Subject: Common or not common
Posted by morisset on Fri, 14 Nov 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Helo,

I read already lot of times David (and others) speaking
about "never use common" (except to save values used in multiple
call of the _same_ routine). I cannot understand how I could
develop my codes without any common. I have a lot of routines
that have to deal with a lot of variables (of various type).

I can't imagine passing 25 or more parameters to the routines.
And with the previous version of IDL, the parameters where
limited in number.

And what happens when I decide to add one other parameter to
one routine? I have to check all the call to the routine
to be sure that the new parameter is passed
(I can't wait for IDL to do it itself ;-) ?

I use parameters when I want to deal with procedure or
function I want to apply to different variables. But when
in every routines/functions of the program
the spectrum, lambdas, corrections of
different types and a lot of other things,
have always the same name, it's easier and safer
to pass them via a common, no?

Can somebody explain me what I miss?

C. Morisset, IAG/USP, Brasil
http://www.iagusp.usp.br/~morisset/idl

-------------------==== Posted via Deja News ====-----------------------
http://www.dejanews.com/   Search, Read, Post to Usenet

Subject: Re: Common or not common
Posted by davidf on Fri, 14 Nov 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Liam Gumley (liam.gumley@ssec.wisc.edu) answered
Mr. Morisset's original question better than I could have
myself. :-)

But let me make just a few other points.

Mr. Morisset writes:

> I can't imagine passing 25 or more parameters to the routines.

Nor can I. Although event handlers often need at least this
much information to function properly. You can pass the
information by common blocks, which has many, many limitations
or you can pass the information via "pointers". I prefer the
latter.

> And what happens when I decide to add one other parameter to
> one routine?

More to the point, what happens when *you* and your common
blocks have to add another parameter. I don't have to exit
IDL, that is for sure. :-) Nor do I have to update all of
my program modules. I just add the parameter to my structure.

> I use parameters when I want to deal with procedure or
> function I want to apply to different variables. But when
> in every routines/functions of the program
> the spectrum, lambdas, corrections of
> different types and a lot of other things,
> have always the same name, it's easier and safer
> to pass them via a common, no?

No. I think it is a great idea to always name things the
same. I wouldn't do it any other way. User values are
completely safe because each event is processed individually.
If you take the info structure out with a No_Copy at the top
of your event handler, and put it back with a No_Copy at the
bottom (you *do* use No_Copy don't you), then you NEVER get
in trouble. Ain't possible.

> Can somebody explain me what I miss?

You miss the ability to have more than one version of your
program running at any one time. How good is that great image
processing routine if you can only process one image at a time?

Liam writes:

> PS: David I'll take that case of beer whenever you're ready.

If he buys my book, I'll mail you the case of beer. :-)

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Common or not common
Posted by Liam Gumley on Fri, 14 Nov 1997 08:00:00 GMT

View Forum Message <> Reply to Message

morisset@iagusp.usp.br wrote:

> I read already lot of times David (and others) speaking
> about "never use common" (except to save values used in multiple
> call of the _same_ routine). I cannot understand how I could
> develop my codes without any common. I have a lot of routines
> that have to deal with a lot of variables (of various type).
>
> I can't imagine passing 25 or more parameters to the routines.
> And with the previous version of IDL, the parameters where
> limited in number.
>
> And what happens when I decide to add one other parameter to
> one routine? I have to check all the call to the routine
> to be sure that the new parameter is passed
> (I can't wait for IDL to do it itself ;-) ?
>
> I use parameters when I want to deal with procedure or
> function I want to apply to different variables. But when
> in every routines/functions of the program
> the spectrum, lambdas, corrections of
> different types and a lot of other things,
> have always the same name, it's easier and safer
> to pass them via a common, no?

Two words:
Structures,
Pointers.

Structures can contain anything that you would normally put in a common
block. In the simplest case, you maintain one global structure that contains
all the information you need (see any of David's example widget programs).
You define it once it your main program, or in an included file.

A pointer (or handle if you're still using IDL 4.0) to then passed to any
procedures or functions that need items from the structure (the pointer
points to the structure). Then whenever you make changes to the structure
(e.g. adding a new variable), the changes are automatically available to any
routines that use the structure. If you want to change the size or type of
an item in the structure, then instead of putting the item itself in the
structure, put a *pointer* to the item in the structure.

Trust me, it works. David wouldn't keep saying it if it didn't! Please look
at any of David's example widget programs: you will definitely learn
something. I recall reading quite a while ago (maybe it was in one of
David's manuals) that the first step to becoming a proficient in IDL is
learning how to *think* like an IDL programmer, not a FORTRAN programmer.

Cheers,
Liam.
PS: David I'll take that case of beer whenever you're ready.

Subject: Re: Common or not common
Posted by morisset on Mon, 17 Nov 1997 08:00:00 GMT

View Forum Message <> Reply to Message

From meinel@aero.org who can't post:

Could one of you forward this to comp.lang.idl-pvwave if you think
it is worth it. For some reason it refused to post.

davidf@dfanning.com (David Fanning) writes:
>
> Mr. Morisset writes:
>
>> I can't imagine passing 25 or more parameters to the routines.
>
> Nor can I. Although event handlers often need at least this
> much information to function properly. You can pass the
> information by common blocks, which has many, many limitations
> or you can pass the information via "pointers". I prefer the
> latter.
>

Talk to C programmers. Pointers are a two-edged sword. You can do
pretty neat stuff with pointers, but they are also the source of
about half of the bugs.

>
>> And what happens when I decide to add one other parameter to
>> one routine?

>
> More to the point, what happens when *you* and your common
> blocks have to add another parameter. I don't have to exit
> IDL, that is for sure. :-) Nor do I have to update all of
> my program modules. I just add the parameter to my structure.
>

I currently use commons and named structures. There are times
when I don't know the size of some arrays until well after the
structure is created. However, I just read in the manual (TFM)
that anonymous structures _can_ be modified after creation.
I'll have to give that a try...

Which brings up an IDL gripe -- why do named structures have
different properties than anonymous structures? And why aren't
structure properties consistent between types? According to TFM:

"Once defined, a given named structure type cannot be changed."

Yet in the given example, NAME is defined as a zero-length string,
but subsequent assignments can have any number of characters. Are
strings the only variables that can be changed in a named structure?
Why are strings handled differently? Should I make everything in my
named structure a string and then convert to some other type at the
appropriate time? Inquiring minds want to know.

>> Can somebody explain me what I miss?
>
> You miss the ability to have more than one version of your
> program running at any one time. How good is that great image
> processing routine if you can only process one image at a time?
>

That's odd. I have a "great image processing routine" that uses
commons and I can process more than one image at a time. Maybe
I'm just "smarter than your average bear" (for those of you who
didn't watch Hanna-Barberra cartoons, that was a quote from Yogi
Bear).

Ed
meinel@aero.org

Subject: Re: Common or not common
Posted by mgs on Wed, 19 Nov 1997 08:00:00 GMT
View Forum Message <> Reply to Message

In article <879774434.18173@dejanews.com>, meinel@aero.org wrote:

> davidf@dfanning.com (David Fanning) writes:
>>
>> Mr. Morisset writes:
>>
>>> I can't imagine passing 25 or more parameters to the routines.
>>
>> Nor can I. Although event handlers often need at least this
>> much information to function properly. You can pass the
>> information by common blocks, which has many, many limitations
>> or you can pass the information via "pointers". I prefer the
>> latter.
>>
>
> Talk to C programmers. Pointers are a two-edged sword. You can do
> pretty neat stuff with pointers, but they are also the source of
> about half of the bugs.

No doubt about it. However the amount of code they save reduce the number
of possible bugs by at least half. It's a matter of debugging a few lines
of not terribly intuitive code vs. debugging dozens of lines of only
slightly more understandable code.

...
> I currently use commons and named structures. There are times
> when I don't know the size of some arrays until well after the
> structure is created. However, I just read in the manual (TFM)
> that anonymous structures _can_ be modified after creation.
> I'll have to give that a try...

For either structure you can use a pointer to your data.

> Which brings up an IDL gripe -- why do named structures have
> different properties than anonymous structures?

They each have their advantages. I could usually care less about the names
of my structures, yet I depend on the named event structures to determine
the type of events being handled by the user interface. Here's a portion
of my user interface event handlers:

```
PRO IasGps_Event, event
   sEventType = Tag_Names(event, /Structure_Name)
   CASE sEventType OF
      'WIDGET_KILL_REQUEST': BEGIN
         ;   code
```

```
          END
       'WIDGET_TRACKING': BEGIN
          ;   code
          END
       ELSE: BEGIN
          ;   code
          END
   ENDCASE
END
```

The idea I use is that IDL's named event structures will allow different types of events to be processed. Without them, I'd have to parse out elements of the structure to see what was actually intended. By using the ELSE I can pass in my own unnamed structures as events, since I know they'll be picked up in the ELSE section.

> And why aren't
> structure properties consistent between types? According to TFM:
>
> "Once defined, a given named structure type cannot be changed."
>
> Yet in the given example, NAME is defined as a zero-length string,
> but subsequent assignments can have any number of characters. Are
> strings the only variables that can be changed in a named structure?
> Why are strings handled differently? Should I make everything in my
> named structure a string and then convert to some other type at the
> appropriate time? Inquiring minds want to know.

Strings are treated different. Consider it as a pointer to a string, without having to worry about the pointer. Very much like the difference between char and string. Define the structure as it is needed, don't use all strings. That would be highly inefficient, especially for your image data.

>> You miss the ability to have more than one version of your
>> program running at any one time. How good is that great image
>> processing routine if you can only process one image at a time?
>
> That's odd. I have a "great image processing routine" that uses
> commons and I can process more than one image at a time. Maybe
> I'm just "smarter than your average bear" (for those of you who
> didn't watch Hanna-Barberra cartoons, that was a quote from Yogi
> Bear).

Actually, I think you may have misinterpreted David's question. If I start multiple versions of an application from the command line (since the command line can now be active with the XManager changes), they do not interfere with each other. They are two entirely different applications.

This is not possible with commons since each instance of the appication
would be using the same common block. Consider compound widgets for a
moment. If you use something like the CW_BGroup in several areas of your
application, does selecting one button ina button group directly affect
the data of the other button groups? The answer is no, because they are
not using common blocks.

Back to your example, what happens when you apply a smoothing function to
one image and an edge detection to the other? If they are the same data in
the common block you'll get an edge detection of your smoothed data. You
can get around this with pixmaps and copies of the original data, but
you're taking extra steps to avoid something which can be done more
appropriately without commons.

--
Mike Schienle   Interactive Visuals
mgs@sd.cybernex.net   http://ww2.sd.cybernex.net/~mgs/