Subject: Dynamically creating C++ objects in DLM
Posted by Rick Towler on Sat, 21 Dec 2002 00:21:45 GMT
View Forum Message <> Reply to Message

It has been a slow day here so maybe this can stir up some passion :)

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).

A solution to this problem would be dynamically creating the C++ object upon initialization and returning a pointer to the object back to IDL. When calling the dlm routines I could pass the pointer back to my dlm to gain access to my object of interest.

My questions are:

How do I dynamically create a C++ object (I think I use "new" but am a little unclear on the correct use)?

```
    myObject *test = new myObject;   //?????
```

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
```

Thanks for any thoughts.

-Rick

Subject: Re: Dynamically creating C++ objects in DLM
Posted by Nigel Wade on Thu, 02 Jan 2003 12:04:14 GMT
View Forum Message <> Reply to Message

Rick Towler wrote:

> It has been a slow day here so maybe this can stir up some passion :)
>
> 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).
>
> A solution to this problem would be dynamically creating the C++ object
> upon
> initialization and returning a pointer to the object back to IDL. When
> calling the dlm routines I could pass the pointer back to my dlm to gain
> access to my object of interest.

A reasonable solution; I do this in several of my DLMs.

>
>
>
> My questions are:
>
> How do I dynamically create a C++ object (I think I use "new" but am a
> little unclear on the correct use)?
>
```
>     myObject *test = new myObject;   //?????
```

Not speaking C++ particularly well, I'd hazzard a guess that that's correct,
it looks ok. My DLM's are all in C.

>
> 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?
```

I wouldn't recommend this method. There's no guarentee that a long is the
same size as a pointer, or that it will remain so. What I do is to store
the pointer in a byte array which is created the same size as the pointer.

This is an example from one of my DLMs which needs to pass the value of the pointer PI_handle back to IDL in the 4th positional parameter. Using IDL_ImportArray allows you to define a callback routine which can free your allocated memory if the variable stored in IDL is erased, in this case it's called idl_PI_strategy_cb. Also note that IDL_ImportArray creates a *temporary* variable, so IDL_VarCopy doesn't allocate new memory; it uses the memory already allocated by a temp variable :

```
   UCHAR *store;
   IDL_VPTR new_array;
   IDL_LONG dims[1];
   PI_Strategy *PI_handle = NULL;

   dims[0] = sizeof(PI_handle);
   store = (UCHAR *)malloc( dims[0] );
   memcpy( store, &PI_handle, dims[0]);
   new_array = IDL_ImportArray( 1, dims, IDL_TYP_BYTE,
        store, idl_PI_strategy_cb, NULL );
   IDL_VarCopy( new_array, parameters[3] );
```

I don't know how much of this can be executed directly in C++.

```
>
>
>  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
>
```

Again, not sure about C++, but in C I use memcpy to copy the data array back into a pointer:

```
   PI_Strategy *PI_handle;

   IDL_ENSURE_ARRAY(argv[0]);
   IDL_EXCLUDE_EXPR(argv[0]);
   if ( argv[0]->type != IDL_TYP_BYTE ||
       argv[0]->value.arr->n_dim != 1 ||
       argv[0]->value.arr->dim[0] != sizeof(PI_Strategy *) ||
       argv[0]->value.arr->free_cb != idl_PI_strategy_cb )
      IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
           "arg 1 is not a valid PI handle.");
   /*
    * copy the embedded strategy pointer from the IDL byte array into
    * the C pointer.
    */
   memcpy(&PI_handle, argv[0]->value.arr->data, sizeof(PI_Strategy *));
```

>
> Thanks for any thoughts.
>

Sorry I don't know C++ well enough to answer the question directly.
Hopefully some of the C code might help though.

--
Nigel Wade, System Administrator, Space Plasma Physics Group,
University of Leicester, Leicester, LE1 7RH, UK
E-mail : nmw@ion.le.ac.uk
Phone : +44 (0)116 2523568, Fax : +44 (0)116 2523555

Subject: Re: Dynamically creating C++ objects in DLM
Posted by Rick Towler on Thu, 02 Jan 2003 16:49:18 GMT
View Forum Message <> Reply to Message

"Nigel Wade" <nmw@ion.le.ac.uk> wrote

> Sorry I don't know C++ well enough to answer the question directly.
> Hopefully some of the C code might help though.

Neither do I! :/

Thanks for your response, Nigel. This is gold!

-Rick

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