Subject: Re: Dynamically creating C++ objects in DLM
Posted by Randall Skelton on Thu, 02 Jan 2003 18:55:20 GMT
View Forum Message <> Reply to Message

Hi Rick,

As long as you aren't trying to directly interface IDL and C++, I think
you will be fine.

> How do I pass pointers back and forth between my DLM and IDL? Is it as
> simple as passing a long?
>
> return IDL_GetTmpLong(test); //Is it this easy?
>
> If I do pass IDL a long representing my pointer, how do I use it when I pass
> it back to my DLM? The compiler sees it as a long, not a pointer to an object.
>
> pointer = IDL_LongScalar(argv[0]); //receive the pointer address from IDL
> *pointer.Test(); //not the way to do it

My suggestion is to simply abstract the problem away. I suggest you use a global
pointer array in your DLM:

```
#define MAX_LIST  10
...
static char* global_ptr[MAX_LIST]
```

Now, instead of passing the pointer back as a long, simply pass the index number as an
IDL_LONG. If you don't want to simply use the index number then you could create a
unique reference number derived from the index. This way you are not blindly
assuming your pointer can be passed around as an IDL_LONG (read: abstraction is good).
Of course, if you aren't happy with a static global pointer array, you could make it
dynamic and allocate space in chunks as they are needed. The abstraction, however,
should remain the same as you will pass back an index or derived index and not the
raw pointer. I have an example of this in my DLM for PostgreSQL communication with
IDL.

> I have a few .dlm's where I declare a global C++ object and use that
> object throughout the dlm. The dlm functions and procedures allow me to
> initialize the object, do stuff with it, and destroy it. Easy enough.
>
> The one limitation is that I only have one instance of the object
> available. If I run a second instance of a program that uses that .dlm the
> two IDL programs will overwrite the C++ object causing general mayhem
> (like using a common block in IDL).

The above array of pointers (or array of C++ objects) will help you here
as well. In this case, you can simply search your array of

pointers/objects for the first non-null slot and then create the pointer/object at that index. If there are no non-null indicies in your array, then you must either return an error or dynamically increase the array size. When you are done with the object in IDL, simply delete it based on the index number you have in IDL, and destroy the object/free the memory associated on the C side.

If this isn't clear, let me know and I'll try to write a quick and dirty example.

Cheers,
Randall
