Subject: Discussion on global variables in IDL
Posted by alt on Wed, 01 Aug 2001 05:34:55 GMT
View Forum Message <> Reply to Message

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?
First of all, I like IDL. And the most remarkable IDL feature, IMHO, is an opportunity to develop program at the same time you think. You have an idea, you type pair of strings, run it, look how it works, debug it, correct it and so on. If you need variable, you define it at once. I don't need declare it somewhere, define its type, etc. All that you have to point is variable name. Its type, structure can be changed during run flow. It can become even undefined. It's okay. I like it. Sometimes it leads to errors, that couldn't happen in C for example, but everyone choose his own way.
And it's okay while you are dealing with a pair of procedures without widgets. But when you create big widget project you start to have problems with global data. The only way that IDL offers is COMMON block. One can argue that's good enough. May be. Tastes differ. But one thing may be said exactly. COMMON block doesn't allow having many instances of same widget. First time I ran into it I was surprised that I have to put my global data in structure, then put it into some button user value, and get it from there on event. I disliked this method strongly, so I've written several very simple procedures that help me to manage global data. I have been using last version of them for about a year and a half and I'm satisfied. Not always. Sometimes I use COMMONs. It depends on situations, but at least I have a choice. May be this ideas will be useful for someone else.

So, everything is based on pointers. We have main global data pointer p that points on all global data heap.
p = ptr_new( { parr:ptrarr(Q), namearr:strarr(Q) })
Parr points on global variables, namearr contains global var names. This p I pass as parameter to every procedure where I need global data.

So, how it looks for the user. Very simple in my opinion.

ini, p ; global data initialization
sav, p, var1, 'var1_global_name' ; save var1 to global heap as var1_global_name

var, p, var1, 'var1_global_name' ;  restore var1_global_name from
global heap as var1
del, p ; clean up heap

You can save and restore several variables at once.
sav, p, var1, 'var1', var2, 'var2', var3, 'var3'
var, p, var1, 'var1', var2, 'var2'

You can restore var as pointer to avoid copy huge arrays copying.
var, p, var1, '*var1'
(*var1)[100,200] = 20
The vars to be saved can be undefined.

So the widget application looks like that.

```
pro main
 ini, p
 sav, p, var1, 'var1'
 ....
 baseID = widget_base( uvalue = p )
 ....
 sav, p, var2, 'var2'
 ....
 xmanager, 'main', baseID, cleanup = 'main_clean'
end
pro main_event, ev
 widget_control, ev.top, get_uvalue = p
 ....
 proc1, p, par1, par2
 proc2, p, par3, par4
end
pro proc1, p, par1, par2
 ...
 var, p, var1, 'var1'
 ...
 sav, p, var1, 'var1'
end
pro proc2, p0, par1, par2
 ini, p ; initialization of new global data heap for another widget,
for example
 ....
end
pro main_cleanup, id
 ....
 widget_control, id, get_uvalue = p
 del, p
 ....
end
```

This technique main advantages are:
1) We create, save and restore global variables just where we need them
2) We can have multiple instances of the same widget
Disadvantages:
1) We can't restore a group of variables at once as in case COMMON block.
2) It is more time consuming. So it is better not to use this method in time critical parts.

Implementing this technique in IDL kernel can solve these disadvantages. So we could get FLOATING COMMON block with pointer on it.

If someone is interested in those programs you can e-mail me or I can put them as is on our server.

Thank you for your attention.

Altyntsev Dmitriy
Remote Sensing Center of ISTP, Irkutsk
alt@iszf.irk.ru

---

Subject: Re: Discussion on global variables in IDL
Posted by Pavel A. Romashkin on Wed, 08 Aug 2001 16:54:49 GMT
View Forum Message <> Reply to Message

I agree, this is how an IDL program starts. But I got to the point where once I get a slightest suspicion that I may ever want to do again what I am typing on the command line, I write a reusable code for it. It paid off more times than it didn't.
Pavel

Martin Schultz wrote:
>
> And this is exactly their weakness. I would guess that most IDL applications
> are born out of ad-hoc code once written to analyse a specific situation,
> and then generalized. To get rid of common blocks in this provess is much
> harder than to avoid them in the first place.

---

Subject: Re: Discussion on global variables in IDL
Posted by david[2] on Fri, 10 Aug 2001 11:43:55 GMT
View Forum Message <> Reply to Message

Altyntsev Dmitriy writes:

> I do not understand why IDL can not do all this stuff itself. It can
> be done several ways.

My all-time favorite quote about IDL was made
on this newsgroup. Someone in a similar thread
complained that "IDL looks like it had been
thrown together by a bunch of different programmers
over a long period of time."

Well, ... exactly.

The remarkable thing about IDL, it seems to me, is
not that it works in such a quirky way after almost
20 years of existence, but that it works at all.
What would your car look like, do you think, if all
the modern improvements had to be added to the original
20-year old chassis and design? Probably NOT like the
new 2001 model.

IDL is no different. It is a very BIG program.
With a LONG history. One can certainly argue
(as many in this newsgroup have, including me)
that IDL's implementation of objects is not
perfect. But the fact that they are available
at all--in any form--is a remarkable programming
achievement given the constraints the RSI
programming team had to work under.

Would IDL be better off, more self-consistent,
a sleeker, more high-powered language if it
was written over from scratch? You better believe
it! And I'm sure every one of the IDL programmers
would be grateful for the opportunity to build
that wonderful new program.

But it can't be easy, or some young hot-shot
in some garage somewhere would already have
done it.

I don't mean to discourage people like Dmitriy
from questioning what IDL is and where it is
going. Lord knows there are many things that
CAN be done that (for whatever reason) are not.
But you can't make a silk purse out of a sow's
ear. You can't turn several million lines of
C code into the latest and greatest C++ program

in a one-year maintenance cycle. Having experienced
first hand how frustrating it is to maintain a
few measly thousand lines of code, I am enormously
sympathetic to anyone carrying out the thankless
task of maintaining--and extending--IDL functionality.

IDL is far from perfect, true. But, like Paul van Delst,
I would find it hard to live without it.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

---

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

Subject: Re: Discussion on global variables in IDL
Posted by mvukovic on Fri, 10 Aug 2001 15:47:20 GMT
View Forum Message <> Reply to Message

alt@iszf.irk.ru (Altyntsev Dmitriy) wrote in message
news:<6b9fda50.0108100132.13a58695@posting.google.com>...

>
> 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.

...stuff deleted
> Regards,
> Altyntsev Dmitriy

OOP is for ``high level'' programming. In my (not that vast)
experience you don't write an OOP application by sitting in front of
the terminal. You think, specify, and _plan_ the application. Then
you type it in.

BTW, I ``discovered'' a gorgeous book on OOP called Design Patterns.
I don't have the author names, but you can find it easily enough on
amazon.com. It was an eye-opener for me on making re-usable object
patterns.

If I had a top ten wish, it would be to make IDL a more OO language,
implementing all of the basic OO features and some of the more
advanced OO features, including operator overloading. Note that this
is a dangereous wish, as highly obtuse and non-transparent code can be
result from a mis-use of those tools.

But other wishes of your list are appealing too.

Cheers,

Mirko

Subject: Re: Discussion on global variables in IDL
Posted by Pavel A. Romashkin on Fri, 10 Aug 2001 17:04:14 GMT
View Forum Message <> Reply to Message

Wishful thinking has its positive sides, but I fully agree with David - changing IDL internals to act the way users may wish is very likely far beyond real capability of RSI.
In their current state, objects in IDL are *very* useful, fairly easy to write (once planned well) and allow to get your job done faster and cleaner. And it is not too much pain at all to find simple and short ways to write reuseable object code.
When you are fixing your car, you may want that wrench to be 2" longer, but it isn't. So you take the one you've got and do your work.
Cheers,
Pavel

---

Subject: Re: Discussion on global variables in IDL
Posted by alt on Mon, 13 Aug 2001 04:29:39 GMT
View Forum Message <> Reply to Message

david@dfanning.com (David Fanning) wrote in message news:<MPG.15dd76e818f14349989e60@news.frii.com>...
> Altyntsev Dmitriy writes:
>
>> I do not understand why IDL can not do all this stuff itself. It can
>> be done several ways.
> IDL is far from perfect, true. But, like Paul van Delst,
> I would find it hard to live without it.
>
> Cheers,
>
> David

I felt to be thrown off balance in some way about IDL, but now I feel I've got into the 'stream'. I seem to have understood the situation and the only thing that one might desire is some feedback from RSI like "Okay, guys. We are looking after you. We know what do you want. We do a hard work and we will make it better without a doubt. IDL forever, etc" Wishful thinking again ;-) Meanwhile I thank you for your encouraging (I would even say psychotherapeutic) work, I'm taking my wrench and continue doing my work on a slow road.

Regards,
Dmitriy

---

Subject: Re: Discussion on global variables in IDL
Posted by david[2] on Mon, 13 Aug 2001 13:27:26 GMT
View Forum Message <> Reply to Message

Altyntsev Dmitriy writes:

> I felt to be thrown off balance in some way about IDL, but now I feel
> I've got into the 'stream'. I seem to have understood the situation
> and the only thing that one might desire is some feedback from RSI
> like "Okay, guys. We are looking after you. We know what do you want.
> We do a hard work and we will make it better without a doubt. IDL
> forever, etc" Wishful thinking again ;-) Meanwhile I thank you for
> your encouraging (I would even say psychotherapeutic) work, I'm taking
> my wrench and continue doing my work on a slow road.

I'm not sure if it's the result of Perestroika
or the Internet, but the world has certainly
become a smaller place. Who would have thought
the residents of the former Soviet Union would
turn out to have such a good sense of humor!

Cheers,

David

P.S. Let's just say that the names "Pavel and
Dmitriy" have replaced "incomprehensible discussions
on Histogram functionality are infrequent" on my
Ten Best Things About the IDL Newsgroup list.

--
David Fanning, Ph.D.
Fanning Software Psychotherapy

---

Subject: Re: Discussion on global variables in IDL
Posted by btt on Mon, 13 Aug 2001 13:41:37 GMT
View Forum Message <> Reply to Message

JD Smith wrote:

>
> 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.

Oh geez,

Just when I think I'm finally beginning to understand objects, JD throws another curveball. 'Publicly available instance data'. I'm wondering what that is, and would like to know what you mean; I hope you can field these questions for me.

All I can imagine is something like the following...

myObj = OBJ_NEW('OBJECT_CLASS', data)

myObj.Name = 'Bob' (rather than myObj->SetProperty, Name = 'Bob')

Am I getting that straight? No method is needed to fiddle with an object's data?

> My
> secondary complaint is the lack of class variables: variables accessible
> to every instance of a class, for inter-instance communication.

Does this mean that the variable is defined in the OBJECT_CLASS__DEFINE procedure so it gets intialized with the right value? Instead of my data field NAME being assigned the value '' at initialization, it gets assigned the value 'BOB' always?

Thanks,

Ben

--
Ben Tupper
Bigelow Laboratory for Ocean Sciences
180 McKown Point Rd.
W. Boothbay Harbor, ME 04575
btupper@bigelow.org

---

Subject: Re: Discussion on global variables in IDL
Posted by John-David T. Smith on Mon, 13 Aug 2001 14:27:56 GMT
View Forum Message <> Reply to Message

Ben Tupper wrote:
>
> JD Smith wrote:
>
>>

>> 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.
>
> Oh geez,
>
> Just when I think I'm finally beginning to understand objects, JD throws
> another curveball. 'Publicly available instance data'. I'm wondering
> what that is, and would like to know what you mean; I hope you can field
> these questions for me.
>
> All I can imagine is something like the following...
>
> myObj = OBJ_NEW('OBJECT_CLASS', data)
>
> myObj.Name = 'Bob' (rather than myObj->SetProperty, Name = 'Bob')
>
> Am I getting that straight? No method is needed to fiddle with an
> object's data?

Precisely. Different languages have differing approaches to data encapsulation. Some let you articulate the divide between public and private data. Some leave all data public and leave it up to the programmer to behave. And some, like IDL, lock everything away like a distrusting father with his truck keys.

>
>> My
>> secondary complaint is the lack of class variables: variables accessible
>> to every instance of a class, for inter-instance communication.
>
> Does this mean that the variable is defined in the OBJECT_CLASS__DEFINE
> procedure so it gets intialized with the right value? Instead of my
> data field NAME being assigned the value '' at initialization, it gets
> assigned the value 'BOB' always?

No, it means variables can be defined within the class itself, not within a given instance of the class (aka an object). You can then have "global" variables within one class. So, for all objects a b c d of the same class, a.class_var, b.class_var, c.class_var, and d.class_var are the same actual variable, and can be used to communicate among them.

Why would you want to do that? Take a look at a post from a few years ago on a Singleton method. There are lots of other good uses you can

imagine, also.

But anyway, we should all be glad IDL's higher level programming tools are somewhat crude; otherwise, it might be adopted by an open source develpment project and a raft of 13 year old kernel programmers would flood our happy little newsgroup with increasingly irrelevant code snippets in a pseudo-machismo recognized only by their kind. They'd probably even start up untold "incomprehensible discussions on Histogram functionality." Right, David?

JD

---

Subject: Re: Discussion on global variables in IDL
Posted by Pavel A. Romashkin on Mon, 13 Aug 2001 16:00:55 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Who would have thought
> the residents of the former Soviet Union would
> turn out to have such a good sense of humor

You just never had a chance to find out before, because the US was hiding behind the Iron Curtain :-)
Cheers,
Pavel

---