Subject: object memory management
Posted by Anonymous on Wed, 12 Nov 2008 17:31:43 GMT
View Forum Message <> Reply to Message

Originally posted by: Demitri

Quick on the heels of my previous question about empty arrays... I have a question about memory management.

Let's say I have a function that will return an array, but as it can be empty, I'd like to return an IDL_Container instead. No problem:

```
FUNCTION f
 container = NEW_OBJ('IDL_CONTAINER')
 container->add, NEW_OBJ('my_obj')
 container->add, NEW_OBJ('my_obj')

 return, container
END
```

(Let's ignore the memory management of the 'my_obj's for the moment.) Another method calls this and gets the container, but now the responsibility to destroy that object is in the hands of the calling routine, where it's not obvious (or maybe depending on the type it is?) that it will need to be freed by hand.

<Mac programmers only>
In Obj-C, this problem solved by the autorelease / retain messages, which of course IDL doesn't have. But that's the first thing I thought of.
</Mac programmers only>

Is this something that should be published in my class' API and the responsibility is passed to anyone using the function? It seems that calling OBJ_DESTROY will also destroy the objects within the container, and I may not want that. Should I ignore it and call HEAP_GC occasionally (*cough*hack!*cough*)? What is the IDL convention here?

Cheers,

Demitri

Subject: Re: object memory management
Posted by David Fanning on Wed, 12 Nov 2008 18:23:10 GMT
View Forum Message <> Reply to Message

Demitri writes:

> Quick on the heels of my previous question about empty arrays... I have
> a question about memory management.
>
> Let's say I have a function that will return an array, but as it can be
> empty, I'd like to return an IDL_Container instead. No problem:
>
> FUNCTION f
> container = NEW_OBJ('IDL_CONTAINER')
> container->add, NEW_OBJ('my_obj')
> container->add, NEW_OBJ('my_obj')
>
> return, container
> END
>
> (Let's ignore the memory management of the 'my_obj's for the moment.)
> Another method calls this and gets the container, but now the
> responsibility to destroy that object is in the hands of the calling
> routine, where it's not obvious (or maybe depending on the type it is?)
> that it will need to be freed by hand.
>
> <Mac programmers only>
> In Obj-C, this problem solved by the autorelease / retain messages,
> which of course IDL doesn't have. But that's the first thing I thought
> of.
> </Mac programmers only>
>
> Is this something that should be published in my class' API and the
> responsibility is passed to anyone using the function? It seems that
> calling OBJ_DESTROY will also destroy the objects within the container,
> and I may not want that. Should I ignore it and call HEAP_GC
> occasionally (*cough*hack!*cough*)? What is the IDL convention here?

This is always a dilemma. I've tried various things. I've even
returned undefined variables, on the theory that whoever is going
to get the value might check to see if the variable is defined:

```
ptr = Ptr_New(/Allocate_Heap)
RETURN, *ptr
```

This never works, because users (including me) never check. Although
it does cause them grief, which is something. :-)

I've tried documenting the user's responsibilities in the documentation
of the API. Want to guess how well *that* worked? :-(

In the Catalyst Library, where I do most of my object programming, we
implemented reference counting. Objects are not destroyed until either

all of their parents have released them, or their parents are destroyed.
This works *very* well, and I almost never have problems with leaking
memory. (As long as I remember to call the superclass CLEANUP method in
the CLEANUP method of the objects I need to write.)

I'm not sure there is a Real Good solution, although I'm pretty
sure HEAP_GC is *not* the answer. :-)

Cheers,

David
--
David Fanning, Ph.D.
Coyote's Guide to IDL Programming (www.dfanning.com)
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: object memory management
Posted by Anonymous on Wed, 12 Nov 2008 22:50:55 GMT
View Forum Message <> Reply to Message

Originally posted by: Demitri

Thanks for the reply David. I think you're right - it's best not to
have a solution that pushes the problem onto another user.

I thought of another solution that I'll list here for posterity. Since
so much IDL code seems to be checking for these things anyway (is this
defined? is the number of elements zero?) I might as well stick with
what people are used to, but make it a little easier:

```
FUNCTION f, count_returned
 container = OBJ_NEW('IDL_CONTAINER')
 container->add, OBJ_NEW('my_obj')
 container->add, OBJ_NEW('my_obj')

 count_returned = container->count()

 if (count_returned gt 0) then begin
  list_to_return = container->get(/all)
  OBJ_DESTROY(container)
  return, list_to_return
 endif else begin
  OBJ_DESTROY(container)
  return, 0
 end
END
```

This is really the C way of doing things. It's a little more code, but
since it's under the hood I'm not too worried, and there's no chance of
a memory leak. The user is still required to check that
"count_returned" is nonzero, otherwise to consider the result undefined
(although it is in fact zero). Pretty much any other solution is going
require some testing and checking anyway.

Cheers,

Demitri

On 2008-11-12 13:23:10 -0500, David Fanning <news@dfanning.com> said:

> Demitri writes:
>
>> Quick on the heels of my previous question about empty arrays... I have
>> a question about memory management.
>>
>> Let's say I have a function that will return an array, but as it can be
>> empty, I'd like to return an IDL_Container instead. No problem:
>>
>> FUNCTION f
>> container = NEW_OBJ('IDL_CONTAINER')
>> container->add, NEW_OBJ('my_obj')
>> container->add, NEW_OBJ('my_obj')
>>
>> return, container
>> END
>>
>> (Let's ignore the memory management of the 'my_obj's for the moment.)
>> Another method calls this and gets the container, but now the
>> responsibility to destroy that object is in the hands of the calling
>> routine, where it's not obvious (or maybe depending on the type it is?)
>> that it will need to be freed by hand.
>>
>> <Mac programmers only>
>> In Obj-C, this problem solved by the autorelease / retain messages,
>> which of course IDL doesn't have. But that's the first thing I thought
>> of.
>> </Mac programmers only>
>>
>> Is this something that should be published in my class' API and the
>> responsibility is passed to anyone using the function? It seems that
>> calling OBJ_DESTROY will also destroy the objects within the container,
>> and I may not want that. Should I ignore it and call HEAP_GC

>> occasionally (*cough*hack!*cough*)? What is the IDL convention here?
>
> This is always a dilemma. I've tried various things. I've even
> returned undefined variables, on the theory that whoever is going
> to get the value might check to see if the variable is defined:
>
> ptr = Ptr_New(/Allocate_Heap)
> RETURN, *ptr
>
> This never works, because users (including me) never check. Although
> it does cause them grief, which is something. :-)
>
> I've tried documenting the user's responsibilities in the documentation
> of the API. Want to guess how well *that* worked? :-(
>
> In the Catalyst Library, where I do most of my object programming, we
> implemented reference counting. Objects are not destroyed until either
> all of their parents have released them, or their parents are destroyed.
> This works *very* well, and I almost never have problems with leaking
> memory. (As long as I remember to call the superclass CLEANUP method in
> the CLEANUP method of the objects I need to write.)
>
> I'm not sure there is a Real Good solution, although I'm pretty
> sure HEAP_GC is *not* the answer. :-)
>
> Cheers,
>
> David

Subject: Re: object memory management
Posted by Craig Markwardt on Thu, 13 Nov 2008 05:18:41 GMT
View Forum Message <> Reply to Message

On Nov 12, 12:31 pm, Demitri wrote:
>
> Is this something that should be published in my class' API and the
> responsibility is passed to anyone using the function? It seems that
> calling OBJ_DESTROY will also destroy the objects within the container,
> and I may not want that. Should I ignore it and call HEAP_GC
> occasionally (*cough*hack!*cough*)? What is the IDL convention here?

This is exactly the same problem as, "who frees a pointer?" In a low level language like C, this question makes sense to ask. But for a high level language like IDL (or Python or Perl etc.), the *interpreter* does the job for us. Why should we have to figure out when an object or a pointer needs to be freed when the interpreter knows *exactly* when? [for example it can do its own reference

counting.] Just think if we had to explicitly allocate and free
every variable. In my opinion, it's an area where RSI really dropped
the ball, and makes pointers/objects a lot less effective than they
could be.

Craig

Subject: Re: object memory management
Posted by Anonymous on Thu, 13 Nov 2008 18:57:11 GMT
View Forum Message <> Reply to Message

Originally posted by: Demitri

On 2008-11-13 00:18:41 -0500, Craig Markwardt <cbmarkwardt@gmail.com> said:

> On Nov 12, 12:31ï¿½pm, Demitri wrote:
>>
>> Is this something that should be published in my class' API and the
>> responsibility is passed to anyone using the function? It seems that
>> calling OBJ_DESTROY will also destroy the objects within the container,
>> and I may not want that. Should I ignore it and call HEAP_GC
>> occasionally (*cough*hack!*cough*)? What is the IDL convention here?
>
> This is exactly the same problem as, "who frees a pointer?" In a low
> level language like C, this question makes sense to ask. But for a
> high level language like IDL (or Python or Perl etc.), the
> *interpreter* does the job for us. Why should we have to figure out
> when an object or a pointer needs to be freed when the interpreter
> knows *exactly* when? [for example it can do its own reference
> counting.] Just think if we had to explicitly allocate and free
> every variable. In my opinion, it's an area where RSI really dropped
> the ball, and makes pointers/objects a lot less effective than they
> could be.
>
> Craig

So basically, IDL is presenting us with what they are calling a
beautiful woman, but we're looking and saying, "Um, clearly that's a
man in makeup. The beard is a dead giveaway."

Thanks Craig for the comments. (BTW, your mpfit.pro rocks!)

Subject: Re: object memory management
Posted by David Fanning on Thu, 13 Nov 2008 19:03:28 GMT
View Forum Message <> Reply to Message

Demitri writes:

> So basically, IDL is presenting us with what they are calling a
> beautiful woman, but we're looking and saying, "Um, clearly that's a
> man in makeup. The beard is a dead giveaway."

I have a feeling you are going to fit in very well
around here, Demitri. :-)

Cheers,

David
--
David Fanning, Ph.D.
Coyote's Guide to IDL Programming (www.dfanning.com)
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: object memory management
Posted by Craig Markwardt on Fri, 14 Nov 2008 03:29:04 GMT
View Forum Message <> Reply to Message

On Nov 13, 1:57 pm, Demitri wrote:
> On 2008-11-13 00:18:41 -0500, Craig Markwardt <cbmarkwa...@gmail.com> said:
>
>> This is exactly the same problem as, "who frees a pointer?" In a low
>> level language like C, this question makes sense to ask. But for a
>> high level language like IDL (or Python or Perl etc.), the
>> *interpreter* does the job for us. Why should we have to figure out
>> when an object or a pointer needs to be freed when the interpreter
>> knows *exactly* when? [for example it can do its own reference
>> counting.] Just think if we had to explicitly allocate and free
>> every variable. In my opinion, it's an area where RSI really dropped
>> the ball, and makes pointers/objects a lot less effective than they
>> could be.
...
> So basically, IDL is presenting us with what they are calling a
> beautiful woman, but we're looking and saying, "Um, clearly that's a
> man in makeup. The beard is a dead giveaway."

And the hairy legs!

> Thanks Craig for the comments. (BTW, your mpfit.pro rocks!)

You're welcome :-)
