Overloaded functions in C++ DLL def file

Asked 16 years, 4 months ago Modified 11 years, 9 months ago Viewed 15k times

11

I'm writing a C/C++ DLL and want to export certain functions which I've done before using a .def file like this

```
LIBRARY "MyLib"
EXPORTS
  Foo
  Bar
```

with the code defined as this, for example:

```
int Foo(int a);
void Bar(int foo);
```

However, what if I want to declare an overloaded method of Foo() like:

```
int Foo(int a, int b);
```

As the def file only has the function name and not the full prototype I can't see how it would handle the overloaded functions. Do you just use the one entry and then specify which overloaded version you want when passing in the properly prototyped function pointer to LoadLibrary() ?

Edit: To be clear, this is on Windows using Visual Studio 2005

Edit: Marked the non-def (__declspec) method as the answer...I know this doesn't actually solve the problem using def files as I wanted, but it seems that there is likely no (official) solution using def files. Will leave the question open, however, in case someone knows something we don't have overloaded functions and def files.

c++ c dll

Share
Improve this question
Follow

edited Aug 25, 2008 at 14:56

asked Aug 25, 2008 at 14:22

Adam Haile
31.3k ● 60 ● 195 ● 290

6 Answers

Sorted by: Highest score (default)

11

Function overloading is a C++ feature that relies on name mangling (the cryptic function names in the linker error messages).

By writing the mangled names into the def file, I can get my test project to link and run:

```
LIBRARY "TestDLL"
EXPORTS
    ?Foo@@YAXH@Z
    ?Foo@@YAXHH@Z
```

seems to work for

```
void Foo( int x );
void Foo( int x, int y );
```

So copy the C++ function names from the error message and write them into your def file. However, the real question is: Why do you want to use a def file and not go with __declspec(dllexport) ?

The mangled names are non-portable, I tested with VC++ 2008.

Share Improve this answer Follow

answered Aug 25, 2008 at 14:42

Timbo

28.1k ● 11 ● 51 ● 75

Interesting approach. By non-portable you mean across different versions of Visual Studio? Which would suggest they change their name mangling schemes between versions perhaps? – jxramos Oct 20, 2015 at 22:03

@jxramos I am not sure whether they actually can change the name mangling scheme. But I doubt this will work in the same way when switching to another compiler, unless that compiler tries to emulate VC's behavior. – Timbo Oct 21, 2015 at 7:31

This is surely only a VC++ thing since I don't think other compilers use def files. Also if I remember what someone once told me is that Microsoft has this dll interface idea where user selected elements are exposed publicly, via the def file or __declspec's whereas in Unix with their *.so files everything with a public API is exposed. They don't have a distinction between the logical public API and the library's public API. – jxramos Oct 21, 2015 at 19:42

10

In the code itself, mark the functions you want to export using __declspec(dllexport). For example:

```
#define DllExport __declspec(dllexport)

int DllExport  Foo( int a ) {
  // implementation
}
int DllExport Foo( int a, int b ) {
  // implementation
}
```

If you do this, you do not need to list the functions in the .def file.

Alternatively, you may be able to use a default parameter value, like:

```
int Foo( int a, int b = -1 )
```

This assumes that there exists a value for b that you can use to indicate that it is unused. If -1 is a legal value for b, or if there isn't or shouldn't be a default, this won't work.

Edit (Adam Haile): Corrected to use __declspec as __dllspec was not correct so I could mark this as the official answer...it was close enough.

Edit (Graeme): Oops - thanks for correcting my typo!

Share

Improve this answer

Follow

edited Aug 25, 2008 at 16:53

answered Aug 25, 2008 at 14:32

Graeme Perrow

57.2k ● 24 ● 86 ● 125

1 what if we are using GetProcAddress() with a dynamic DLL? – null Mar 21, 2013 at 6:07

3 Then you need to use the mangled names, or rename one of the functions and make them both `extern "C"`, assuming neither of them takes or returns C++ objects. – Graeme Perrow Mar 21, 2013 at 19:17

9

I had a similar issue so I wanted to post on this as well.

1. Usually using

   ```
   extern "C" __declspec(dllexport) void Foo();
   ```

   to export a function name is fine. It will *usually* export the name unmangled without the need for a .def file. There are, however, some exceptions like

__stdcall functions and overloaded function names.

2. If you declare a function to use the __stdcall convention (as is done for many API functions) then

```
extern "C" __declspec(dllexport) void __stdcall Foo();
```

will export a mangled name like _Foo@4. In this case you may need to explicitly map the exported name to an internal mangled name.

A. How to export an unmangled name. In a .def file add

```
----
EXPORTS
    ; Explicit exports can go here

    Foo
-----
```

This will try to find a "best match" for an internal function Foo and export it. In the case above where there is only one foo this will create the mapping

Foo = _Foo@4

as can be see via dumpbin /EXPORTS

If you have overloaded a function name then you may need to explicitly say which function you want in the .def file by specifying a mangled name using the entryname[=internalname] syntax. e.g.

```
----
EXPORTS
    ; Explicit exports can go here

    Foo=_Foo@4
-----
```

B. An alternative to .def files is that you can export names "in place" using a #pragma.

```
#pragma comment(linker, "/export:Foo=_Foo@4")
```

C. A third alternative is to declare just one version of Foo as extern "C" to be exported unmangled. See here for details.

answered Dec 4, 2008 at 21:03

There is no official way of doing what you want, because the dll interface is a C api.

The compiler itself uses mangled names as a workaround, so you should use name mangling when you don't want to change too much in your code.

Share Improve this answer Follow

answered Aug 27, 2008 at 11:11

Christopher

8,982 ● 3 ● 34 ● 38

3

Systax for EXPORTS definition is:

```
entryname[=internalname] [@ordinal [NONAME]] [PRIVATE] [DATA]
```

entryname is the function or variable name that you want to export. This is required. If the name you export is different from the name in the DLL, specify the export's name in the DLL with internalname.

For example, if your DLL exports a function, func1() and you want it to be used as func2(), you would specify:

```
EXPORTS
func2=func1
```

Just see the mangled names (using Dependency walker) and specify your own functions name.

Source: http://msdn.microsoft.com/en-us/library/hyx1zcd3(v=vs.71).aspx

Edit: This works for dynamic DLLs, where we need to use GetProcAddress() to explicitly fetch a functions in Dll.

Share Improve this answer Follow

answered Mar 21, 2013 at 6:15

null

807 ● 1 ● 8 ● 13

3

There isn't a language or version agnostic way of exporting an overloaded function since the mangling convention can change with each release of the compiler.

This is one reason why most WinXX functions have funny names like *Ex or *2.

Share

2

Improve this answer

Follow

edited Jul 3, 2012 at 15:02

user142162

answered Aug 25, 2008 at 15:54

Mat Noguchi

1,080 ● 6 ● 7

interesting background with the WinXX comment! – jxramos Oct 20, 2015 at 22:06