Subject: Re: Object Madness or Restoring Nightmares
Posted by JD Smith on Wed, 03 Mar 2004 22:15:02 GMT
View Forum Message <> Reply to Message

On Wed, 03 Mar 2004 13:26:36 -0700, David Fanning wrote:

> JD Smith writes:
>
>> I think it's actually simpler than that. I suspect that if you follow
>> the train of objects containing pointers to objects with pointers
>> etc., you'll find that some object somewhere beneath your "theStudy"
>> object actually has a pointer or object reference to the top-level
>> application object in it.
>
> OK, I like this theory. Here is my problem. (Those of you whose
> eyes are already glazing over are excused. You can read the
> Executive Summary to follow in a couple of days.)
>
> My study object contains three object references: one to the imageCube
> object that exists at the main application level and which is stored
> as a field in the top-level object, and two to IDL container objects.
> One container is empty, so don't worry about it. The other
> container contains ImageSlice objects, which are abstractions simply
> to draw a particular slice of the imageCube object. As such, they
> also contain references to the original imageCube object. The only
> other object an ImageSlice contains is a WindowIndex number reference
> to a drawWidgetObject, that is a child of the "self" or main object.
> This may be your connection back to the main object.
>
> OK, before saving the studyObject I null out the ImageCube reference,
> and I go through both containers, get all the ImageSlice objects and
> null out their imageCube references and their WindowIndex references.
> The imageSlice objects now don't point to anything, and the studyObject
> only has references to ImageSlice objects that should be nulled out.
>
> So now, I get the original data from the ImageCube object everyone has
> been pointing to, and I save it *along* with the the studyObject in
> a save file. (I need the data, that is the most important part of a
> "study".)
>
> Are you ready? I am chagrined to see that my save file (which you
> remember should have been about 3 MB and was already a bloated
> 10 MB) is now nearly 13 MB!. And, I *still* have 1203 unnecessary
> objects stored in my save file. I take it my save file is now all
> the objects from before *plus* the real data.

Hmmm, this tells me you didn't manage to prune out the problem bit.
Just as an exercise, start off with your original save method, nothing

funny, and start by nulling out one object or pointer at a time anywhere in theStudy. Between each such pruning, save the object and examine its contents, file size, etc. I suspect at some point you'll find the magic ladder (think chutes and ladders) which leads all the way back to the top, and your file sizes will drop and you'll have rid yourself of the bulk of those unwanted saved objects. Once you've found the wayward object or pointer, re-attach everything, descend one level into it and repeat, until you isolate the miscreant path. Of course, maybe this is what you mean when you say you went over it with a fine-toothed comb.

> Now, *all* objects in this program inherit from a single object
> class. I can understand if we had to save down to that object
> class. But I think IDL got down to that object class (CATATOM,
> by the way) and saved *all* the current objects that inherit from
> that class! That is the only explanation that makes sense to me.

That's far fetched I think. Unless this is an out-and-out bug whereby the heap descent code gets lost, I can't imagine how it would occur.

> If that is true, I begin to understand some of the complexity of
> the iTools system, which doesn't work with objects at all, but
> with "descriptions" of objects. They would absolutely have to
> do this or they could never save and restore any of their iTool
> objects. Someone there must have run into this problem.
>
>
>> If this is the case, here's what happens: IDL very dutifully follows all
>> of these downward-linking object/pointer chains, collecting and saving
>> everything it finds on the way. This is the correct thing to do, since,
>> as far as it knows, to have a valid "theStudy" object on disk requires
>> all of its various holdings. Now, if at some point down the chain, IDL
>> runs into an object which is just a convenience reference to the top
>> level application object, it will dutifully jump right to the top of the
>> heap and start saving the whole thing.
>
> I can't think how, in its current configuration, IDL could possibly get
> back to the top. I've been through these objects with a fine-tooth comb.
> There are *no* valid object references except to containers of objects
> that do not have valid object references.

Remember, it's not just objects, but any data structure (pointer, structure, array of the same, etc.) which can contain object references which point back to the top. Even cached messages, anything.

>> This is a problem. It's actually a bigger problem than you think,
>> because (see the various articles on your site describing it), any

>> object which is saved has implicit in it its class definition, so if you
>> accidentally save 10 extra objects of different classes along with the
>> one you're really interested in, when you restore them, any updates to
>> any of the class definition files (class__define.pro) will never be
>> consulted, since IDL thinks it already knows all about them. The
>> much-discussed solution is to explicitly resolve the class *before*
>> restoring the object. You can find my latest incarnation of my routine
>> which automates this here:
>>
>> http://turtle.as.arizona.edu/idl/restore_object.pro
>
> Alas, that EXECUTE statement makes this virtually useless
> to me except as an academic exercise. :-(

It's only necessary if you have more than one object of the desired
class in the file, and you only want the one which corresponds to a
given variable name (like "self"). I could also use ROUTINE_NAMES
tricks to accomplish the same without EXECUTE, but that would be
cheating, i.e. instead of:

```
tmp=execute('thisvar='+var)
```

use

```
tmp=routine_names(var,FETCH=0)
```

but you didn't hear it from me.

> I've preferred not to think about this for the moment.
> I'm just assuming the client is always working with a fully
> compiled project so that which definition we are using is well-defined.
> That will probably bite me later, as this project just never seems
> to go away.

If the code never changes, that's fine. But what happens when you
upgrade the class definition to include that new whiz-bang object, and
the client would like to use his old saved projects with the new
version? You've killed backward compatibility.

>> So, how do you avoid this situation? What I do is "detach" all the
>> irrelevant data from my object before saving it. I've talked about this
>> before, but the basic idea is (in your terms):
>>
>> theStudy = self.currentStudy
>> theStudy->Save,'somename.sav'
>>
>> with
>>

```
>>  pro theStudy::Save,filename
>>     saved_ptr=self.BigAndUselessDataPtr ; detach
>>     self.BigAndUselessDataPtr=ptr_new() ; a null pointer
>>     save, self,FILENAME=filename
>>     self.BigAndUselessDataPtr=saved_ptr  ; reattach
>>  end
>>
>>
>>  and to restore it:
>>
>>  theStudy=restore_object(file,'theStudy')
>>  if obj_valid(theStudy) then begin
>>     if NOT obj_isa(theStudy,'theStudy') then $
>>        message,'Error restoring Study file: '+file
>>     ;; The study is valid
>>     obj_destroy,self.currentStudy
>>     self.currentStudy=theStudy
>>  endif
>>
>>  This requires, of course, that you plan ahead and group all of the data
>>  that isn't necessary to include in the save file in some conveniently
>>  detachable object or pointer (or perhaps a few of them).  Aside
>>  from convenience object references, widget data is a good
>>  candidate for detachment.  Detaching an object reference works just the
>>  same, but with "obj_new()" instead of "ptr_new()".
>
>  If I understand you correctly, this is exactly what I have
>  tried to do, and I find myself worse off than before. I've
>  appealed to the IDL newsgroup because the RSI technical support
>  people don't exactly like to hear from me with my "big file"
>  examples. :-)
```

I wouldn't say you're worse off, just the same (since you added something else to the save as well). I think you'll eventually find it. Part of the reason I communicate among objects using messages is to avoid problems like this: instead of having to hold a reference to an object to query it for properties on occasion, you can just subscribe to messages which give you the relevant properties and the right time (i.e. property push not pull). Since the messages are dealt with as they come, they are ephemeral, and don't stick around to cause problems like this.

JD
