Subject: Q: interfacing IDL to DLL
Posted by Herbert H. Tsang on Wed, 28 Jul 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Anyone has epxerience in interfacing IDL to C functions' DLL? Is
there anything special I need to do in order to make a DLL that's works?

-- Herbert (tsang@vcn.bc.ca)

Subject: Re: Q: interfacing IDL to DLL
Posted by Peter Mason on Fri, 30 Jul 1999 07:00:00 GMT
View Forum Message <> Reply to Message

davidf@dfanning.com (David "Definitely Not Decomposed" Fanning) wrote:
> Herbert H. Tsang (tsang@vcn.bc.ca) writes:
>> Anyone has epxerience in interfacing IDL to C functions' DLL? Is
>> there anything special I need to do in order to make a DLL that's
works?
>
> No, just write it correctly. :-)

I would wager that David means that it's much easier than you might
think, and that you should just go for it.
It can be a bit daunting the first time, what with the docs implying
that DLLS are for the heavies and with the IDL call-external example
using that massive include file, but for the average IDL user an
external library just has number-crunching routines and such a DLL is
very straightforward. An outline...

In Microsoft Visual C++, go "New...", "Win32 dynamic-link library" to
set up your project. All you really have to do here is give your
project a name. Then create a C file: go "Project"->"Add to
project"->"New", pick "C++ source file" and type in a name. Use a .C
extension. (Normally I'd give this file the same name as the project.)

OK, now if you're just going to have crunch routines in the DLL then the
only windowsy thing (well almost, more later) about the whole deal is
that you must have a DLLMain function. Just a token one, like this:

```
BOOL WINAPI DllMain(HINSTANCE hinst, unsigned long reason, void *resvd)
{
  //just access the vars to suppress compiler warnings
  hinst=hinst;  reason=reason;  resvd=resvd;
  return 1;
}
//Note you have to #include <windows.h> to get "WINAPI".
```

After this you'd have your crunch routines. Having read the stuff in the main IDL docs about external routines, and the IDL external development guide (you *have* read these, haven't you?), you'd know that these external routines must use a special "int argc, void *argv[]" parameter-passing mechanism. Each routine has just these 2 parameters. argc shows how many "actual" parameters have been passed, and the argv array has each one's address or value. You'd also know that by default IDL passes things by reference (so there are just addresses in argv), but you can override this if you really want to. I won't go into this any more - check out the IDL docs for more info.

Now comes the trickiest part, I think, and it really isn't that bad. There isn't just one way that routines in a DLL can be exported and presented to the outside world. So how does IDL want things? There are two issues here: function naming, and parameter-passing mechanism. Some export methods attach bits of flack to your function names. e.g., "stdcall" will take a name like IDLMean and turn it into _IDLMean@8 (given that it has 8 bytes worth of parameters). (In the trade this is called "decoration".) Some convert to all upper or lowecase. If you know what's going to happen to your function names you can use them in their mucked-up morm - er.. form - on the IDL side. But most of us prefer the original names, and the way to get them "back" is to use a .DEF file, which at the same time exports the functions. Getting the names wrong isn't so bad - IDL will report an error along the lines that it "couldn't find entry point IDLMean" or such, and you can fish around until you get it to work.
The parameter-passing mechanism is the all-critical issue. This is about whether parameters are pushed onto the stack from left to right or from right to left, and whether the caller or the callee must clean up the stack. (This is all "under the hood" - it doesn't involve programming changes on your part.) If you don't get this right then sooner or later your IDL+external program will crash.

OK there's some background. Now on the IDL side, IDL will try to use whatever function name you give it (decorated or not), and it offers a choice between the "stdcall" and the "cdecl" parameter-passing mechanisms, the default being "stdcall". I'll show you how to use "stdcall" along with a .DEF file for exporting and name-decoration removal. I'll use a simple example that calculates and returns the mean of a float array. I'm assuming that you're naming your DLL "mydll".

IDL side:

```
n=1000L
test_array=randomn(undef_seed,n)
test_array_mean=call_external("\mypathto\mydll","idlmean",test_array,n)
```

C side:

(Add this routine to your .C file that so far just has DLLMain.)

```
float WINAPI idlmean(int argc, float *argv[])
{
  float  *a;
  double av;
  int    n,i;
  if(argc!=2) return 0.0F;  //you might prefer a fancier error handler
  a=argv[0];                //our array
  n=*((int *)(a[1]));       //our element count
  for(av=0.0,i=0; i<n; i++) av+=a[i];
  return (float)(av/n);
}
// In case you're wondering, "WINAPI" translates to "__stdcall"
```

.DEF file:
GO "Project"->"Add to project"->"New", pick "C++ source file" and type in a filename with a .DEF extension. (Probably use the same base-name as your .C file.) Edit it and put something like this in it. The function names you specify in the EXPORTS section are the names that you want to see on the IDL side:

```
  LIBRARY  mydll
  DESCRIPTION 'This and that'
  EXPORTS  idlmean @1
```

Compile and link, and Robert is your relative.

I hope this helps to get you started,
cheers
Peter Mason