Subject: Re: Important object lesson
Posted by Mark Hadfield on Tue, 30 Jun 1998 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote in message ...
>> Much deleted
>
> Yes, it must be that when the first object definition is made that
> the two lifecycle methods are "registered" for the object. Probably
> much the way keywords are registered for procedures and functions
> when they are compiled. If you inadvertently forget to define an INIT
> or CLEANUP method, and you have already made an object of that class,
> then you must exit IDL and start over for those INIT and CLEANUP
> methods to be recognized. This does NOT apply to other methods,
> however, which you can add in the same way you add other procedures
> and functions to programs.

This is my current working hypothesis for how all this works, based on a little experimentation, a modest amoung of logic, generous conjecture & a minimal scanning of the documentation...

If IDL encounters

    MyClass->MyMethod

the three situations are:

 1. IDL finds a MyClass::Method in memory and uses it. (In the normal course of events the method will have been included in the myclass__define.pro, before the myclass__define procedure, so it will have been compiled the first time an instance of the class was created.) If MyClass::MyMethod is recompiled, the modifications are recognised.

2. Not finding MyClass::Method, IDL searches up the inheritance tree, finds a ASuperClass::Method in memory and uses it for the remainder of the session. If MyClass::MyMethod is recompiled, the modifications are not recognised, because this method is never called. Which is confusing.

3. Failing 1 & 2, IDL searches the !path for myclass__mymethod.pro (and maybe then for similar files for all superclasses). This can take a while. For ordinary methods, failure to find it results in an error. Obj_new and obj_destroy look for an Init and Cleanup respectively, but if they fail to find them, they just skip that step--until the next time & the next time & ...etc.

> Indeed. I am beginning to wonder if it is not one source
> for the slowness of objects in general, apart from the
> problems of just manipulating this huge 3D space.

I don't think objects themselves are all that slow. Objects are just references to named structures on the heap, which are pretty lightweight things, much like pointers. You can have a few tens of thousands of the things before there is any slowdown. Binding of methods to objects is much like resolving procedures, and not all that slow, except in situations where methods can't be found, as above.

Object Graphics are slow because they hold so much more information than graphics windows--when you draw a 10,000-point plot to a Direct Graphics window you end up with a bunch of pixels; when you add it to a model & a view & a window, you still have all those points in 3D space.

>> It also suggests a solution (though not a very elegant one): make all your
>> objects subclasses of something, if only a dummy class, and make sure one of
>> the superclasses has explicit Init and Cleanup methods.
>
> I feel certain you are going to try this, Mark. Could you
> fill us in on the result? :-)

Are you suggesting that I am a software fiddler? I'm afraid it's true. I did think a while back about having a general class called Object with genreally useful behaviours that all my other classes could descend from, as in Java. Trouble is, I couldn't think of anything very useful for Object to do and it still wouldn't be available for IDL built-in classes. Re the present situation, it's debatable whether my suggestion is more or less trouble than adding non-functional Inits & Cleanups to all classes.

--
Mark Hadfield, m.hadfield@niwa.cri.nz http://www.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand