

Using C++ classes in .so libraries

Asked 16 years, 3 months ago Modified 8 years, 6 months ago

Viewed 26k times



12



I'm trying to write a small class library for a C++ course.

I was wondering if it was possible to define a set of classes in my shared object and then using them directly in my main program that demos the library. Are there any tricks involved? I remember reading this long ago (before I started really programming) that C++ classes only worked with MFC .dlls and not plain ones, but that's just the windows side.

c++

linux

class-library

Share

Improve this question

Follow

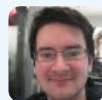
edited Jun 2, 2016 at 9:06



frogatto

29.2k ● 13 ● 89 ● 134

asked Sep 12, 2008 at 0:58



Flame

2,207 ● 2 ● 22 ● 41

3 Answers

Sorted by:

Highest score (default)





14



C++ classes work fine in .so shared libraries (they also work in non-MFC DLLs on Windows, but that's not really your question). It's actually easier than Windows, because you don't have to explicitly export any symbols from the libraries.



This document will answer most of your questions:



<http://people.redhat.com/drepper/dsohowto.pdf>



The main things to remember are to use the `-fPIC` option when compiling, and the `-shared` option when linking. You can find plenty of examples on the net.

Share Improve this answer

edited Jun 2, 2016 at 9:07

Follow

community wiki

3 revs, 2 users 90%

Kristopher Johnson



My solution/testing

10

Here's my solution and it does what i expected.



Code



cat.hh :



```

#include <string>

class Cat
{
    std::string _name;
public:
    Cat(const std::string & name);
    void speak();
};

```

cat.cpp :

```

#include <iostream>
#include <string>

#include "cat.hh"

using namespace std;

Cat::Cat(const string & name):_name(name){}
void Cat::speak()
{
    cout << "Meow! I'm " << _name << endl;
}

```

main.cpp :

```

#include <iostream>
#include <string>
#include "cat.hh"

using std::cout;using std::endl;using std::string;
int main()
{
    string name = "Felix";
    cout<< "Meet my cat, " << name << "!" <<endl;
    Cat kitty(name);
    kitty.speak();
}

```

```
    return 0;  
}
```

Compilation

You compile the shared lib first:

```
$ g++ -Wall -g -fPIC -c cat.cpp  
$ g++ -shared -Wl,-soname,libcat.so.1 -o libcat.so.1 c
```

Then compile the main executable or C++ program using the classes in the libraries:

```
$ g++ -Wall -g -c main.cpp  
$ g++ -Wall -Wl,-rpath,. -o main main.o libcat.so.1 #  
prevents the need to use LD_LIBRARY_PATH when testing  
$ ./main  
Meet my cat, Felix!  
Meow! I'm Felix  
$
```

Share Improve this answer

edited Sep 16, 2008 at 2:41

Follow


answered Sep 12, 2008 at 2:12



Flame

2,207 ● 2 ● 22 ● 41

-
- 11** This code shows falls into one of the 'gotchas' of C++ classes across shared libraries. By using a STL class in the 'shared' interface means that the library and the application are each compiling their own implementation of the STL

class 'string'. If both these projects are compiled at the same time on the same machine using the same compiler, then it all looks good. Compile the library, wait 18 months, and then compile the application and it 'may' work. but there is no guarantee. Distribute your shared libraries in binary to other developers and you will have 'issues' – [Michael Shaw](#) Sep 4, 2013 at 8:04 

- 1 As others have suggested, this approach is flawed, due to name mangling differences across compilers. I found a good resource that appears to show the right way (Section 3.3) : tldp.org/HOWTO/C++-dlopen/thesolution.html – [CraigDavid](#) Jul 22, 2020 at 0:35
-



5



As I understand it, this is fine so long as you are linking .so files which were all compiled using the same compiler. Different compilers mangle the symbols in different ways and will fail to link.

That is one of the advantages in using COM on Windows, it defines a standard for putting OOP objects in DLLs. I can compile a DLL using GNU g++ and link it to an EXE compiled with MSVC - or even VB!

Share Improve this answer

answered Sep 12, 2008 at 1:24

Follow



[Adam Pierce](#)

34.3k ● 23 ● 71 ● 89

-
- 2 There is a caveat to this: you can use shared libraries containing C++ code compiled by different compilers if: (a) the symbol mangling schemes match or (b) you have some `extern C` functions to use as API glue. – [Ben Collins](#) Sep 16, 2008 at 2:38
-