How do I control which symbols a Windows DLL imports from the application?

Asked 16 years, 1 month ago Modified 16 years, 1 month ago

Viewed 634 times

1

I'm trying to build a shared library (DLL) on Windows, using MSVC 6 (retro!) and I have a peculiar link issue I need to resolve. My shared library must access some global state, controlled by the loading application.

Broadly, what I have is this:

application.c:

```
static int g_private_value;

int use_private_value() {
    /* do something with g_private_value */
}

int main (...) {
    return shared_library_method ();
}
```

shared_library.c:

```
__declspec(dllexport) int __stdcall
shared_library_method() {
    use_private_value();
}
```

(**Updated** - I forgot the `__declspec(dllexport) int __stdcall` portion, but it's there in the real code)

How do I set up shared_library.dll so that it exports `shared_library_method` and imports `use_private_value`?

Please remember that A) I'm a unix programmer, generally, and B) that I'm doing this without Visual Studio; our automated build infrastructure drives MSVC with makefiles. If I'm omitting something that will make it easier to answer the question, please comment and I'll update it ASAP.

windows linker shared-libraries

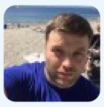
Share

Improve this question

Follow

edited Oct 29, 2008 at 17:16

asked Oct 29, 2008 at 16:27



Chris R

17.9k ● 25 ● 110 ● 174

3 Answers

Sorted by: Highest score (default)

2

This is actually going to be pretty difficult to get working. On Unix/Linux you can have shared objects and applications import symbols from each other, but on Windows you can't have a DLL import symbols from the application that loads it: the Windows PE executable format just doesn't support that idiom.

I know that the Cygwin project have some sort of work-around to address this problem, but I don't believe that it's trivial. Unless you want to do lots of PE-related hacking you probably don't want to go there.

An easier solution might be to just have some sort of initializer method exported from the DLL:

```
typedef int (*func_ptr)();
void init_library(func_ptr func);
```

The application must call this at start-up, passing in the address of the function you want to share. Not exactly elegant, but it should work okay.

Share Improve this answer Follow

answered Oct 29, 2008 at 17:22

user31394

1

I'll start with half of the answer.

In shared_library.c:

```
__declspec(dllexport) int __stdcall
shared_library_method(void)
{


}
```

The MSDN article about exporting function from DLL:s.

Share Improve this answer Follow

answered Oct 29, 2008 at 16:59

Peter Olsson
1,322 ● 3 ● 13 ● 20

I should have included that; my declarations in the shared library DO have that. Updated in the post. – Chris R Oct 29, 2008 at 17:14

1

For the second half you need to export the functions from your application.c. You can do this in the linker with:

```
/export:use_private_value@0
```

This should get you a lib-file that you build with your DLL.

The option to link the lib-file is to use GetProcAddress().

As DavidK noted if you only have a few functions it is probably easier to pass the function pointers in an init function. It is however possible to do what you are asking for.

Share Improve this answer

Follow

answered Oct 29, 2008 at 17:30

Peter Olsson

1,322 ● 3 ● 13 ● 20