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.")
