Subject: Re: FORTRAN DLL and CALL_EXTERNAL
Posted by steinhh on Thu, 21 May 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Shane wrote,

> I'm trying to get call_external working under NT4 (i.e. using DLL's)
> using FORTRAN. Has anybody done this before with success? Is there any
> example code (like there is for C in the Advanced development
> guide)...please this would help alot?. The tools I'm using are IDL
> v.5, Digital visual fortran and NT4.

If you can get a DLL written in C to work, and if you can get
a normal C-program to call a FORTRAN subroutine, then you
simply make a C wrapper that calls the FORTRAN subroutine,
and make the DLL by linking the C wrapper + FORTRAN compiler
output (not linked, just compiled) together into a DLL.
Your IDL routine should then call the C wrapper.

If you can't get a C-source DLL to work, you should consult
the Adv. dev. guide + the man pages for your C compiler.

If you can't get a C program to call a FORTRAN routine,
consult the C and FORTRAN man pages.

Hint: Some times there is a lot of problems identifying
the names of subroutines from one compiler to another,
due to various name-altering conventions. Some compilers
put an underscore (or two!) at the beginning or end
of compiled functions, some don't.

Look in the ...idl/external/sharelib directory for examples
of both C and FORTRAN examples.

Regards,

Stein Vidar

Subject: Re: FORTRAN DLL and CALL_EXTERNAL
Posted by Peter Mason on Fri, 22 May 1998 07:00:00 GMT
View Forum Message <> Reply to Message

On 21 May 1998, Stein Vidar Hagfors Haugan wrote:
> Shane wrote,
>> I'm trying to get call_external working under NT4 (i.e. using DLL's)
>> using FORTRAN. Has anybody done this before with success? Is there any
>> example code (like there is for C in the Advanced development

>> guide)...please this would help alot?. The tools I'm using are IDL
>> v.5, Digital visual fortran and NT4.
>
> If you can get a DLL written in C to work, and if you can get
> a normal C-program to call a FORTRAN subroutine, then you
> simply make a C wrapper that calls the FORTRAN subroutine,
> and make the DLL by linking the C wrapper + FORTRAN compiler
> output (not linked, just compiled) together into a DLL.
> Your IDL routine should then call the C wrapper.
<...>

I don't use Fortran myself these days, but out of curiosity I had a crack at trying to get an IDL DLL going with MS VC++5 along with MS Powerstation Fortran 4 (predecessor to Digital Visual Fortran) the other day. These two are essentially a generation apart, and the linking (which RTLS?) got pretty ugly. The new DEC Fortran is supposed to include the current MS Visual Studio environment, or something, but I haven't tried linking Visual C++5 routines with its Fortran routines myself - it might not be that straightforward.

My point being ...?
You can probably write your DLL entirely in Fortran. Back in October 1996 Eric Dors sent a post to this group describing how to do this (for Unix, admittedly), and behold, the gist of it is in the new "external development guide" that comes with IDL 5.1.

Now this deals with accessing the parameters that IDL passes in the C-style (argc, *argv[]) way, from within a Fortran program. DEC Fortran extensions are used to do this, so I'm sure that this part will work for you.

In addition to this, you will have to make sure that your external functions are exported in the right way. This involves:
1 Specify the "stdcall" calling convention. The critical bits here are that the callee (your routine) pops its own arguments from (cleans) the stack, arguments are passed right-to-left, and arguments are passed by value unless a pointer is passed. (e.g., "argv" goes by value, "argc" by ref.)
2 Clean up the exported names, or at least know how they've been changed. We have to use stdcall because IDL sends the parameters this way. Unfortunately, stdcall (in C, at least) makes a right stuff-up of the exported names. It sticks a "_" on the front and an "@x" on the end of each name (where the x in "@x" gives the number of bytes in the argument list - typically 8 for routines exported for IDL). Your compiler might even add some beauties of its own - you'll have to check out what it does. *** Also check what it does w.r.t. case. *** In C we use .DEF files to get rid of this junk. You might not have this mechanism, but you might have some compiler options to get rid of at least some of the junk. If you can't get rid of the junk then all is not lost -

you will probably still be able to access your routines by calling them with their "decorated" names. e.g., If you had a routine called "test", you might call it "_test@8" in IDL. (I haven't tried this but I can't see why it shouldn't work.)

I hope this helps

Peter Mason
