using a class defined in a c++ dll in c# code

Asked 16 years, 1 month ago Modified 5 years, 4 months ago

Viewed 124k times

85

I have a dll that was written in c++, I need to use this dll in my c# code. After searching I found that using P/Invoke would give me access to the function I need, but these functions are defined with in a class and use non-static private member variables. So I need to be able to create an instance of this class to properly use the functions. How can I gain access to this class so that I can create an instance? I have been unable to find a way to do this.

I guess I should note that the c++ dll is not my code.

c# c++ dll pinvoke

Share

Improve this question

Follow

asked Nov 24, 2008 at 18:49

Dan Vogel

3,938 ● 7 ● 44 ● 57

6 Answers Sorted by: Highest score (default)

111

There is no way to directly use a C++ class in C# code. You can use PInvoke in an indirect fashion to access your type.

The basic pattern is that for every member function in class Foo, create an associated non-member function which calls into the member function.

```
class Foo {
public:
  int Bar();
};
extern "C" Foo* Foo_Create() { return new Foo(); }
extern "C" int Foo_Bar(Foo* pFoo) { return pFoo->Bar()
extern "C" void Foo_Delete(Foo* pFoo) { delete pFoo; }
```

Now it's a matter of PInvoking these methods into your C# code

```
[DllImport("Foo.dll")]
public static extern IntPtr Foo_Create();

[DllImport("Foo.dll")]
public static extern int Foo_Bar(IntPtr value);

[DllImport("Foo.dll")]
public static extern void Foo_Delete(IntPtr value);
```

The downside is you'll have an awkward IntPtr to pass around but it's a somewhat simple matter to create a C# wrapper class around this pointer to create a more usable model.

Even if you don't own this code, you can create another DLL which wraps the original DLL and provides a small PInvoke layer.

Share Improve this answer

Follow

edited Jun 29, 2018 at 16:02

milleniumbug

15.8k ● 3 ● 51 ● 72

answered Nov 24, 2008 at 18:54

JaredPar

753k ● 151 ● 1.3k ● 1.5k

2 The statement of "There is no way to directly use a C++ class in C# code" is not correct. A static C++ function works the same way as C-style function and the instance functions can be declared as ThisCall calling convention by adding ThisObject pointing to the instance itself as the first argument. For details, you may want to read my Blogs. You may also want to try our tool.(I am the author) – xInterop Apr 11, 2014 at 1:09

3 Though a fine answer at the time, it's arguably better to use c++/CLI so as to avoid the manual creation of proxy functions - something that could become quite tedious rather quickly – user585968 Feb 15, 2015 at 17:40

2 Should the C++ proxy functions be surrounded by `extern "C" { .... }` or is it fine to do it purely in C++? – divB Apr 1, 2015 at 6:26

2 @jaredpar where should I copy c++ dll? beside c# exe? I get System.EntryPointNotFoundException: 'Unable to find an entry point named '...' in DLL – Reza Akraminejad Jul 24, 2019 at 10:01

Marshal C++ Class and use the PInvoke

C++ Code ,ClassName.h

```
class __declspec(dllexport) CClassName
{
 public:
    CClassName();
    ~CClassName();
    void function();

};
```

C++ Code, ClassName.cpp

```
CClassName::CClassName()
{
}

CClassName::~CClassName()
{
}

void CClassName::function()
{
    std::cout << "Bla bla bla" << std::endl;

}
```

C++ Code, ClassNameCaller.h file for the caller function

```
#include "ClassName.h"

#ifdef __cplusplus
extern "C" {
```

```
#endif

extern __declspec(dllexport) CClassName* CreateClassNa

extern __declspec(dllexport) void DisposeClassName(CCl

extern __declspec(dllexport) void function(CClassName*


#ifdef __cplusplus
}
#endif
```

C++ Code, ClassNameCaller.cpp file for the caller function

```
#include "ClassNameCaller.h"


CClassName* CreateClassName()
{
    return new CClassName();
}

void DisposeClassName(CClassName* a_pObject)
{
    if(a_pObject!= NULL)
    {
        delete a_pObject;
        a_pObject= NULL;
    }
}

void function(CClassName* a_pObject)
{
    if(a_pObject!= NULL)
    {
        a_pObject->function();
    }
}
```

C# code

```
[DllImport("ClassNameDll.dll")]
static public extern IntPtr CreateClassName();

[DllImport("ClassNameDll.dll")]
static public extern void DisposeClassName(IntPtr pCla

[DllImport("ClassNameDll.dll")]
static public extern void CallFunction(IntPtr pClassNa

//use the functions
IntPtr pClassName = CreateClassName();

CallFunction(pClassName);

DisposeClassName(pClassName);

pClassName = IntPtr.Zero;
```

Share Improve this answer

Follow

edited Mar 8, 2018 at 5:19

answered Apr 12, 2016 at 12:59

Amir Touitou

3,421 ● 1 ● 37 ● 32

2 Shouldn't there be a "dllexport" instead "dllimport" in the first file, ClassName.h? – P.W. Mar 7, 2018 at 17:11

1 Yes , It is used to export this class to .Net project so we can use the CClassName class – Amir Touitou Mar 8, 2018 at 5:20

2 This example really helped a lot, Thank you so much:-) – Ashish Rana Mar 28, 2018 at 10:15

1 A side note, if you are getting a "BadImageFormatException", you might have to change the platform. I compiled my C++ code as x64 as usual, but the default project I created for the C# side was set to "Any CPU". This does not work, but it does once switched to x64. Make sure that matches.
– clocktown Aug 29, 2019 at 13:47

1 Great example. This got me started on how to wrap npcap.dll, which is a C++'ish (mostly C) dll for use in a .Net Windows Forms application, by wrapping the npcap.dll in an unmanaged C++ dll and then .Net to accessed the wrapped DLL – MtnManChris Jan 23 at 20:47

5

Here is a sample how to call C++ class method from VB - for C# you only have to rewrite the sample program in Step 4.

Share Improve this answer

Follow

answered Nov 24, 2008 at 19:05

Dmitry Khalatov
4,349 ● 4 ● 35 ● 39

4

The way I've done this is by creating a thin Managed C++ wrapper around my unmanaged C++ DLL. The managed wrapper contains "proxy" classes that wrap around the unmanaged code exposing the interface that's needed by the .NET application. This is a bit of double work but it allows quite seamless operation in normal environments. Things do get trickier with dependencies in some circumstances (such as ASP.NET) but you will probably not run into that.

Share Improve this answer

Follow

answered Nov 25, 2008 at 9:30

Joris Timmermans

11k ● 2 ● 52 ● 78

3

You may need to write an intermediary DLL (in C++, perhaps) that handles this for you and exposes the interface you need. Your DLL would be in charge of loading the 3rd party DLL, creating an instance of this C++ object, and exposing its member functions as needed via whatever API you design. You would then use P/Invoke to get at your API and cleanly manipulate the object.

Note: For the API of your DLL, try keeping the data types limited to primitives (long, int, char*, etc.) to prevent module boundary issues.

Share Improve this answer

Follow

answered Nov 24, 2008 at 18:56

Brian

3,555 ● 5 ● 33 ● 41

2

I agree with JaredPar. Creating instances of unmanaged classes in managed code should not be possible.

Another thing is - if you could recompile the DLL in managed C++ or make a COM component out of it, it would be much easier/

Share Improve this answer

answered Nov 24, 2008 at 18:58

Follow

Paul Kapustin

3,297 5 37 45