Subject: DLM returning a pointer...
Posted by Randall Skelton on Mon, 23 Apr 2001 19:38:44 GMT
View Forum Message <> Reply to Message

Hi all,

I am trying to write a few IDL functions which mirror those of a C library I frequently use for getting data directly from a unix database connection. My problem is that the interface document states that the internal C structures should not be directly used; rather, an additional layer of abstraction should be used. The basic operation that I want in IDL is the ability to (1) connect to the database, (2) give a string of queries, (3) get a stream of data, and (4) close the connection. Ideally, I would like each of these steps to be a separate IDL function that mirrors the C functions. Implementing step 1 is proving to be somewhat difficult with the abstraction requirement. In order to establish a connection I use a C function:

DBcon *DBconSet(char *host, char *port, char *dbName, ...)

which returns DBconn (a nasty structure that, for the reasons above, I don't want to emulate directly in IDL). Nevertheless, this structure is passed in subsequent functions so I need the structure to proceed with steps 2, 3, and 4.

DBresult *DBexecute(DBconn *connection, const char *query)
char *DBgetval(DBconn *connection, int tup, int index)

Is it possible to return a pointer from C -> IDL such that I can pass a pointer of the DBcon memory block in subsequent functions from IDL -> C?

Does anyone have any code that shows how to do this?

Thanks in advance,
Randall

Subject: Re: DLM returning a pointer...
Posted by Craig Markwardt on Mon, 23 Apr 2001 21:01:01 GMT
View Forum Message <> Reply to Message

Randall Skelton <rhskelto@atm.ox.ac.uk> writes:
> Hi all,
>
> I am trying to write a few IDL functions which mirror those of a C library
> I frequently use for getting data directly from a unix database
> connection. My problem is that the interface document states that the
> internal C structures should not be directly used; rather, an additional

> layer of abstraction should be used. The basic operation that I want in
> IDL is the ability to (1) connect to the database, (2) give a string of
> queries, (3) get a stream of data, and (4) close the connection.
> Ideally, I would like each of these steps to be a separate IDL function
> that mirrors the C functions. Implementing step 1 is proving to be
> somewhat difficult with the abstraction requirement. In order to establish
> a connection I use a C function:
>
> DBcon *DBconSet(char *host, char *port, char *dbName, ...)
>
> which returns DBconn (a nasty structure that, for the reasons above, I
> don't want to emulate directly in IDL). Nevertheless, this structure is
> passed in subsequent functions so I need the structure to proceed with
> steps 2, 3, and 4.
... deleted ...

I don't think it would be wise to return a pointer, although technically it is possible. You could in principle cast the pointer to an integer, and return the integer. Then, in the subsequent calls, you would cast back to a pointer.

However, this is pretty dangerous since there is always a risk that the pointer gets corrupted while in IDL-land (for example, what if you set it to zero? Worse yet, what if it gets to some random part of memory which then gets overwritten).

A better choice I think is to have your interface routine internally maintain a *list* of DBconn connections. Then you can have the interface for DBconSet return an integer index into this list. Subsequent calls to the other library routines then simply have to look up the pointer in the list.

Error checking here is easier and safer as well, since you can verify that the index is GE 0 and LT N_PTR, and you can be sure that only your wrapper is modifying pointer values in the list. All you need is a little bookkeeping.

Good luck,
Craig

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: DLM returning a pointer...
Posted by Martin Schultz on Tue, 24 Apr 2001 09:51:10 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
>
> Randall Skelton <rhskelto@atm.ox.ac.uk> writes:
>> Hi all,
>>
>> I am trying to write a few IDL functions which mirror those of a C library
>>
>
> I don't think it would be wise to return a pointer, although
> technically it is possible. You could in principle cast the pointer
> to an integer, and return the integer.

... you probably meant an unsigned 64-bit value (in IDL speak
ULONG64).

Martin

--
[[[[[[[[[
[[Dr. Martin Schultz Max-Planck-Institut fuer Meteorologie [[
[[Bundesstr. 55, 20146 Hamburg [[
[[phone: +49 40 41173-308 [[
[[fax: +49 40 41173-298 [[
[[martin.schultz@dkrz.de [[
[[[[[[[[[

Subject: Re: DLM returning a pointer...
Posted by Randall Skelton on Tue, 24 Apr 2001 11:48:33 GMT
View Forum Message <> Reply to Message

After reading Craig's email, I am somewhat confused...

What I really need to know is how do I allocate a memory block in my C function such that I am sure that it cannot be overwritten or otherwise corrupted. I want a routine that can establish a connection to a given 'port' which requires me to allocate some memory and return a pointer to this memory block. I then want to be very sure that the memory allocated is 'safe' while now that I am back in IDL. Then I want to be able to pass this pointer from IDL back to C so as the database connection must be established before data can be sent or received. Finally, from IDL I would close the connection (again requiring me to pass this pointer) and de-allocate the memory. Obviously, such an implementation is risky as it

could lead to memory leaks in IDL if the programmer fails to close the connection properly. I am open to other ideas, but I want to separate the open and close connection functions. I am thinking about putting a time-out on the connection so that if idle for more than n minutes it deallocates. I fear, however, that a time-out would likely lead to problems and would be rather tricky to implement. It would be nice if there were some way to ensure that there was a matching 'open' and 'close' connection function with the IDL compiler...

Will the memory allocated with the IDL_GetScratch function span the forked C process life? i.e. if I use IDL_GetScratch to allocate memory, will IDL (potentially) cleanup and deallocate the memory before I call IDL_Deltmp()? What about IDL_MemAlloc and IDL_MemFree? Should I just consider defining an list say 10 of these structures with IDL_MemAllocPerm (giving me 10 possible connections) and forget about reclaiming the memory?

(I am assuming here that since IDL is calling the C program, this is a unix fork process giving C access to IDL's memory space alone. I am reluctant to use malloc directly in C as I doubt that IDL would respect the memory it allocates when I return to IDL).

All comments, suggestions and queries are greatly appreciated!

Randall Skelton

NB: just wait until I start asking how to make this multi threaded with asynchronous output from simultaneous connections ;)

On Tue, 24 Apr 2001, Martin Schultz wrote:

>>> Hi all,
>>>
>>> I am trying to write a few IDL functions which mirror those of a C library
>>> [...]
>>
>> I don't think it would be wise to return a pointer, although
>> technically it is possible. You could in principle cast the pointer
>> to an integer, and return the integer.
>
> ... you probably meant an unsigned 64-bit value (in IDL speak
> ULONG64).
>
> Martin

Subject: Re: DLM returning a pointer...

Posted by Richard Younger on Tue, 24 Apr 2001 14:34:53 GMT
View Forum Message <> Reply to Message

Randall Skelton wrote:
>
> After reading Craig's email, I am somewhat confused...
>
> What I really need to know is how do I allocate a memory block in my C
> function such that I am sure that it cannot be overwritten or otherwise
> corrupted. I want a routine that can establish a connection to a given
> 'port' which requires me to allocate some memory and return a pointer to
> this memory block. I then want to be very sure that the memory allocated
> is 'safe' while now that I am back in IDL. Then I want to be able to pass
> this pointer from IDL back to C so as the database connection must be
> established before data can be sent or received. Finally, from IDL I
> would close the connection (again requiring me to pass this pointer) and
> de-allocate the memory. Obviously, such an implementation is risky as it
> could lead to memory leaks in IDL if the programmer fails to close the
> connection properly. I am open to other ideas, but I want to separate the
> open and close connection functions. I am thinking about putting a
> time-out on the connection so that if idle for more than n minutes it
> deallocates. I fear, however, that a time-out would likely lead to
> problems and would be rather tricky to implement. It would be nice if
> there were some way to ensure that there was a matching 'open' and 'close'
> connection function with the IDL compiler...
>
> Will the memory allocated with the IDL_GetScratch function span the forked
> C process life? i.e. if I use IDL_GetScratch to allocate memory, will IDL
> (potentially) cleanup and deallocate the memory before I call
> IDL_Deltmp()? What about IDL_MemAlloc and IDL_MemFree? Should I just
> consider defining an list say 10 of these structures with IDL_MemAllocPerm
> (giving me 10 possible connections) and forget about reclaiming the
> memory?
>
> (I am assuming here that since IDL is calling the C program, this is a
> unix fork process giving C access to IDL's memory space alone. I am
> reluctant to use malloc directly in C as I doubt that IDL would respect
> the memory it allocates when I return to IDL).
>
> All comments, suggestions and queries are greatly appreciated!
>
> Randall Skelton
>
> NB: just wait until I start asking how to make this multi threaded with
> asynchronous output from simultaneous connections ;)
>
> On Tue, 24 Apr 2001, Martin Schultz wrote:
>
>>>> Hi all,

>>>>
>>>> I am trying to write a few IDL functions which mirror those of a C library
>>>> [...]
>>>
>>> I don't think it would be wise to return a pointer, although
>>> technically it is possible. You could in principle cast the pointer
>>> to an integer, and return the integer.
>>
>> ... you probably meant an unsigned 64-bit value (in IDL speak
>> ULONG64).
>>
>> Martin

Hi, Randall.

I've had a bit of a similar problem in a data acquisition routine. Fortunately (for me) the 3rd-party drivers I'm using don't require me to keep hunks of memory, just pointers out of IDL-space, so I can't speak too knowledgeably about the memory allocation issues.

But one thing to consider might be to use an object to store your pointers. Your object would have simple members to call your C functions. The size of unsigned IDL variable to use is probably best matched to your C pointer size, to 32- or 64- bit. Depends on your platform, I'd guess.

Here comes the good part. You can use the init and cleanup routines to enforce the off main level IDL allocation / deallocation requirement, and cleanup and deallocation should be done whenever the object is destroyed by IDL. You can also create as many connections as you like without having to resort to global memory, or worrying about some connections stepping on others. I'd think you could use IDL_MemAlloc() to get the memory, since you're more or less assured to deallocate it later, but I'm not positive.

Rich

--
Richard Younger MIT Lincoln Laboratory
Email: younger@ll.mit.edu Mail Stop C-130
Phone: (781)981-4464 244 Wood St.
Fax: (781)981-0122 Lexington, MA 02144-9108

Subject: Re: DLM returning a pointer...
Posted by Richard Younger on Tue, 24 Apr 2001 15:27:00 GMT
View Forum Message <> Reply to Message

PS-
The object is another layer of abstraction, but it needn't be difficult.
IDL code might look something like this, ignoring argument checking,
error checking, etc. You can do that either on this level or on the
level of the C wrapper routines.

```
.....................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; File: DBinterface__define.pro

FUNCTION DBinterface::init

 self.struct_ptr = DBconnect_wrap
 RETURN, 1

END ;init

;-----------
PRO DBinterface::cleanup

 error = DBclose_wrap(self.struct_ptr)
 self.struct_ptr = 0ULL

END ;cleanup

;-----------
FUNCTION DBinterface::DBexecute, query

 RETURN, DBexec_wrap(self.struct_ptr, query)

END

;-----------
FUNCTION DBinterface::DBgetval, index

 RETURN, DBgetv_wrap(self.struct_ptr, index)

END

;-----------
PRO DBinterface__define

 void = { $
  struct_ptr : 0ULL  $
  }

END ;DBinterface__define
.....................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Cheers,
Rich

--
Richard Younger
Assistant Technical Staff MIT Lincoln Laboratory
Electro-Optical Materials and Devices Mail Stop C-130
Email: younger@ll.mit.spmdecoy.edu 244 Wood St.
Phone: (781)981-4464 Lexington, MA 02144-9108

Subject: Re: DLM returning a pointer...
Posted by Craig Markwardt on Tue, 24 Apr 2001 19:06:14 GMT
View Forum Message <> Reply to Message

Randall Skelton <rhskelto@atm.ox.ac.uk> writes:
> After reading Craig's email, I am somewhat confused...
>
> What I really need to know is how do I allocate a memory block in my C
> function such that I am sure that it cannot be overwritten or otherwise
> corrupted. I want a routine that can establish a connection to a given
> 'port' which requires me to allocate some memory and return a pointer to
> this memory block. I then want to be very sure that the memory allocated
> is 'safe' while now that I am back in IDL. Then I want to be able to pass
> this pointer from IDL back to C so as the database connection must be
> established before data can be sent or received. Finally, from IDL I
> would close the connection (again requiring me to pass this pointer) and
> de-allocate the memory. Obviously, such an implementation is risky as it
> could lead to memory leaks in IDL if the programmer fails to close the
> connection properly. I am open to other ideas, but I want to separate the
> open and close connection functions. I am thinking about putting a
> time-out on the connection so that if idle for more than n minutes it
> deallocates. I fear, however, that a time-out would likely lead to
> problems and would be rather tricky to implement. It would be nice if
> there were some way to ensure that there was a matching 'open' and 'close'
> connection function with the IDL compiler...

Randall, if I'm not mistaken, it's the job of the DBconset function to allocate the memory for the DBcon structure, so in principle you don't have to worry about that. What I am suggesting is to maintain a list of such pointers in your C wrapper. So it could be something like the below. Here I have made a static list of 10 DBcon pointers, which can hold up to ten simultaneous connections. If you want to get smarter, you could dynamically allocate memory if the number of simultaneous connections is unknown and > 10.

Of course I haven't done any error checking, and the dealing with IDL

<-> C translation isn't handled here. I'm just showing how a C pointer can be converted to and from an integer value using a lookup table (the table is dbcons[] in this case).

The question of allocating memory is really a separate question altogether. My personal feeling is that there is no harm to use malloc(), but you can use the IDL_MemAlloc() too. Both of these are for raw memory. The other functions are for allocating IDL variables, which carry along other baggage which you don't need here, and as you say, IDL can trample on that data. Of course it will persist across function calls. [If your DLM is unloaded with .IDL_SUPER_SESSION_RESET or whatever I have no idea what happens.]

Good luck,
Craig

```
#define DBMAXCON 10
/* Global lookup table - default initialized to zero */
static DBcon *dbcons[DBMAXCON];

int dbconset_wrapper(char *host, char *port, char *dbname, ...)
{
  int i;
  DBcon *dbcon;

  /* Search for an available slot - one with a zero */
  for (i=0; i<DBMAXCON; i++)
    if (dbcons[i] == 0) break;
  if (i == DBMAXCON) { Maximum connections reached! }

  /* Open the connection and return */
  dbcon = DBconSet(host, port, dbname, ...);
  if (dbcon == 0) { report error! }

  dbcons[i] = dbcon;
  return i;
}

int dbclose_wrapper(int num)
{
  /* Error checking */
  if (i < 0 || i >= DBMAXCON || dbcons[i] == 0) { Invalid handle! }

  DBclose(dbcons[i]);
  dbcons[i] = 0;      /* Reset to zero so it can be reused. */
  return 0;
}
```

DBcon *DBconSet(char *host, char *port, char *dbName, ...)

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: DLM returning a pointer...
Posted by Randall Skelton on Tue, 24 Apr 2001 21:04:38 GMT
View Forum Message <> Reply to Message

Wow... I am now well on my way with this little project thanks to Craig, Martin, Richard (and a few emails from Jim). You guys are great!

All of the suggestions given have been very helpful and I have incorporated many of them into my code. I must admit to have been rather confused regarding the need for allocating memory in IDL, but Craig's last email and code has resolved much of this.

Thanks again,
Randall

On 24 Apr 2001, Craig Markwardt wrote:

> Randall, if I'm not mistaken, it's the job of the DBconset function to
> allocate the memory for the DBcon structure, so in principle you don't
> have to worry about that. What I am suggesting is to maintain a list
> of such pointers in your C wrapper. So it could be something like the
> below. Here I have made a static list of 10 DBcon pointers, which can
> hold up to ten simultaneous connections. If you want to get smarter,
> you could dynamically allocate memory if the number of simultaneous
> connections is unknown and > 10.
>
> Of course I haven't done any error checking, and the dealing with IDL
> <-> C translation isn't handled here. I'm just showing how a C
> pointer can be converted to and from an integer value using a lookup
> table (the table is dbcons[] in this case).
>
> The question of allocating memory is really a separate question
> altogether. My personal feeling is that there is no harm to use
> malloc(), but you can use the IDL_MemAlloc() too. Both of these are
> for raw memory. The other functions are for allocating IDL variables,

> which carry along other baggage which you don't need here, and as you
> say, IDL can trample on that data. Of course it will persist across
> function calls. [If your DLM is unloaded with
> .IDL_SUPER_SESSION_RESET or whatever I have no idea what happens.]
>
> Good luck,
> Craig
>
>
[snip]
