Subject: Re: Yet another object graphics question
Posted by Antonio Santiago on Thu, 24 Feb 2005 14:38:48 GMT
View Forum Message <> Reply to Message

Michael Wallace wrote:
> Say that you want to write a utility function which will create a basic
> plot. Let's say this function returned an IDLgrModel with some
> IDLgrPlots on the the inside and various axes and the like. Because all
> of the objects are in a single tree, destroying them is a snap -- just
> destroy the top level object and the destruction cascades down.
>
> Now what do you do if you want to create an IDLgrFont or other "helper"
> object inside the utility function? You can't destroy the helper
> because you'll get an invalid object reference where it had been used
> and once you fall out of the function, you won't have a named variable
> reference to the helper object. The helper will still be present on the
> heap, but there isn't any name to pass obj_destroy. Once I finish using
> the IDLgrModel returned from the function, I can destroy it, but the
> helpers are left dangling.
>
> Is there any rule of thumb ya'll follow for cases like this? I don't
> want to have heap_gc commands in my code just to clean up after myself.
> :-)
>
> -Mike

You can use an IDL_Container object to contains all helper object (like IDLgrFont).
In my case, i store the reference in an IDL_Conteiner. At the moment of the destruction, one called to
OBJ_DESTROY, container
destroy all its associated object.

In my particular case, I use IDL objects to work with Object Graphics and many times stores references to helper objects as class attributes. Perhaps it will be usefull for you.

Bye.
Antonio.

Subject: Re: Yet another object graphics question
Posted by btt on Thu, 24 Feb 2005 14:40:10 GMT
View Forum Message <> Reply to Message

Michael Wallace wrote:
> Say that you want to write a utility function which will create a basic
> plot. Let's say this function returned an IDLgrModel with some

> IDLgrPlots on the the inside and various axes and the like. Because all
> of the objects are in a single tree, destroying them is a snap -- just
> destroy the top level object and the destruction cascades down.
>
> Now what do you do if you want to create an IDLgrFont or other "helper"
> object inside the utility function? You can't destroy the helper
> because you'll get an invalid object reference where it had been used
> and once you fall out of the function, you won't have a named variable
> reference to the helper object. The helper will still be present on the
> heap, but there isn't any name to pass obj_destroy. Once I finish using
> the IDLgrModel returned from the function, I can destroy it, but the
> helpers are left dangling.
>
> Is there any rule of thumb ya'll follow for cases like this? I don't
> want to have heap_gc commands in my code just to clean up after myself.

Hi,

I'm sure there are plenty of paradigms out there (and I am quite curious to see how others handle this), but I have fallen into a simple one. Here's the definition of a graphic axis similar to one I have used...

```
struct = {My_Axis , $
 INHERITS IDLgrAxis, $
 LocalFont: 0, $
 Font: OBJ_NEW()}
```

And here is the cleanup statement...

```
If (Self.LocalFont NE 0) AND (Obj_Valid(self.Font)) Then Obj_Destroy, self.Font
```

I have a method SETFONT that accepts either and IDLgrFont or a string description of the font (just like in the INIT of an IDLgrFont object.) In the latter case, the font is considered 'local' and will be cleanup up by this object, otherwise I assume that the font is being managed 'globally' and that some other object will be cleaning it up. I assume that the title and ticktext share the same font; keeping a reference to it in the ticktext, the title and in the class definition isn't very "heavy" if you know what I mean.

Cheers,
Ben

Subject: Re: Yet another object graphics question
Posted by David Fanning on Thu, 24 Feb 2005 14:42:05 GMT
View Forum Message <> Reply to Message

Antonio Santiago writes:

> You can use an IDL_Container object to contains all helper object (like
> IDLgrFont).
> In my case, i store the reference in an IDL_Conteiner. At the moment of
> the destruction, one called to
> OBJ_DESTROY, container
> destroy all its associated object.
>
> In my particular case, I use IDL objects to work with Object Graphics
> and many times stores references to helper objects as class attributes.
> Perhaps it will be usefull for you.

Yes, good advice, especially the object part. I was just looking to
see how I did this, and it is always with objects.

But I was thinking that your main program could have a garbage
container, and that you could pass a reference to that to
your function, too. The function could put all the "helper"
objects into the container before it returned. That way you
have one container to destroy at the end and you get everything.

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Yet another object graphics question
Posted by Antonio Santiago on Thu, 24 Feb 2005 17:31:25 GMT
View Forum Message <> Reply to Message

> And here is the cleanup statement...
>
> If (Self.LocalFont NE 0) AND (Obj_Valid(self.Font)) Then Obj_Destroy,
> self.Font

It is really necesary the "if" test about the object validity ?? :)

```
PRO My_Axis::Cleanup
 OBJ_DESTROY, self.oFont
 self->IDLgrAxis::Cleanup
END
```

Bye,

Antonio.

Subject: Re: Yet another object graphics question
Posted by Antonio Santiago on Thu, 24 Feb 2005 17:43:21 GMT
View Forum Message <> Reply to Message

> And here is the cleanup statement...
>
> If (Self.LocalFont NE 0) AND (Obj_Valid(self.Font)) Then Obj_Destroy,
> self.Font

This is only an observation, but it is really necessary the OBJ_VALID()
call?

IF self.LocalFont NE 0 THEN OBJ_DESTROY, self.Font

Bye,
Antonio.

Subject: Re: Yet another object graphics question
Posted by btt on Thu, 24 Feb 2005 17:55:36 GMT
View Forum Message <> Reply to Message

Antonio Santiago wrote:

>
>> And here is the cleanup statement...
>>
>> If (Self.LocalFont NE 0) AND (Obj_Valid(self.Font)) Then Obj_Destroy,
>> self.Font
>
>
>
> This is only an observation, but it is really necessary the OBJ_VALID()
> call?
>
> IF self.LocalFont NE 0 THEN OBJ_DESTROY, self.Font
>
Well, you are probably right. This code, like all my code has evolved from
something else - so it probably made sense somewhere back along its journey (or
maybe it didn't make sense even then).

Subject: Re: Yet another object graphics question
Posted by Mark Hadfield on Thu, 24 Feb 2005 20:15:37 GMT
View Forum Message <> Reply to Message

Ben Tupper wrote:
> I'm sure there are plenty of paradigms out there (and I am quite curious
> to see how others handle this), but I have fallen into a simple one.
> Here's the definition of a graphic axis similar to one I have used...

I always store helper objects (IDLgrFont, IDLgrSymbol) in a container attached to the view. This seems like the right place, as they are often used by several different atoms in a view (eg an IDLgrFont will be used by several axes, an IDLgrSymbol will used by an IDLgrPlot object and by an IDLgrLegend). In fact, most of the "intelligence" in my Object graphics code is attached to view objects, in the form of methods of a subclass of IDLgrView.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: Yet another object graphics question
Posted by David Fanning on Thu, 24 Feb 2005 20:23:39 GMT
View Forum Message <> Reply to Message

Mark Hadfield writes:

> I always store helper objects (IDLgrFont, IDLgrSymbol) in a container
> attached to the view.

That makes sense, too. Gosh, you are *all* making sense today. Weird. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Yet another object graphics question
Posted by Mark Hadfield on Thu, 24 Feb 2005 20:42:08 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Mark Hadfield writes:
>
>> I always store helper objects (IDLgrFont, IDLgrSymbol) in a container
>> attached to the view.
>
> That makes sense, too. Gosh, you are *all* making sense
> today. Weird. :-)

Step up the dose, man.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: Yet another object graphics question
Posted by Michael Wallace on Fri, 25 Feb 2005 21:44:12 GMT
View Forum Message <> Reply to Message

Hey guys, thanks for all the responses. It really helps my thinking about things. I guess I'm now getting to the point in my IDL coding that the issues of more importance are not how something is done in IDL, but rather what are the best patterns and solutions to use when working with IDL, especially in the land of objects. I'm quite comfortable with objects in other languages (Java/C++), but I'm still trying to wrap my head around the IDL way. As such, some of the familiar ways of attacking problems in Java/C++ don't map too well to IDL and vice versa. One of these days I'll have to write my book on how to learn IDL if you've programmed in a real* programming language before. ;-)

* FORTRAN doesn't count :-)

-Mike

Subject: Re: Yet another object graphics question
Posted by Paul Van Delst[1] on Fri, 25 Feb 2005 22:16:21 GMT
View Forum Message <> Reply to Message

Michael Wallace wrote:
> Hey guys, thanks for all the responses. It really helps my thinking

> about things. I guess I'm now getting to the point in my IDL coding
> that the issues of more importance are not how something is done in IDL,
> but rather what are the best patterns and solutions to use when working
> with IDL, especially in the land of objects. I'm quite comfortable with
> objects in other languages (Java/C++), but I'm still trying to wrap my
> head around the IDL way. As such, some of the familiar ways of
> attacking problems in Java/C++ don't map too well to IDL and vice versa.
> One of these days I'll have to write my book on how to learn IDL if
> you've programmed in a real* programming language before. ;-)
>
> * FORTRAN doesn't count :-)

Wha...?!?!? The cheek! <insert arm flailing and much huffing and puffing>

FORTRAN (i.e. the 1977 and earlier standard) maybe. But Fortran95 can teach you a lot. In particular, that pointers are mostly redundant in a well designed language. :o) Linked lists, queues, trees etc notwithstanding of course (oh, and I guess they're good for array aliasing too. harumph.) And, with the approval of the Fortran2003 standard, the language has the usual complement of OOP stuff (for folks that like that sort of thing). Apart from the polymorphism stuff in f2003, I hang out for the PROTECTED attribute (as opposed to just the current PUBLIC/PRIVATE ones), allocatable components of derived types (aka structures. Now we have to use stoopid pointers in structures to mimic it), and, *FINALLY*, stream I/O. Soon, gone will be the days when we'll need to use the /F77_UNFORMATTED keyword in OPEN statements in IDL.

Woohoo!

I would suggest that Java/C++ are not good languages for learning how to design software. That should be language independent. (That's my dig. :o)

cheers,

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC

Subject: Re: Yet another object graphics question
Posted by Michael Wallace on Sat, 26 Feb 2005 00:01:38 GMT
View Forum Message <> Reply to Message

>> * FORTRAN doesn't count :-)
>
>
> Wha...?!?!? The cheek! <insert arm flailing and much huffing and puffing>

Okay, I have to admit that I purposely inserted the FORTRAN comment just to see what kind of reaction I could get out of you folks. ;-) Just some good natured kidding -- that's all. :-)

While I do agree with you that FORTRAN has come a long way, many of the scientists who I work with still use F77. Scientists on the cutting edge use F90. The newer crop of scientists don't even touch FORTRAN, they go straight to IDL.

> I would suggest that Java/C++ are not good languages for learning how to
> design software. That should be language independent. (That's my dig. :o)

I believe Java, C and sometimes C++ are the best languages to use when learning how to design software. However, software design should be language neutral. You need to make a distinction between the process of designing software and the implementation of the software. If you're learning software design, you need to learn about stacks, queues, trees, graphs, pointers, objects, memory access and everything else that falls into the conceptual realm. It's possible to talk about and understand these ideas at length without ever having written a line of code. This is great theory, but to be useful and to really understand it, you have to put it in practice. I don't know about everyone else, but my understanding of a concept only really crystallizes once I see it in action and have it reinforce all the theory.

For putting concepts into theory, you have to pick a language to do it. There is no universal language that's equally good at everything. Therefore, you have to pick a couple languages to use as the "teaching languages" and only divert from those in special cases. When trying to learn concepts, the last thing you want to do is introduce a new language with new syntax and new style. You should be focused on putting the theory into practice rather than learning new syntax. There will be time to learn syntax later -- now is for learning about compilers or whatnot. That said, you'd want to use languages that'd be generally good for most problems and understanding most concepts.

For procedural languages C fits the bill. It's general purpose and there's a lot of good books and information about the language. Also, C is very good for understanding pointers, memory access, operating systems and networking simply because it doesn't hide all the details that other languages do. I'd be willing to bet that the person who has written networking code in C has a much better understanding of the core networking concepts than someone who's written equivalent code in another language such as Java.

For object-oriented programming, Java is currently the best one out there. There's many good resources and since it does hide some of the details of pointers and the like, it allows you to focus in on a

specific algorithm or data structure without being bogged down by handling all the memory access yourself. C++ was once used as the standard language for teaching OO, but Java is clearly a better alternative, not only because you don't have to worry about getting your pointer arithmetic wrong, but more importantly C++ lets you break OO if you want to.

With my IDL programming, I know conceptually what I want to do, but now I need to be concerned with the idiosyncrasies of the language itself. IDL objects are not the same thing as Java objects. IDL's objects are more like glorified structures that anything else. There are common ways that I've learned to attack problems in Java that make for a mostly elegant approach while also being pretty quick and make good use of memory. When I first jumped into IDL, I tried using the same approach with IDL and found my IDL chewing up every bit of available memory and grinding to a halt. I've come to learn that there are things which IDL does better and things which IDL does worse. Even though you have a great theory, sometimes that theory doesn't work out very well because of constraints of the language itself. You need to adapt the concepts somewhat in order for you to get the elegant, efficient program on the other end. That's what I'm trying to learn how to do right now and that's what I meant by saying that I needed to write my book on how to write IDL if you've programmed in a real language before.

Anyway, happy trails with whatever language you use, even if it is FORTRAN. ;-)

-Mike
