Subject: global variables and IDLSPEC issues
Posted by J.D. Smith on Wed, 22 Apr 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Martin Schultz wrote:
>
> J.D. Smith wrote:
>>
>> Allow me to elaborate on the situation which would require a more
>> flexible mechanism for importing and exporting main level variables.
> [...]
>
> Thanks! That makes sense indeed.
>
>> And as for the philosophical question of greater power vs. consolidation
>> and organization, I see it as a non-issue. I argue that if the
>> introduction of new features and flexibility makes a program less
>> accessible, they were not correctly implemented. The common backbone of
>> all good programs I've encountered is the hierarchical organization of
>> functionality: a gentle learning curve whose gentleness nonetheless
>> does not impose arbitrary limits on how high the curve goes. I realize
>> this is difficult to implement in the real world, but I don't see this
>> as an excuse. Take as an example the IDL Advanced Development tools for
>> linking with external programs, and even embedding IDL within a custom
>> program. These tools are certainly above the heads of most IDL users
>> (including myself, for the most part), but they are eminently useful and
>> powerful. Most users, however, can be perfectly productive without
>> knowing anything about them.
>
> I certainly agree with you on this. It's just that I seem to know more
> people who struggle with the basics in IDL than with other "plotting"
> software. So there must be a big step before you can gently ride uphill
> on the learning curve. It may be true that one should not temper with
> IDL if one is only interested in producing the occasional line graph,
> there may be other point-and-click programs which are less frustrating,
> but I am convinced that IDL could win many more users if the first steps
> were simpler. If David's book became the standard users' manual and all
> those "but"s were eliminated (the consolidation) that could greatly
> facilitate beginner's access to our favorite software. And although I
> easily admit that I probably know less than 20% of IDL's features, I
> keep wondering why I have to look up all these !X and !Y tags in the
> online help every time I want to produce a plot that looks just a little
> different from others. And sometimes it is really hard to find out about
> "new" features: unless you know the name of the routine you are looking
> for, it can take quite a while before you find it, and if you are not
> sure whether it exists, you may give up early.

I do agree IDL plotting is a mess. I use it for interactive programs,

and quick looks, but for real, publish-quality plots, I use another package altogether. It is disheartening that getting a fully customised plot is so difficult in a package which offers so much graphics power, and I firmly believe this issue could be dealt with. Whether Object Graphics will provide the solution is yet to be seen.

>
>>
>> I believe IDL *should* focus on consolidating and cleaning their
>> interface, but I don't think they should delay or inhibit the
>> introduction of new features to help achieve this consolidation. As we
>> all know, the simplest program is the one which does nothing at all.
>>
>
> That may be a matter of resources, too. But you are certainly right: if
> there already i ssome code to do what you want, and it's just not
> documented and/or accessible, then release of this should certainly not
> be delayed. And as I understand David and others, there may be a couple
> of things to improve in the OOP part which may be of greater importance
> as well.
>
> Regards,
> Martin.
>
> PS: BTW: do you have an idea how much the results of your speed survey
> could be affected by network speed rather than machine speed? True: not
> too many users may sit right at the fancy workstation directly, so the
> results may well reflect "wall clock time" in a real environment. But
> can one judge the machines from this? Somehow I have a hard time
> believing that so many PC's have faster graphics than an SGI
> workstation.
>

The speed survey results for calculational performance should be independent of network vs. non-network access since the I/O test, which could possibly depends on this factor, was removed from the sort key. For graphics results, it is true that several entries acknowledged being run over a network, and I documented all of these. Some entrants might have neglected to tell me they were running over a network. However, there were many workstation results which seemed legitimate and fell well below the Pentium 133 machine. So, to sum up, while I can't say that none of the machines in my survey had better performance than the current top contender, I can say that the Pentium did beat several legitimate high-end entries.

JD

--

J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-4083
206 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: Global variables and IDL
Posted by davidf on Tue, 13 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Walter Roberson (roberson@ibd.nrc.ca) writes:

> Jim Russell wrote:
>
> :I'm certain that IDL has a global variable (I remember it being mentioned in
> :Fanning's book), but don't remember how to invoke it. Maybe someone else can
> :provide that for you.
>
> Use a common block. That requires adding only one extra statement to each
> routine that uses an element of the common, and requires no other code
> changes.

I'm quite sure you didn't find a common block recommendation in
Fanning's book. :-(

What I may have recommended was to create your own *system
variable*, which is the IDL equivalent of a global variable.

For example, if I were building a large application that I
wanted to run in both a Windows and UNIX environment, and that
application has a lot of sub-directories, etc. then I can have
a great deal of trouble with filenames. For example, the
"data" directory might be E:\secret\data on the PC and
/usr/people/bob/secret/data on the UNIX machine. In fact,
every installation may have their own data directory.
Writing code that can actually find the application's
data directory could be a nightmare.

So I might write a "preferences" file for the application
that each person who installs the application has to modify
for their site. One item might be the location of the
"data directory". On the PC, this might look like this:

```
DEFSYSV, '!Data_Directory', 'E:\secret\data'
```

The UNIX user would modify this line to this:

```
DEFSYSV, '!Data_Directory', '/usr/people/bob/secret/data'
```

But now it is easy for me to write code to look in this
directory. I simply construct my filenames like this:

```
  datafile = Filepath(Root_Dir=!Data_Directory, $
    SubDir=['experiment', 'daytwo'], 'exper12a.dat')
```

In the past 7-8 years I've found more use for GOTOs than
I have for common blocks, and I NEVER use a GOTO if I
can avoid 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: Global variables and IDL
Posted by roberson on Tue, 13 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <19990412230259.15932.00002313@ng117.aol.com>,
Russ051990 <russ051990@aol.com> wrote:
:I'm certain that IDL has a global variable (I remember it being mentioned in
:Fanning's book), but don't remember how to invoke it. Maybe someone else can
:provide that for you.

Use a common block. That requires adding only one extra statement to each
routine that uses an element of the common, and requires no other code
changes.

:In the meantime, you could use a "FORTRAN-like" structure to hold your "global"
:variables, and pass just the pointer to the structure around if you'd like.
:The structure can be declared like this
:common = PTR_NEW ({ varname1:value1, varname2:value2, ..., varnamen:valuen})

You would have to carry the pointer around everywhere (the pointer name
will not be global in scope even though the heap values pointed to are
global); and you would have to make many code changes to dereference.
Fortunately those changes could be made in batch mode with any good
editor.

The online help shows, in Widget Example 2, another method.
When you are using a widget and you ask for the value of the 'top' field
of the event structure, you get the user value associated with the top
level widget of the widget you are processing the event for. If that
user value has been set to a widget ID, you can use WIDGET_CONTROL
routines to extract values from that widget, or to set values in that widget.
But if you're going to do that and you don't need the graphical part to
change, you might as well make the user value a PTR. [Not a structure,
though, as it is a copy of the value that is put into the widget unless
you use /NO_COPY.]

In any case, this discussion is inappropriate for comp.lang.idl
(which is about the Interface Definition Language). I have cross-posted
to the appropriate newsgroup. comp.lang.idl-pvwave, and set followups
to go to there.

Subject: Re: Global variables and IDL
Posted by davidf on Wed, 14 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Rose (rmlongfield@my-dejanews.com) writes:

> My problem is that this tool of mine is getting rather complicated (read:
> messy) and I have three WIDGET_CONTROL, GET_UVALUE statements at the
> beginning of many of my event handlers, because I have several info pointers
> which hold information that I need. (One each from the event ID, top and main
> info) Maybe I am just dis-organized, but it seems to be getting out of hand.

Humm. I've been there before too. And to tell you the truth, I've
even thought about COMMON blocks a time or two. (I never told
anyone COMMON blocks are evil. I've just tried to live my
life as if they were. :-)

But in the end I just either reorganized my program (usually with
some kind of object at the heart of it) or I just turned the whole
mess over to Dick, who is far and away the better programmer, and
went back to work on the theory section of my damn book. :-(

> I am wondering whether using a COMMON statement for a set of
> variables or data arrays that aren't changing would not be such a bad idea.
> What about using constants such as PI or earth radius? Where can they be
> defined once and then used throughout a program ? I am considering putting
> my data in a COMMON statement so that I can get to it in an easy way (i.e.
> without (*(*infoPtr).dataArrPtr) type statements all over the place)

I'm sympathetic. I really am. I'm more sympathetic to constants. (PI by the way is available via a system variable.) I'm really loath to put data in common because then you really do lose the ability to use more than one instance of your program at a time. On the other hand, how many versions of a program like ENVI do you really want running?

> It is this or re-writing the tool from scratch (which it really needs if only
> I could find the time).

My daddy used to say, "No time to do it right. Plenty of time to do it over." :-)

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: Global variables and IDL
Posted by steinhh on Wed, 14 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

rmlongfield writes:

> HI All, I've been thinking about this question myself lately.
> As far as I understand, the use of COMMON blocks is not
> recommended when working with widgets, but is ok otherwise.
> I have heard that there may be a problem if there are too
> many variables, but 20 seems ok to me.

Well, I'd say that common blocks should only be used when there is *no* conceivable way you'd *ever* want to have more than one version (instance) of your data. You *may* want to generate a slightly altered data set and view that along side the original data set at some point in time.

As such, only a few situations *really* qualify -- like the common blocks used (internally) by XMANAGER. There's no way you'd ever want to have *two* lists of managed widgets.

Another extremely good example is the list of singleton objects

used by J.D. Smith's singleton abstract class.

A singleton (sub)class is defined by the fact that only one
instance (object) of that class should exist at any one
time. The singleton INIT method needs to know whether or not an
object of the same class already exists. In order to do this,
one needs to have a list of existing singleton objects. There
is no way you'd ever want to have *two* such lists (defeats the
purpose...), so you can safely put it into a common block.

In general, common blocks are good for storing information used
in a (globally available) system that keeps track of things...
The "things" themselves should not be put into common blocks
:-)

Having said this, I must confess that yes, I have used common
blocks for other purposes, but that's only for very small, very
experimental programs.

> I have an image processing tool which uses a base data set
> but many different widget modules, each with its own TOP
> LEVEL WIDGET. I would like to have access to this data from
> whatever or whichever widget I am working in. The data sets
> I read are created in the middle of my processing, so it
> would do no good to read it at the start and have a general
> pointer that can be included in each Top Level Widget.

You should take advantage of the fact that you can create a
pointer without pointing it at anything:

```
storage = ptr_new()
xstartprog,storage=storage ;; Data will be available later...
xutility,storage=storage   ;; Ditto.
xreaddata,storage=storage  ;; This one will read the data...
```

Regards,

Stein Vidar

Subject: Re: Global variables and IDL
Posted by luthi on Wed, 14 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

> Use a common block. That requires adding only one extra statement to each
> routine that uses an element of the common, and requires no other code
> changes.

I just did this recently when writing code with some 30 subroutines calling each other and being called by procedures for optimization (IMSL-routines of PV-Wave). I found no other way to share huge arrays of data between all routines than by using COMMON blocks and thus created some 20 of them. Of course not every block is shared between all routines, but in this way I can select the ones I need.

Further I was concerned with the speed issue and thought, that COMMON blocks would be a good idea instead of actually passing variables (huge arrays) as paramters to routines which are called some 100000 times, which would result in a large overhead of memory assignment and data type checking. (Okay, probably I should have used C or Fortran, but Wave is that convenient...)

Does anybody have an idea wheter COMMON blocks could help speed up a program?
And has anybody an idea wheter COMMON blocks are evil?

> In the past 7-8 years I've found more use for GOTOs than
> I have for common blocks, and I NEVER use a GOTO if I
> can avoid it. :-)

For me it's inverse: I absolutely NEVER use a GOTO (actually I forgot how to use it... the old Basic times are so far! ;-) but I make plenty use of COMMON blocks.

Cheers

Martin
--
==========
Martin Luethi Tel. +41 1 632 40 92
Glaciology Section Fax. +41 1 632 11 92
VAW ETH Zuerich
CH-8092 Zuerich mail luthi@vaw.baum.ethz.ch
Switzerland
==========

Subject: Re: Global variables and IDL
Posted by rmlongfield on Wed, 14 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

HI All, I've been thinking about this question myself lately. As far as I understand, the use of COMMON blocks is not recommended when working with widgets, but is ok otherwise. I have heard that there may be a problem if there are too many variables, but 20 seems ok to me.

I've got another question regarding this issue. I've been using IDL in widget p rogramming. Following DWF's advice, I do not use COMMON statements.

Here comes a big HOWEVER, however. I have an image processing tool which uses a base data set but many different widget modules, each with its own TOP LEVEL WIDGET. I would like to have access to this data from whatever or whichever widget I am working in. The data sets I read are created in the middle of my processing, so it would do no good to read it at the start and have a general pointer that can be included in each Top Level Widget. The only alternative is to keep track of all the widget ID's and then notify them when the data has been updated, a procedure discussed in DWF's book.

My problem is that this tool of mine is getting rather complicated (read: messy) and I have three WIDGET_CONTROL, GET_UVALUE statements at the beginning of many of my event handlers, because I have several info pointers which hold information that I need. (One each from the event ID, top and main info) Maybe I am just dis-organized, but it seems to be getting out of hand.

I am wondering whether using a COMMON statement for a set of variables or data arrays that aren't changing would not be such a bad idea. What about using constants such as PI or earth radius? Where can they be defined once and then used throughout a program ? I am considering putting my data in a COMMON statement so that I can get to it in an easy way (i.e. without (*(*infoPtr).dataArrPtr) type statements all over the place)

It is this or re-writing the tool from scratch (which it really needs if only I could find the time).

Rose

Subject: Re: Global variables and IDL
Posted by Martin Schultz on Tue, 20 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Walter Roberson (roberson@ibd.nrc.ca) writes:
>
>> Jim Russell wrote:
>>
>> :I'm certain that IDL has a global variable (I remember it being mentioned in
>> :Fanning's book), but don't remember how to invoke it. Maybe someone else can
>> :provide that for you.
>>
>> Use a common block. That requires adding only one extra statement to each

>> routine that uses an element of the common, and requires no other code
>> changes.
>
> I'm quite sure you didn't find a common block recommendation in
> Fanning's book. :-(
>

Here is my confession: yes, I use common blocks (and goto statements), and I don't even
feel too bad about it -- although it happens probably mostly for lack of knowledge of
better ways (perhaps Davids' third book will be able to change this?). It is defintively
true that you should avoid common blocks (and gotos) whenever possible (where "possible"
has to be defined in a feasible manner considering the available time etc.), and that
you should resort to UVALUES with pointers in widget applications as soon as there is
the slightest possibility that any one using these widgets will ever run more than one of
them (very few people probably write widgets when they intend a program exclusively for
personal use). But here are two examples where I used common blocks -- and I would be happy
to learn how I could have avoided them:

* in my EXPLORE tool (which can handle several "instances" at least if opened from within), I use
a common block to keep track of the drawing windows that have been opened and used. This is
needed to kill a window when the associated widget is closed as well as to open a new window
for a new widget instance. At first it would seem that I could simply
use the /free keyword to WINDOW and store the window number locally with the widget, but I have to close *all* windows when I exit the program.
Should I use the event notification method? Sounds like a viable thing
-- but I would have to rewrite a large part of a running program which my boss never likes ...

* in our new 3D model output analysis tool, we store data descriptors
for all data that has been read in a common block. This makes these data
available to all instances of the tool (although there currently is only
one and it is not even widgetized), and the common block avoids multiple
copies of large arrays which take up a lot of space and time when read.

Here are two little practice tips for common blocks:

(1) Make sure they are initialized correctly!
If you know where you encounter them the first time, you can write
something like

```
COMMON bla, thisdata

if (n_elements(thisdata) eq 0) then thisdata = ptr_new()
```

If not (i.e. you include your common blocks via @my_common , you can
call a specific init routine
with the above test from within your include file. To prevent an
infinite loop, you can write
something like this in my_common.pro:

```
COMMON myprobablyunnecessarycommonblock, thisdata, thatdata

if (strupcase(routine_name()) ne 'GAMAP_INIT') then gamap_init
```

Where the "routine_name()" function is available from my library (see
web site below).

(2) The use of @include files helps a great lot to avoid conflicting
definitions of common blocks which IDL doesn't like (you always have to
restart IDL when you change the structure of a common block).

(3) Become smarter than me and learn to use singletons and other methods
to avoid them ;-)

Martin.

--
---------- -------
Dr. Martin Schultz
Department for Engineering&Applied Sciences, Harvard University
109 Pierce Hall, 29 Oxford St., Cambridge, MA-02138, USA

phone: (617)-496-8318
fax : (617)-495-4551

e-mail: mgs@io.harvard.edu
Internet-homepage: http://www-as.harvard.edu/people/staff/mgs/
---------- -------

Subject: Re: Global variables and IDL
Posted by Pavel Romashkin on Wed, 21 Apr 1999 07:00:00 GMT

Hi Martin,

> In my oppinion it would be nice to be able to explicitely declare
> variables as global, such as
> GLOBAL glbarray=findgen(20)

I personally see no use for GLOBAL variables. I have been (and to some degree still am) a user of IGOR Pro for a long time and our applications relied heavily on the use of global variables. This led to lots of undesired results when the time had come to update applications: it was impossible to track down what GLOBALs were meant for, and required creating more GLOBALs to use old code. GLOBALs look to me like an inferior way of defining the required parameters, because then I lose the responsibility for "clean" programming.

> This would also help the problem discussed some months ago of how to
> retrieve information
> from widgets that are/were running in NO_BLOCK mode (e.g. data that has
> been manipulated).

I am sorry, I missed that, but why would you like to retrieve data from a widget into an interactive session before you are done with the widget?! II always save the data from the widget to a binary file and then restore it, if I want to manually play with it.

> and the program that wants to use that data can as always test for
> n_elements() gt 0.

Data can easily be requested by other programs from the widget's STATE that you keep around anyway in order to be able to act on the data from the event handling procedure.

> Also,
> if there are several instances of the widget application, the latest
> (current) application
> could overwrite the global variable by redefining it

That, in my opinion, is exactly where a lot of confusion will start from! How are you going to know which instance has just updated the GLOBAL? You will have to keep the track of it yourself.

> For me, a computer program is somewhat like a house: you have certain
> fixed structures
> like the doors and windows (which you can change but with some effort)
> and you have furniture
> that you can move around as you wish.

This sounds OK to me, but the way I found to be the best is to make "walls and doors" from the major DATA field in the widget STATE structure, and "furniture" was made of local variables inside daughter functions that branch from the main widget application. This way I don't need to worry that my GLOBAL "walls" can get altered by another process and my main program will not like that. Lack of the way to define session-global variables seems to be a good motivation to write self-consistent, independent and very flexible applications.

Subject: Re: Global variables and IDL
Posted by Martin Schultz on Wed, 21 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Stein Vidar Hagfors Haugan wrote:
>
> In article <371CD72E.C77A38A7@io.harvard.edu>
> Martin Schultz <mgs@io.harvard.edu> writes:
> [...]
> Why not simply create a detached widget_base containing a widget_draw
> instead ...
Thanks Stein! That's sort of what I have in mind for a future version of that program - including proper widget handling and a couple more options
like selecting plot symbols etc. I might even end up venturing into object
space and come up with some plot object as David adevrtizes on his page.
But first I need to finish this and that and ...

>
> But it's ok (in my humble opinion) to use common blocks to
> implement a system to keep track of such "global" data. I.e.,
> make two-three routines that share one (private) common block,
> along the lines of:
>
> REGISTER_ITEM,"NAME",DATA ;; Will "undefine" DATA,
> ;; to avoid copying
> DATAPTR = RETRIEVE_ITEM("NAME") ;; Returns a pointer to
> ;; the registered DATA item.

May not be as clean as this, but in principle that comes close to what I am doing. I
should add one more tip for use of common blocks which is - as you say -

* limit common blocks to a few routines that handle the input/output of data

Thinking a little more about this, true "global" variables may be a
better concept
although not supported in IDL. I think it would be nice to have the
flexibility of
normal IDL variables available for global variables as well (including
UNDEFINE ;-).
One example are option settings that are personal but (mostly) stable:
printer
paper size, preferred standard character size and font, maybe even a
sequence of
preferred plotting symbols and colors. You might argue that one can do
this with
system variables, but I found them a little too inflexible: it's hard to
re-assign
values (if you change the type or number of elements), and it is
somewhat clumsy
to test if a system variable was already defined? If you type
   if (n_elements(!undefined) eq 0) then defsysv,'!undefined',0
your program will stop (you have to use
defsysv,'!undefined',exists=answer and then
query the result of answer).

  In my oppinion it would be nice to be able to explicitely declare
variables as
global, such as
    GLOBAL glbarray=findgen(20)
This would also help the problem discussed some months ago of how to
retrieve information
from widgets that are/were running in NO_BLOCK mode (e.g. data that has
been manipulated).
Currently, this is another situation where you must use common blocks.
The GLOBAL approach
would be more flexible in that the widget application could define the
variables when needed,
and the program that wants to use that data can as always test for
n_elements() gt 0. Also,
if there are several instances of the widget application, the latest
(current) application
could overwrite the global variable by redefining it, or it could append
to it, etc. These
things are not easily done with common blocks.

   For me, a computer program is somewhat like a house: you have certain
fixed structures
like the doors and windows (which you can change but with some effort)
and you have furniture
that you can move around as you wish. When you settle in a new home, you
are likely to

change some of the fixed structures to accomodate your needs, but
afterwards you will mostly
rearrange furniture. Yet, you still wnt to be able to use all the doors
and windows that
are there.

Thanks for this helpful discussion,
Martin

--
---------- -------
Dr. Martin Schultz
Department for Engineering&Applied Sciences, Harvard University
109 Pierce Hall, 29 Oxford St., Cambridge, MA-02138, USA

phone: (617)-496-8318
fax : (617)-495-4551

e-mail: mgs@io.harvard.edu
Internet-homepage: http://www-as.harvard.edu/people/staff/mgs/
---------- -------

Subject: Re: Global variables and IDL
Posted by steinhh on Wed, 21 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <371CD72E.C77A38A7@io.harvard.edu>
Martin Schultz <mgs@io.harvard.edu> writes:
[...]
> But here are two examples where I used common blocks --
> and I would be happy
> to learn how I could have avoided them:
>
> * in my EXPLORE tool (which can handle several "instances" at
> least if opened from within), I use a common block to keep track
> of the drawing windows that have been opened and used. This is
> needed to kill a window when the associated widget is closed as
> well as to open a new window for a new widget instance. At first
> it would seem that I could simply use the /free keyword to
> WINDOW and store the window number locally with the widget, but
> I have to close *all* windows when I exit the program. Should I
> use the event notification method? Sounds like a viable thing --
> but I would have to rewrite a large part of a running program
> which my boss never likes ...

I've no experience with the EXPLORE tool, but from your

description here it seems like you're creating normal draw windows
that "belong" to a given widget instance. Why not simply create a
detached widget_base containing a widget_draw instead, then get
the draw window number (for WSET,WIN) through WIDGET_CONTROL,
DRAW_ID,GET_VALUE=WIN, and store it in the info structure for the
"owner" widget instance. By setting the owner widget instance as
the group leader for the detached top level base containing the
widget draw window, the window will automatically be killed
whenever your widget instance is killed. You should take a look at
some of David F's applications to ensure that resizing the draw
windows will work, though...

> * in our new 3D model output analysis tool, we store data
> descriptors for all data that has been read in a common
> block. This makes these data available to all instances of the
> tool (although there currently is only one and it is not even
> widgetized)

And non-widgetized suites of programs are in fact a lot harder to
do without any common blocks at all. Nothing will do except
explicitly passing (a pointer to) data around in all function
calls. Clean as h*ll, but not very practical!

But it's ok (in my humble opinion) to use common blocks to
implement a system to keep track of such "global" data. I.e.,
make two-three routines that share one (private) common block,
along the lines of:

```
  REGISTER_ITEM,"NAME",DATA       ;; Will "undefine" DATA,
                            ;; to avoid copying
  DATAPTR = RETRIEVE_ITEM("NAME") ;; Returns a pointer to
                            ;; the registered DATA item.
```

If you want to avoid pointer notation, you could
temporarily put your data into local variables by

```
  DATA = TEMPORARY(*DATAPTR) ;; Get data without copying
  ;; Do the processing
  *DATAPTR = TEMPORARY(DATA) ;; Put it back.
```

To be crash tolerant and still avoid the pointer notation as well
as data copying, you'd need a slightly different approach, with
one publicly available common block (yes!!!), and do something
like this in your "client" programs:

```
PRO MY_ROUTINE
  COMMON DATA_SYSTEM_CACHE,ITEMNAME,DATA ;;You can use whatever
                              ;;variable names that
```

```
                        ;;are appropriate for
                        ;;this procedure.

  NEWDATA = READ_DATA(FILENAME)
  REGISTER_ITEM,"NAME",NEWDATA

  NEWDATA = READ_DATA(FILENAME2)
  REGISTER_ITEM,"ANOTHER_NAME",NEWDATA

  CHECKIN_ITEM,"NAME"  ;; Stores any current DATA_SYSTEM_CACHE
                ;; contents and puts the requested
                ;; data there instead.
  DATA.element = 5.0

  CHECKIN_ITEM,"ANOTHER_NAME" ;; Ditto.
  DATA.something_else = "True"

END
```

I made something like this (though a *lot* less general) back in, oh, 1994 in fact, and the software is still in operation. Of course back then we didn't even have handles, much less pointers. I had to use unrealized widget bases, and the /no_copy keyword...

Regards,

Stein Vidar

Subject: Re: Global variables and IDL
Posted by rmlongfield on Wed, 21 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <371CD72E.C77A38A7@io.harvard.edu>,
  Martin Schultz <mgs@io.harvard.edu> wrote:

Some useful advice about COMMON blocks

Hi All, Thanks for the input. Martin especialy, I had a look at your explore almost a year ago but since I was just starting, I didn't use it. I plan on having a second look at it because my familiarity with widgets has improved and I actually have results that I would like to try analysing with EXPLORE. You mentioned dealing with GROUPS and that is particularly a problem I am encountering now because it is another variable that needs to be stored somewhere. It is however useful when opening and closing different windows.

Rose

Subject: Re: Global variables and IDL
Posted by steinhh on Thu, 22 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <371E4831.1CDB0AFE@io.harvard.edu>
Martin Schultz <mgs@io.harvard.edu> writes:

> Thinking a little more about this, true "global" variables may
> be a better concept although not supported in IDL. I think it
> would be nice to have the flexibility of normal IDL variables
> available for global variables as well (including UNDEFINE ;-).
> One example are option settings that are personal but (mostly)
> stable: printer paper size, preferred standard character size
> and font, maybe even a sequence of preferred plotting symbols
> and colors. You might argue that one can do this with system
> variables, [...]

No, no, no :-)

Singleton objects. That's the way to go. See the posting by
J.D. Smith:

http://www.dejanews.com/[ST_rn=ap]/getdoc.xp?AN=365131522

Personally, I prefer the inheritance and obj_new('class')
creation method instead of the singleton('class') method (he
suggests both), but that's only a matter of taste.

> In my oppinion it would be nice to be able to explicitely
> declare variables as global, such as
> GLOBAL glbarray=findgen(20)
> This would also help the problem discussed some months ago of
> how to retrieve information from widgets that are/were running
> in NO_BLOCK mode (e.g. data that has been manipulated).

No, no :-)

Again - singleton objects will do this for you. A singleton class instance is, in fact, a kind of global variable. You can access it from anywhere, simply by asking for an object of that class. Since only one object of that kind exists, it must be the same object that the other programs are talking to. If this object is keeping track of your data/preferences/whatever, then *you* can ask the object to return a data pointer, and your widget program can ask for the same data pointer. Though you're stuck with pointer notation....unless you want to use the common block cache approach.

> Currently, this is another situation where you must use common
> blocks. The GLOBAL approach would be more flexible in that the
> widget application could define the variables when needed, and
> the program that wants to use that data can as always test for
> n_elements() gt 0. Also, if there are several instances of the
> widget application, the latest (current) application could
> overwrite the global variable by redefining it, or it could
> append to it, etc. These things are not easily done with common
> blocks.

Hmmm. Most sensible programs that would use n_elements() to check for the existence of a (global) variable should know the name in advance. If it doesn't know the name, it could just as well ask some singleton object for a registered data set by that name (since you cannot write "if n_elements(unknown_name) then..." into your program anyway).

But there might be something to be said about the user point of view. Let's say I have a /no_block widget up and running all the time to pick data sets from a data base.. If there's no widget for the manipulation of the data, there isn't really much of a problem, cause I could click on the widget to read in data, then say:

IDL> mydata = fetchdata()

or

IDL> dummy=obj_new('communicator',object=comm) ;; Only once per IDL session
IDL> mydata = comm->fetchdata()

In other words, I have full control over the main level variable names, and I can send the data to other routines quite easily. The fetchdata() routines above would use RETURN,TEMPORARY(*dataptr) to avoid copying.

The problem is to have the data available at the main level, with

any variable name, as well as available for a data manipulation widget program, without using kludgy pointer notation at the main level.

The data manipulation widget program would be *very* kludgy indeed, since you cannot hardcode the variable names, you'd have to use a lot of EXECUTE() statements etc..

And you'd still have to tell your widget program which of the main level programs it was supposed to manipulate at any time... In my opinion, it doesn't look like there's very much to gain even if global variables were possible.

Regards,

Stein Vidar

Subject: Re: Global variables and IDL
Posted by David Foster on Thu, 22 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Martin Schultz wrote:
>
> David Fanning wrote:
>>
>> Walter Roberson (roberson@ibd.nrc.ca) writes:
>>
>>> Jim Russell wrote:
>>>
>>> :I'm certain that IDL has a global variable (I remember it being mentioned in
>>> :Fanning's book), but don't remember how to invoke it. Maybe someone else can
>>> :provide that for you.
>>>
>>> Use a common block. That requires adding only one extra statement to each
>>> routine that uses an element of the common, and requires no other code
>>> changes.
>>
>> I'm quite sure you didn't find a common block recommendation in
>> Fanning's book. :-(
>>
>
> Here is my confession: yes, I use common blocks (and goto statements),
> and I don't even
> feel too bad about it -- although it happens probably mostly for lack of
> knowledge of
> better ways (perhaps Davids' third book will be able to change this?).

I applaud your courage Martin! My name is David Foster, and I have
been a common-block user...

I think the idea of global variables would be very powerful, and the
ability to have global pointers would be extremely useful in our
applications. We have a series of programs that are run consecutively,
on the same LARGE data-sets, and we use common blocks to store these
data sets. We find this approach much faster than other possibly
"cleaner" programming approaches, and for our purposes we would never
want to have more than one instance of one of these programs running.
If we did, we could certainly structure the common variables to
accomodate this!

If global variables were to be introduced into IDL, I would hope
that we would also get some mechanism for limiting the visibility
of these variables, some way to say "use this global variable" within
a routine. Something similar to the EXTERN construct in C.

Coming from a C background, my first impression of IDL was that it
provides *very* limited control over scope and visibility of variables
and function/procedure names.

Dave
--

~~~~~~ ~~~~~~~
David S. Foster Univ. of California, San Diego
Programmer/Analyst Brain Image Analysis Laboratory
foster@bial1.ucsd.edu Department of Psychiatry
(619) 622-5892 8950 Via La Jolla Drive, Suite 2240
La Jolla, CA 92037
~~~~~~ ~~~~~~~

Subject: Re: Global variables and IDL
Posted by Martin Schultz on Fri, 23 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

rmlongfield@my-dejanews.com wrote:
>
Hi Rose,

[...]
> Another problem with WIDGET_CONTROL (and I have seen others here with a
> similar problem when starting) is that the variable in the statement:
>
> WIDGET_CONTROL,event.id,GET_UVALUE= variable
>

> Can have ANY name. One can call it variablePTR or variableString or whatever.
> The name is completely irrelevent if what has been saved in this UVALUE is an
> integer.

that's probably unavoidable, since the programmer has to make some
decisions somewhere. But fortunately enough, IDL provides you with the
SIZE command (which has very useful keywords since 5.xx) so you can
always find out whether the UVALUE info is of type BYTE, INTEGER, LONG,
... POINTER (or even OBJECT ?) And if it is not what you thought it
should be, you either have more than 1 person muddling around with the
code or you are "actively enhancing your IDL skills" which means you
make so many changes to old programs that you loose control over all the
side effects ;-)

Martin.

--
---------- -------
Dr. Martin Schultz
Department for Engineering&Applied Sciences, Harvard University
109 Pierce Hall, 29 Oxford St., Cambridge, MA-02138, USA

phone: (617)-496-8318
fax : (617)-495-4551

e-mail: mgs@io.harvard.edu
Internet-homepage: http://www-as.harvard.edu/people/staff/mgs/
---------- -------

Subject: Re: Global variables and IDL
Posted by L. Paul Mix on Fri, 23 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

David Foster wrote:

> <Deleted >
>
> I applaud your courage Martin! My name is David Foster, and I have
> been a common-block user...
>
> I think the idea of global variables would be very powerful, and the
> ability to have global pointers would be extremely useful in our
> applications. We have a series of programs that are run consecutively,
> on the same LARGE data-sets, and we use common blocks to store these
> data sets. We find this approach much faster than other possibly
> "cleaner" programming approaches, and for our purposes we would never
> want to have more than one instance of one of these programs running.

> If we did, we could certainly structure the common variables to
> accomodate this!
>
> If global variables were to be introduced into IDL, I would hope
> that we would also get some mechanism for limiting the visibility
> of these variables, some way to say "use this global variable" within
> a routine. Something similar to the EXTERN construct in C.
>
> Coming from a C background, my first impression of IDL was that it
> provides *very* limited control over scope and visibility of variables
> and function/procedure names.
>
> Dave
> --
>
> ~~~~~~ ~~~~~~~
> David S. Foster Univ. of California, San Diego
> Programmer/Analyst Brain Image Analysis Laboratory
> foster@bial1.ucsd.edu Department of Psychiatry
> (619) 622-5892 8950 Via La Jolla Drive, Suite 2240
> La Jolla, CA 92037
> ~~~~~~ ~~~~~~~

As a long time IDL programmer I also admit to using common blocks for global variables.

Common blocks for widget programs are generally a bad idea, but, for variables with truly
universal scope in an application, common blocks allow the variable the necessary scope

while hiding them from the user.

RSI developed pointer variables as a replacement for handles but they suffer from the problem
that a user can be completely overwhelmed if he types: help, /heap and an application has used
several hundred pointers.

My background is Fortran not C, but I totally agree with Dave that the ablility to limit the scope
and visibility of a variable should not be overlooked in the quest for global variables.

Paul
==========
L. Paul Mix
Distinguished Member of the Technical Staff
Electromagnetics and Plasma Physics Analysis

Sandia National Laboratories
MS 1186, P.O. Box 5800
Albuquerque, NM 87185-1186
E-mail: lpmix@sandia.gov
Phone: (505) 845-7493
FAX: (505) 845-7890

Subject: Re: Global variables and IDL
Posted by rmlongfield on Fri, 23 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Hi all,
Whoah, this global variable stuff is getting heavy. I know that deep down in my soul, objects is the way to go. Objects seem to be similar to structures. This means that many different things associated with it can be lumped together in one group and then referenced only by passing around the name of the group and not what is in it.

Stein wrote:
> Again - singleton objects will do this for you. A singleton class
> instance is, in fact, a kind of global variable. You can access
> it from anywhere, simply by asking for an object of that
> class. Since only one object of that kind exists, it must be the
> same object that the other programs are talking to. If this
> object is keeping track of your data/preferences/whatever, then
> *you* can ask the object to return a data pointer, and your
> widget program can ask for the same data pointer. Though you're
> stuck with pointer notation....unless you want to use the common
> block cache approach.

Are you telling us that we don't even have to know what the variable name is? We don't have to pass around ANY information? We just have to type in some routine that says, "See if anything is in this box " Aha, but we still have to know how box would be defined (class?'), so this definition of box must be passed around. If we manage to do this correctly, and if it finds something, then can we assume that it is what we are looking for?

For example, many times I have used WIDGET_CONTROL to get back a pointer and have found out that the wrong value is returned because I used the wrong widget ID. (Lost track of whether TOP or ID was holding the information I wanted). Can this still happen with OBJECTS or is it more "fool proof"?
Another problem with WIDGET_CONTROL (and I have seen others here with a similar problem when starting) is that the variable in the statement:

WIDGET_CONTROL,event.id,GET_UVALUE= variable

Can have ANY name. One can call it variablePTR or variableString or whatever.
The name is completely irrelevent if what has been saved in this UVALUE is an
integer.

Can objects solve this confusion? I do not know.
But I am reading all this with great interest and appreciate the discussion.

Rose

Subject: Re: Global variables and IDL
Posted by davidf on Sat, 24 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Folks,

In returning to this newsgroup after a short absence I have
been shocked and appalled to see evidence of the ever
quickening pace of moral decay in the world today. I
refer, of course, to the large number of IDL programmers
confessing to the use of COMMON blocks in their programs.

I, too, used a COMMON block--once. I think I stored the
seed of a random number in it until I realized I could just
as well include the seed in my info structure along with all
the other stuff I needed in my program. And there is
evidence that even more programmers might soon come forward
with the same confession. It is hard to imagine how
it could be otherwise, what with RSI supplying example
code liberally sprinkled with COMMON blocks and most
of us cutting our teeth on FORTRAN programs. (Is it
my imagination, or are most of the people confessing to
COMMON block usage at least middle aged?)

But, alright. The evidence is overwhelming--even
to me. There are some very good reasons why a COMMON
block may be used for some good purpose. Just because
I've never stumbled onto one in the course of my own
programming doesn't make it any less true. It's
probably because I don't have enough programming
imagination to see how clever using a COMMON block
would be.

But I would like to correct a few misconceptions about
COMMON blocks, if I may.

Martin Shultz writes:

> But here are two examples where I used common blocks --
> and I would be happy
> to learn how I could have avoided them:
>
> * in my EXPLORE tool (which can handle several "instances" at least if
> opened from within), I use
> a common block to keep track of the drawing windows that have been
> opened and used.

There is absolutely no need for a COMMON block in this instance. Widget programs can easily keep track of drawing windows through the used of draw widget IDs, even draw widgets that are detached from the main window. See, for example, the SLICE program on my web page, which can spawn any number of copies of itself, or--for a simpler example, the ZIMAGE program which can tell if the zoom window is open and on the display or not. If it is not on the display (it had, perhaps, been killed by the user) the program simply creates another.

http://www.dfanning.com/programs/zimage.pro

Martin goes on to write this, which scares me very much indeed:

> This is
> needed to kill a window when the associated widget is closed as well as
> to open a new window
> for a new widget instance. At first it would seem that I could simply
> use the /free keyword to WINDOW and store the window number locally with
> the widget, but I have to close *all* windows when I exit the program.
> Should I use the event notification method?

Using normal IDL graphics windows from within a widget program is a *terrible* idea! (No offense to Martin, who I know is a VERY good IDL programmer.) But if you display graphics in normal IDL windows you have absolutely no control over them from within your widget program. You would, indeed, probably need a COMMON block to account for them. If you need to display graphics inside a widget program I can say categorically, without exception, that you need a draw widget in which to display them! (I'm not saying it can't be done. I'm saying you don't want to do it.)

Cheers,

David

P.S. After thinking about it for a long time I just
put a COMMON block back into one of the programs on
my web page. (I won't say which one, because I'm not
certain that I am going to leave it there.) Whew!
It wasn't as bad as I thought it was going to be. ;-)

--
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: Global variables and IDL
Posted by davidf on Mon, 26 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Martin Schultz (mgs@io.harvard.edu) writes:

> I feel ashamed.

Uh, look, this confession business has me VERY
nervous. If we are all going to start telling
about our worst programs, well...let's just say
I *DON'T* want to get into it. And please don't
mention PS_FORM to anyone! :-(

> I should probably tell everyone to delete their
> version of EXPLORE and start right away coding it properly.

Totally unnecessary. I've found that just putting the
program in the public domain means that it will come back
to you in short order written *exactly* the way you
thought you were writing it in the first place. :-)

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: Global variables and IDL
Posted by Martin Schultz on Mon, 26 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Martin goes on to write this, which scares me very much indeed:
> [...]
> Using normal IDL graphics windows from within a widget program
> is a *terrible* idea!

> Cheers,
>
> David
>

Hi David (and others),

I feel ashamed. I should probably tell everyone to delete their version of EXPLORE and start right away coding it properly. (if I only had time for this). I must say that I started to write EXPLORE when I still considered myself an IDL novice (they didn't have insight (pun;-) back then), and David was probably still dreaming about his book. So, in short: it's bad code but it works to my satisfaction. Maybe I should distribute it as sav files only ....
Martin.

--
---------- -------
Dr. Martin Schultz
Department for Engineering&Applied Sciences, Harvard University
109 Pierce Hall, 29 Oxford St., Cambridge, MA-02138, USA

phone: (617)-496-8318
fax : (617)-495-4551

e-mail: mgs@io.harvard.edu
Internet-homepage: http://www-as.harvard.edu/people/staff/mgs/
---------- -------

Subject: Re: Global variables and IDL
Posted by Struan Gray on Tue, 27 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning, davidf@dfanning.com writes:

> Martin Schultz (mgs@io.harvard.edu) writes:
>>
>> I feel ashamed.
>
> Uh, look, this confession business has me VERY
> nervous.

<smug_git>

I've *never* used a common block, and I strip 'em out of any code I get from elsewhere.

</smug_git>

An alternative to common blocks that I use extensively is to create a system variable with a unique name (it helps that I work in a research group called 'synkrotronljusfysik') and then use it to store the start handle of a linked list. All my global variables are stored by name in the list. I have utility routines to add, delete, move and modify items, and I can create hierarchies by making any list item the start handle to a sub-list.

All my widget programs know that they can find things like user preferences, large datasets and default directory names by looking for the relevant named parameter in the list. When a widget dies, it's cleanup routine deletes any variables associated only with itself. One of the reasons I still use handles a lot (despite RSI's rather sneering insistence that we use pointers these days) is that if a list is created properly the whole thing can be disposed of automatically simply by freeing the first handle.

To acheive true IDLguru status I should probably objectify the whole thing with a singleton instead of a start handle, but it works well enough that I'm frying other fish for now.

Struan