Subject: Re: Discussion on global variables in IDL
Posted by John-David T. Smith on Fri, 10 Aug 2001 15:34:59 GMT
View Forum Message <> Reply to Message

Altyntsev Dmitriy wrote:
>
> Martin Schultz <martin.schultz@dkrz.de> wrote in message
> news:<ylw66c8q6zy.fsf@faxaelven.dkrz.de>...
>> alt@iszf.irk.ru (Altyntsev Dmitriy) writes:
>>
>>> Hello,
>>>
>>> I'd like to discuss global variable management in IDL. I've read a
>>> pair of NG threads on that theme, and would like to add some
>>> consideration and share experience. Some time ago I've sent a letter
>>> to RSI with some proposals, but since then two version have been
>>> released and I don't see any movement in the direction of improving
>>> global vars management. It seems that everyone is satisfied. Is it
>>> really so?
>>
>> Well, I don't think the situation is as bad as you paint it.
>> First of all I recommend objects
>> Cheers,
>>
>> Martin
>
> My first reaction on yours answers was that I had been misunderstood.
> I even post a message about it, but GOOGLE did not put it for some
> reason in this thread. Then I started to analyze what I need and what
> OOP offers me. I have read a pair of OOP books, scanned one more time
> this NG as far as possible, looked up Martin's IDL_Container and
> others. And to my great surprise discover that I have reinvented the
> OOP wheel. It is funny (or sadly?) that I am using now procedure
> naming conventions in the style of OOP like OBJ_Method. And I am using
> pointer to data of my object in a style I have described before. I was
> completely not right to name this problem as GLOBAL variable problem.
> Global is global. But it is a problem how to share some data between
> several procedures. And it actually is a problem of OOP implementing
> in IDL. And having looked up through this NG I have found that this is
> a problem in IDL and many people are NOT satisfied of current IDL OOP
> state. Some of them have written their own 'containers', others do not
> use OOP, and some even threatened with stopping to use IDL.
>
> The most interesting NG thread (IMHO) was "Top 10 wishes". (BTW, I
> could not find the end and the result of this epic work.) I wish this
> column were persistent and wish to describe my current wish to IDL
> OOP.
>

> So, what is the most annoying thing with IDL OOP for me? The fact that
> when I describe class I loose freedom of changing field type and size.
> I can not have undefined field in class too. And I can not add field
> during execution. Let us view very simple typical example.
>
> IND = WHERE(A GT 0) ; the essence of IDL style
>
> Suppose we want this IND as field of some class. Then I need (as David
> tired to repeat) to use pointers and write like this:
>
> function class1::INIT
> self.ind = ptr_new(/all)
> return, 1
> end
> pro class1::CLEANUP
> ptr_free, self.ind
> end
> pro class1::method1
> ...
> *self.ind = where(A GT 0)
> end
> pro p
> struct = { class1, ..., IND:ptr_new(), ... }
> Obj1 = obj_new('class1')
> Obj1->method1
> obj_destroy, Obj1
> end
>
> So, instead of one simple string I must initialize and destroy pointer
> manually. If during development I suddenly wanted to have IND as field
> of the class, I must add it in class definition, in class init, and in
> class cleanup. Or use some slow container, because all of them are
> written in IDL with a lot of operations.
>
> I do not understand why IDL can not do all this stuff itself. It can
> be done several ways. As for me, I would like to form class fields
> during execution. It seems to me on the whole of IDL style.
>
> pro class1::method1
> ...
> self.IND = where(A GT 0)
> ...
> self.IND = 'We can change type of the field'
> ...
> tmp = temporary(self.IND) ; or destroy it
> end
> pro p
> struct = { class1, public ..., IND, ..., privite ... }

> Obj1 = obj_new('class1')
> Obj1->method1
> obj_destroy, Obj1
> end
>
> As soon as IDL meet self.IND it inits it at once. And destroy by
> obj_destroy or reassigning like local variables.

What you want, I think, is a hash data structure, aka an association list. There is something unhealthy about the work it requires to add a simple variable to the class structure, as you mentioned, but it does have the advantage of forcing you to think through carefully what you're adding. And using structures as the underlying class scaffolding does offer one very important advantage: a fixed and specified set of fields of known size, which lends itself very well to both in-memory and on-disk representation, if requiring occasional awkward pointer manipulation. A free-form class with no more structure imposed on it than the standard IDL variable stack would be very easy to program with, but utterly hopeless when it comes to persistence and forward compatibility.

So yes, I feel your pain. My special difference of opinion with IDL OOP is the complete lack of publicly available instance data members, forcing one through awkward and slow GetProperty calls at every turn (one can verify the inherent slowness by comparing structure lookup speed to method invocation/return speed). I even found code broken by my desire to avoid this penalty: I was caching in one object another object's data members, which later came to be more dynamic. Ouch. My secondary complaint is the lack of class variables: variables accessible to every instance of a class, for inter-instance communication. Very useful. However, given that OOP was grafted on a 20 year old language, they actually did a decent job of sticking to the simple, tried and true OOP features that could be reliably implemented.

> And of course, I must have an opportunity to assign class method as
> event handler in widget applications.

I've been calling for a rewrite of XManager and the core widget_event and event attachment code for a while, to allow such a flexible approach. In the meantime, we pollute the objects we write with glue procedures like:

```
pro myclass_event, ev
 widget_control, self.top, get_uvalue=self
  self->Event, ev
end
```

etc.

> What would you say?
>
> The other my "Top 10" very shortly
> 1) Keystroke events and MDI interface (ENVI and others apps might be
> much more convenient). I use widget_text trick now.

Yes, but years later, my hack is still going strong. I even have a version that traps arrow keys (though less reliably). Maybe I've accidentally derailed the potential for getting *real* key events.

> 2) Handy and effective tools for viewing images and plots (colors,
> multichannel, cursor position, zoom, image enhancement, export,
> printing, ...). I have my own tools. They are far from perfection.

There are many excellent viewer tools available from a variety of sources external to RSI.

> 3) Class explorer, so I could quickly look up my fields and methods.
> Now I use external text editor search capabilities.

If you are a emacs user, try out IDLWAVE, with pop-up method locators, and field completion (among many other powerful features).

Anyway, good luck learning to live with IDL, warts and all. It's a slow road.

JD
