596865
121 1 2

1 Why p/invoke when there is so much .NET for this? – ProfK Mar 3 '15 at 18:57

3 Because why not? – user3596865 Mar 25 '15 at 23:00

6 @user3596865 because it requries a hard dependency to Windows and is not compatible with DNX or Mono. And maybe there is a breaking change in future Windows Versions. So again: why we should use pinvoke here? – Ben Jan 20 '16 at 10:05

5

```
AppDomain.CurrentDomain.BaseDirectory
```

Will resolve the issue to refer the 3rd party reference files with installation packages.

answered Jul 14 '14 at 6:43

Nirav Mehta
69 1 4

11 This answer has already been suggested 5 years ago, even more than once. – P-L Aug 14 '14 at 19:05

20

For Console Applications, you can try this:

```
System.IO.Directory.GetCurrentDirectory();
```

Output (on my local machine):

> c:\users\xxxxxxx\documents\visual studio 2012\Projects\ImageHandler\GetDir\bin\Debug

Or you can try (there's an additional backslash in the end):

```
AppDomain.CurrentDomain.BaseDirectory
```

Output:

> c:\users\xxxxxxx\documents\visual studio 2012\Projects\ImageHandler\GetDir\bin\Debug\

edited Jun 18 '14 at 22:03

answered Jun 18 '14 at 21:34

F.Alves
538 5 8

8

I have used

```
System.AppDomain.CurrentDomain.BaseDirectory
```

when I want to find a path relative to an applications folder. This works for both ASP.Net and winform applications. It also does not require any reference to System.Web assemblies.

answered Apr 7 '14 at 16:57

user2346593
81 2 1

382

You can use the following code to get the current application directory.

```
AppDomain.CurrentDomain.BaseDirectory
```

edited Nov 22 '13 at 12:39

Richard Everett
33.2k 49 166 261

answered May 8 '09 at 19:03

Sarathy

34 Don't use this. The BaseDirectory can be set at runtime. It is not guaranteed to be correct (like the accepted answer is). – usr Jul 15 '12 at 20:29

3 +1 This is likely the answer you want as it compensates for shadow copying. – George Mauer May 8 '14 at 23:42

4 @usr What makes you think that `BaseDirectory` can be set at runtime? It only has a getter. – bitbonk Aug 6 '15 at 14:43

2 @bitbonk it can be set at appdomain creation time. – usr Aug 6 '15 at 14:43

1 Isn't it that BaseDirectory can be changed in a *.lnk file, in the "Start in:" field? – Alexander Jun 15 '16 at 14:46

27

You may be looking to do this:

```
System.IO.Path.GetDirectoryName(
    System.Reflection.Assembly.GetExecutingAssembly().GetName().Codel
```

edited Nov 20 '13 at 15:41

Peter Mortensen
13.9k 19 87 113

answered May 7 '09 at 23:10

PSU_Kardi
2,076 3 33 66

I use this if the exe is supposed to be called by double clicking it

8

```
var thisPath = System.IO.Directory.GetCurrentDirectory();
```

edited Sep 10 '13 at 22:07

answered Feb 15 '13 at 18:19

developer747
5,865 19 64 123

4 This is not correct because you can get random directories in result. – amuliar Jul 3 '13 at 10:58

this Command returns Environment.CurrentDirectory, which may be changed at runtime to any path, so it is not a reliable solution. – Yury Kozlov Jan 11 at 3:41

`Assembly.GetEntryAssembly().Location` or
`Assembly.GetExecutingAssembly().Location`

5

Use in combination with `System.IO.Path.GetDirectoryName()` to get only the directory.

The paths from `GetEntryAssembly()` and `GetExecutingAssembly()` can be different, even though for most cases the directory will be the same.

With `GetEntryAssembly()` you have to be aware that this can return `null` if the entry module is unmanaged (ie C++ or VB6 executable). In those cases it is possible to use `GetModuleFileName` from the Win32 API:

```
[DllImport("kernel32.dll", CharSet = CharSet.Auto)]
public static extern int GetModuleFileName(HandleRef hModule, String
length);
```

edited Jun 11 '12 at 15:40

answered Mar 29 '12 at 12:41

Herman
1,322 10 24

-5

You can create a folder name as Resources within the project using Solution Explorer,then you can paste a file within the Resources.

```
private void Form1_Load(object sender, EventArgs e) {
    string appName = Environment.CurrentDirectory;
    int l = appName.Length;
    int h = appName.LastIndexOf("bin");
    string ll = appName.Remove(h);
    string g = ll + "Resources\\sample.txt";
    System.Diagnostics.Process.Start(g);
}
```

edited May 28 '12 at 11:59

Perception
69k 10 150 174

answered May 28 '12 at 11:36

Devarajan.T
23 1

6 Using Environment.CurrentDirectory is very wrong, don't use this! this path can change at runtime. Even at startup it is non-deterministic. – usr Jul 15 '12 at 20:28

36

The answer above was 90% of what I needed, but returned a Uri instead of a regular path for me.

As explained in the MSDN forums post, How to convert URI path to normal filepath?, I used the following:

```
// Get normal filepath of this assembly's permanent directory
var path = new Uri(
    System.IO.Path.GetDirectoryName(
        System.Reflection.Assembly.GetExecutingAssembly().CodeBase)
    ).LocalPath;
```

answered Apr 13 '12 at 20:20

fizzled
538 4 5

this works well also if the exe in question is a windows service and current directory returns C:\Windows\system32. The above code returns the actual location of the exe – DalmTo Feb 20 '15 at 11:32

Except if you then try to do something like `File.CreateDirectory(path)` , it will give you the exception that it doesn't allow URI paths... – vapcguy Jun 27 '16 at 17:26

1 Unfortunately this is doesn't work for paths that contain a fragment identifier (the `#` character). The identifier and everything following it is truncated from the resulting path. – Mark Feb 2 '18 at 8:07

Why don't you swap `new Uri` and `System.IO.Path.GetDirectoryName` ? That gives you a normal path string instead of a `Uri` . – Timo Nov 5 '18 at 16:50

I find this the best one. This same approach has worked reliably for me in any environment. In production, debugging locally, unit testing... Want to open a content file that you included ("content - copy if newer") in a unit test? It's there. – Timo Nov 5 '18 at 16:54

43

For anyone interested in asp.net web apps. Here are my results of 3 different methods

```
protected void Application_Start(object sender, EventArgs e)
{
  string p1 =
System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecut

  string p2 = System.Web.Hosting.HostingEnvironment.ApplicationPhysi
  string p3 = this.Server.MapPath("");
  Console.WriteLine("p1 = " + p1);
  Console.WriteLine("p2 = " + p2);
  Console.WriteLine("p3 = " + p3);
}
```

result

```
p1 = C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Temporary ASP.NE
Files\root\a897dd66\ec73ff95\assembly\dl3\ff65202d\29daade3_5e84cc01
p2 = C:\inetpub\SBSPortal_staging\
p3 = C:\inetpub\SBSPortal_staging
```

the app is physically running from "C:\inetpub\SBSPortal_staging", so the first solution is definitely not appropriate for web apps.

answered Oct 6 '11 at 19:42

rocketsarefast
3,242 1 17 16

---

protected by Patrick Hofman Aug 4 '14 at 13:56

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?