
Subject: Re: C++ and CALL_EXTERNAL

Posted by [mark.t.douglas](#) on Fri, 30 May 2008 10:26:42 GMT

[View Forum Message](#) <> [Reply to Message](#)

On 29 May, 12:20, James Kuyper <jameskuy...@verizon.net> wrote:

> mark.t.doug...@gmail.com wrote:

>> After an entire evening wasted trying to get IDL to interface with a
>> DLL I made, I thought I'd jot down the things that I wish I had known
>> at the beginning, in the hope that it will be useful to someone,
>> somewhere, sometime. I was using IDL 6.1 on Windows and Microsoft's
>> Visual C++ Express 2008 compiler; the same procedure will work on
>> other OSES, mutatis mutandis.

>

>> OK, here we go. Suppose we have two functions, written in C++, that we
>> wish to use from within IDL. We naively start with the following
>> header:

>

```
>> #ifndef NORMALS_H  
>> #define NORMALS_H
```

>

```
>> namespace Normals
```

```
>> {
```

```
>>   __declspec(dllexport) double InverseCumulative(double x);
```

```
>>   __declspec(dllexport) double Cumulative(double x);
```

```
>> }
```

>

```
>> #endif
```

>

>> After building the DLL and moving it to IDL's working directory, we
>> type the following into IDL:

>

```
>> x = call_external("MyLib.dll","Cumulative",double(0.5),/all_value,/d_value,value=[0])
```

>

>> It can't find the function! Why? Because the polymorphism and
>> overloading features of C++ are usually implemented by mangling your
>> nice function names into something that looks like a core dump.

>> Examine your DLL with a program like PEDUMP to figure out what

>> Normals::Cumulative() is now known as; I get ?

>> Cumulative@Normals@@YANN@Z . That line noise encodes precise
>> information about the argument types accepted by the function, believe
>> it or not. Armed with this information, we type the following into

>> IDL:

>

```
>> x = call_external("MyLib.dll","?
```

```
>> Cumulative@Normals@@YANN@Z",double(0.5),/all_value,/d_value,value=[0])
```

>

> Wouldn't it be simpler to disable the name mangling by declaring the

- > functions as 'extern "C" ? You can still use any feature of C++ that
- > you want, inside the definition of the function. Of course, you can't
- > use any C++ features in the function interface of an 'extern "C"
- > function that are not also supported by C, but CALL_EXTERNAL probably
- > couldn't handle those features anyway.

That would have worked fine and made life simpler for the two functions I outlined here, certainly. However there are other things in the DLL which are "proper" C++ so I elected not to use extern "C" for the sake of consistency, as the DLL was designed as a C++ library in the first instance. I probably should have mentioned this in the original post!
