Is it possible to call a C function from C#.Net

Asked 7 years, 5 months ago Active 2 months ago Viewed 62k times

57

35

I have a C lib and want to call function in this library from C# application. I tried creating a C++/CLI wrapper on the C lib by adding the C lib file as linker input and adding the source files as additional dependencies.

Is there any better way to achieve this as am not sure how to add C output to c# application.

My C Code -

```
__declspec(dllexport) unsigned long ConnectSession(unsigned long handle,
                            unsigned char * publicKey,
                            unsigned char   publicKeyLen);
```

My CPP Wrapper -

```
long MyClass::ConnectSessionWrapper(unsigned long handle,
                                unsigned char * publicKey,
                                unsigned char   publicKeyLen)
    {
        return ConnectSession(handle, publicKey, publicKeyLen);
    }
```

c c#-4.0 interop

edited Jul 15 '12 at 8:51

asked Jul 11 '12 at 3:41

Chinjoo
2,309 5 22 43

2 yeah its possible, and there are many examples if you google it, but no idea what your problem is from the limited information – Keith Nicholas Jul 11 '12 at 3:44

1 If it hurts when you do X, stop doing X. Rename your C++ wrapper method to rule that out as a source of the problem. And post your code :-) – Eric J. Jul 11 '12 at 3:48

I have a c library provided by a vendor which has functions like ConnectToMachine(unsigned char * psig, char apt). I want to cal this function from my c# class. – Chinjoo Jul 11 '12 at 3:48

2 Look into PInvoke – Mark Hall Jul 11 '12 at 3:53

3 Q: Is it possible to call a C function from C#.Net? A: Yes, obviously! Q: Is a C++/CLI wrapper the way to go? Personally, I consider C++/CLI an abomination. But people use it in scenarios just like this. I would prefer to use Interop directly (i.e. PInvoke) - all C#, right down to the native, unmanaged C/C++. There's LOTS of tutorials, and it's *not* difficult. Certainly no more difficult than calling C/C++ from VB6 ;) IMHO.. – paulsm4 Jul 11 '12 at 3:55

4 Answers

84

The example will be, for *Linux*:

1) Create a `C` file, `libtest.c` with this content:

```
#include <stdio.h>

void print(const char *message)
{
  printf("%s\\n", message);
}
```

+150

That's a simple pseudo-wrapper for printf. But represents any `c` function in the library you want to call. If you have a `c++` function don't forget to put extern `c` to avoid mangling the name.

2) create the `c#` file

```
using System;

using System.Runtime.InteropServices;

public class Tester
{
        [DllImport("libtest.so", EntryPoint="print")]

        static extern void print(string message);

        public static void Main(string[] args)
        {

                print("Hello World C# => C++");
        }
}
```

3) Unless you have the library libtest.so in a standard library path like "/usr/lib", you are likely to see a System.DllNotFoundException, to fix this you can move your libtest.so to /usr/lib, or better yet, just add your CWD to the library path: `export LD_LIBRARY_PATH=pwd`

credits from here

EDIT

For *Windows*, it's not much different. Taking an example from here, you only have yo enclose in your `*.cpp` file your method with `extern "C"` Something like

```
extern "C"
{
//Note: must use __declspec(dllexport) to make (export) methods as 'public'
      __declspec(dllexport) void DoSomethingInC(unsigned short int ExampleParam,
unsigned char AnotherExampleParam)
      {
            printf("You called method DoSomethingInC(), You passed in %d and
%c\n\r", ExampleParam, AnotherExampleParam);
      }
}//End 'extern "C"' to prevent name mangling
```

then, compile, and in your C# file do

```
[DllImport("C_DLL_with_Csharp.dll", EntryPoint="DoSomethingInC")]

public static extern void DoSomethingInC(ushort ExampleParam, char
AnotherExampleParam);
```

and then just use it:

```
using System;

    using System.Runtime.InteropServices;

    public class Tester
    {
            [DllImport("C_DLL_with_Csharp.dll", EntryPoint="DoSomethingInC")]

    public static extern void DoSomethingInC(ushort ExampleParam, char
AnotherExampleParam);
```

```
            public static void Main(string[] args)
            {
                    ushort var1 = 2;
                    char var2 = '';
                    DoSomethingInC(var1, var2);
            }
    }
```

edited Jul 21 '12 at 6:14

answered Jul 11 '12 at 3:54

Gonzalo.-
10.3k 4 42 71

6 Good example! This is a very good example of using Interop. It's obviously Linux (not Windows), but the principle is the same. This is *EXACTLY* the approach I'd recommend to the OP. – paulsm4 Jul 11 '12 at 3:59

Could some one edit the answer? I got problems with the include, doesn't show stdio.h – Gonzalo.- Jul 11 '12 at 3:59

guess that should work.. but notsure how to get the output as .so from vs2010.. it only gives .lib and .dll, again .dll when set as dllimport given bad format excpetion – Chinjoo Jul 11 '12 at 7:22

as you can se here en.csharp-online.net/… you could do the same using a .dll – Gonzalo.- Jul 11 '12 at 12:26

1 @Chinjoo consider this codeproject.com/Articles/6912/…. Have you modify your C code, adding _declspec(dllexport) ? If this example works for you, I will add it to the answer – Gonzalo.- Jul 16 '12 at 17:32

28

UPDATE - Feb 22 2019: Since this answer has been getting quite a few upvotes, I decided to update it with a better way of calling the C method. Previously I had suggested using `unsafe` code, but the safe and correct way is to use `MarshalAs` attribute for converting a .NET `string` to a `char*`. Also, in VS2017 there is no Win32 project anymore, you'll probably have to create a Visual C++ dll or empty project and modify that. Thank you!

You can directly call C functions from C# by using P/Invoke.
Here's a short how-to on creating a C# lbrary that wraps around a C dll.

1. Create a new C# Library project (I'll call it "Wrapper")
2. Add a Win32 project to the solution, set application type to: DLL (I'll call it "CLibrary")
   - You can remove all the other cpp/h files since we won't need them
   - Rename the CLibrary.cpp file to CLibrary.c
   - Add a CLibrary.h header file
3. Now we need to configure the CLibrary project, right-click it and go to properties, and select Configuration: "All Configurations"
   - In Configuration Properties > C/C++ > Precompiled headers, set Precompiled Headers to: "Not using Precompiled Headers"
   - In the same C/C++ branch, go to Advanced, change Compile As to: "Compile as C code (/TC)"
   - Now in the Linker branch, go to General, and change Output File to: "$(SolutionDir)Wrapper\$(ProjectName).dll", this will copy the built C DLL to the C# project root.

CLibrary.h

```
__declspec(dllexport) unsigned long ConnectSession(unsigned long   handle,
                                                   unsigned char * publicKey,
                                                   unsigned char   publicKeyLen);
```

CLibrary.c

```
#include "CLibrary.h"
```

```
unsigned long ConnectSession(unsigned long   handle,
                             unsigned char * publicKey,
                             unsigned char   publicKeyLen)
{
    return 42;
}
```

- Right-click CLibrary project, build it, so we get the DLL in the C# project directory
- Right-click C# Wrapper project, add existing item, add CLibrary.dll
- Click on CLibrary.dll, go to the properties pane, set "Copy to output Directory" to "Copy Always"

It's a good idea to make the Wrapper project dependent on CLibrary so CLibrary gets built first, you can do that by right-clicking the Wrapper project, going to "Project Dependencies" and checking "CLibrary". Now for the actual wrapper code:

ConnectSessionWrapper.cs

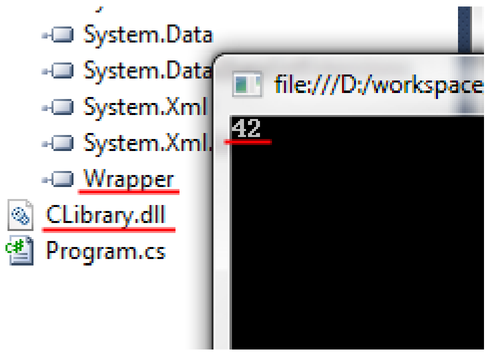
```
using System.Runtime.InteropServices;

namespace Wrapper
{
    public class ConnectSessionWrapper
    {
        [DllImport("CLibrary.dll", CallingConvention = CallingConvention.Cdecl)]
        static extern uint ConnectSession(uint handle,
            [MarshalAs(UnmanagedType.LPStr)] string publicKey,
            char publicKeyLen);

        public uint GetConnectSession(uint handle,
                                      string publicKey,
                                      char publicKeyLen)
        {
            return ConnectSession(handle, publicKey, publicKeyLen);
        }
    }
}
```

Now just call `GetConnectSession`, and it should return `42`.

Result:



```
42
```

edited Feb 22 at 10:33

answered Jul 21 '12 at 15:57
Shahin Dohan
2,313 1 18 29

This worked for me, but I also had to change my .NET project's `Properties->Build->Platform Target to x64`. – Joseph Feb 21 '18 at 21:47

So cool, what happens if you don't marshal the char pointer? – Jonney Shih Nov 30 at 21:37

Okay well, Open VS 2010, Goto *File -> New -> Project -> Visual C++ -> Win32 -> Win32 Project* and give it a name (HelloWorldDll in my case), Then in the window that follows under *Application Type* choose *'DLL'*

3

and under *Additonal Options* choose *'Empty Project'*.

Now goto your *Solution Explorer* tab usually right hand side of VS window, right click *Source Files -> Add Item -> C++ file (.cpp)* and give it a name (HelloWorld in my case)

Then in the new class paste this code:

```
#include <stdio.h>

extern "C"
{
  __declspec(dllexport) void DisplayHelloFromDLL()
  {
    printf ("Hello from DLL !\n");
  }
}
```

Now *Build* the project, after navigate to your projects *DEBUG* folder and there you should find: **HelloWorldDll.dll**.

Now, lets create our C# app which will access the dll, Goto *File -> New -> Project -> Visual C# -> Console Application* and give it a name (CallDllCSharp), now copy and paste this code to your main:

```
using System;
using System.Runtime.InteropServices;
...
        static void Main(string[] args)
        {
            Console.WriteLine("This is C# program");
            DisplayHelloFromDLL();
            Console.ReadKey();
        }
```

and build the program, now that we have both our apps built lets use them, get your *.dll and your *.exe (bin/debug/*.exe) in the same directory, and execute the application output should be

> This is C# program
>
> Hello from DLL !

Hope that clears some of your issues up.

References:

- How to create a DLL library in C and then use it with C#

edited Jul 20 '12 at 20:58

answered Jul 20 '12 at 20:46

David Kroukamp
31.2k 11 65 117

I could not compile the C# program because my project can not find the DLL file. So I used the full path in the DllImport statement : `[DllImport("C:\\Users\\...full path...\\HelloWorldDll\\Debug\\HelloWorldDll.dll")]` – Quazi Irfan Apr 23 '17 at 8:16

Keep getting 'An attempt was made to load a program with an incorrect format. (Exception from HRESULT: 0x8007000B)'. Anyone else got this issue? – Max Aug 19 at 12:51

NOTE : BELOW CODE IS FOR MULTIPLE METHODS FROM DLL.

0

```
[DllImport("MyLibc.so")] public static extern bool MyLib_GetName();
[DllImport("MyLibc.so")] public static extern bool MyLib_SetName(string name);
[DllImport("MyLibc.so")] public static extern bool MyLib_DisplayName(string
name);

public static void Main(string[] args)
{
string name = MyLib_GetName();
MyLib_SetName(name);
MyLib_DisplayName(name);
}
```

answered Oct 2 at 21:51

HaSeeB MiR
364 7 11