Subject: Re: Passing info and destroying widgets...
Posted by steinhh on Tue, 22 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <7kn0h5$s0c$1@news.doit.wisc.edu> "Liam Gumley"
<Liam.Gumley@ssec.wisc.edu> writes:

> David Fanning <davidf@dfanning.com> wrote in message
> news:MPG.11d85d6ea2cc37d9897de@news.frii.com...
[....]
>> ... I mean, you can write
>> an object method that returns a data pointer too, but
>> by doing so you violate every tenet of good object programming
>> practice, in which the data should be encapsulated and
>> unseen by the outside world.
[....]

So what good is it, then, if it cannot be seen :-) See below.

[Liam Gumley:]
> My personal programming philosophy is to only use pointers
> where absolutely necessary (e.g. for retaining information when
> a widget dies, or for storing structure elements which are of
> unknown size and type until runtime). I've never used HEAP_GC
> in any of my programs; I rely on simple wrappers like those I
> posted at http://cimss.ssec.wisc.edu/~gumley/pointers.html to
> prevent me from getting into memory leakage problems.

>> As for me, I'm sticking to widget programs that clean
>> themselves up and don't leave the user holding the bag,
>> er, pointer. :-)
> I agree that widget *users* (e.g. of David's fine XCOLORS
> routine) don't want to know (and shouldn't be allowed to touch)
> the internals of "shrink-wrapped" widget routines. But I think
> any IDL application developer will eventually run across cases
> where items like information structures need to be shared
> between widgets which are performing closely related
> application-specific tasks.

I must say I agree a lot more with Liam than David here.

Although users in general should not be bothered with pointers to
object data, there are (at least) 3 different types of users with
different types of needs: users, damned users, and scientists.

Hey, some of them can even write IDL programs :-)

What do you do if your objects contain *large* arrays that need

to be manipulated, analyzed or plotted - in a fashion that's completely impossible to foresee when you write the object methods in the first place.

There are basically two ways to do it "David's way", without slipping a pointer out to the user:

1. "Just write an object method to do it".
2. Use objects like handles

No. 1 is the "right thing", but requiring a user to write an object method in order to plot his data in a new and exciting way seems a tad, well, optimistic. We're supposed to *hide* the internals of the object from the user, not let him loose on the inside of the object, for heavens sake.

No. 2 is, I suspect, David's favourite way, but it is *either* memory inefficient *or* includes some awkward handle-like syntax plus "security risks", just like giving out a pointer to the user does (in some respects larger, in other respects smaller).

The memory inefficiency is of course related to the fact that if you're not using pointers, you need to provide a copy of the data to the outside whenever the outside (user) needs to look at it, like this:

```
IDL> data = obj->getdata()
IDL> print,sigma(data)
```

And let's say you write a widget program to display the data from two such objects alongside each other - it would have to make a copy of each of the data sets every time part of the data needs to be redisplayed.. And so on.

Or, you could use something like NO_COPY to avoid this but once again it comes with an extra "security risk":

```
pro my_tiny_analysis_program,obj
   data = obj->getdata(/no_copy)
   print,sigma(data)
end
```

Bye bye data!

Actually, I'd say that providing a *pointer* is a lot less dangerous, since you're more aware of the fact that you're messing with something that points to the data itself, it's not just a normal, dynamic variable.. The example above would also be

totally benign if "data" was returned as a pointer.

The one point where pretending an object is a handle would give you some benefits, is to protect the user from e.g. changing the dimensions/type etc of object data without the object detecting it (just check for it in the "obj->setdata,data" call).

But how do know it's time to remind him you want your data back..

And why should we go back to handle notation? Do you *really* want to write this simple action with three lines of code:

```
IDL> data = obj->getdata(/no_copy)
IDL> print,sigma(data)
IDL> obj->setdata,temporary(data)
```

instead of just:

```
IDL> print,sigma(*obj->getdata())
```

So, I hope that not everyone listens too carefully to David's advice - maybe I could even persuade David to at least mention these issues in a future book...?

Regards,

Stein Vidar
