Subject: Common block conumdrum
Posted by williams on Mon, 03 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

When developing programs in IDL, I often work with common blocks.
However, if I try to change the definition of an existing common
block, I get an error because it IDL thinks that I am trying to use a
pre-existing common block in a naughty manner. According to the
user's manual (v3.6 p6-10) this is what IDL should do.
  However, when creating new programs, I often want to change the
definition of the common block as the program developes. How can I do
this, short of exiting IDL and starting over? Is there some "delvar"
equivalent for deleting common blocks or programs from memory?

Sincerely,
Daniel Williams

```
 +---------------------------------------------------------- ------------------+
|Daniel L. Williams | Email:  williams@srl.caltech.edu                    |
|Space Radiation Lab| Telly:  818/395-6634                                |
|Caltech 220-47     | Fax:    818/449-8676                                |
|Pasadena, Ca 91125 | WWW: http://www.srl.caltech.edu/personnel/williams.html |
 +---------------------------------------------------------- ------------------+
```

Subject: Re: Common block conumdrum
Posted by wonko on Tue, 04 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

williams@skrymir.srl.caltech.edu (Daniel Williams) wrote:

> When developing programs in IDL, I often work with common blocks.
> However, if I try to change the definition of an existing common
> block, I get an error because it IDL thinks that I am trying to use a
> pre-existing common block in a naughty manner. According to the
> user's manual (v3.6 p6-10) this is what IDL should do.
>   However, when creating new programs, I often want to change the
> definition of the common block as the program developes. How can I do
> this, short of exiting IDL and starting over? Is there some "delvar"
> equivalent for deleting common blocks or programs from memory?

I don't think there is any. So I just exit IDL and restart -- our SUNs
are getting faster and faster :)
Another idea would be the usage of global variables. They can be created
and changed with DEFSYSV at the main level.
I usually prefer this, only in my actual project I use common blocks,
just out of curiousity, I wanted to know what's so hot about them. Are
there drawbacks I don't see?

Alex
--
Alex Schuster   Wonko@weird.cologne.de
                         alex@pet.mpin-koeln.mpg.de

Subject: Re: Common block conumdrum
Posted by David Foster on Tue, 04 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Daniel Williams wrote:
>
> When developing programs in IDL, I often work with common blocks.
> However, if I try to change the definition of an existing common
> block, I get an error because it IDL thinks that I am trying to use a
> pre-existing common block in a naughty manner. According to the
> user's manual (v3.6 p6-10) this is what IDL should do.
> However, when creating new programs, I often want to change the
> definition of the common block as the program developes. How can I do
> this, short of exiting IDL and starting over? Is there some "delvar"
> equivalent for deleting common blocks or programs from memory?

There are times when common blocks are very useful, such as when
you must share data between various programs. If you are using
common blocks so that you have "global" variables within a single
application, I would strongly suggest that you follow the example
presented in the User's Guide, Ch. 21 "Writing a Compound Widget",
and store the "state" information in the uvalue of the first
child of your main base. Then, in the event handler you can use:

```
 stash = WIDGET_INFO( event.top, /child )
 WIDGET_CONTROL, stash, get_uvalue = state, /no_copy
```

to retrieve this state structure. Then pass the structure to your
routines. See the example above for details.

This method is a little bit slower, but a lot cleaner and safer
(you don't have to worry about using "reserved" variable names and
overwriting something in the common block, for example).

If you really need a common block, then the suggestion by Phil
Williams to use a large structure as your first variable in the
common block is a good one; you can add tags to this structure as
you need them. This also avoids the problem of inadvertently re-using
common-block variables.

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 2200
                         La Jolla, CA  92037
                         [ UCSD Mail Code 0949 ]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~
```

Subject: Re: Common block conumdrum
Posted by Phil Williams on Tue, 04 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Daniel Williams wrote:
>
> When developing programs in IDL, I often work with common blocks.
> However, if I try to change the definition of an existing common
> block, I get an error because it IDL thinks that I am trying to use a
> pre-existing common block in a naughty manner. According to the
> user's manual (v3.6 p6-10) this is what IDL should do.
>   However, when creating new programs, I often want to change the
> definition of the common block as the program developes. How can I do
> this, short of exiting IDL and starting over? Is there some "delvar"
> equivalent for deleting common blocks or programs from memory?
>

There's no way to do this. I have a couple of suggestions though:

1) Get rid of the common blocks. I used to use them alot as well, but
found that my programming became a lot cleaner when I started grouping
like things in structures and passed them into procedures.

2) Have a common block that contains an anonymous structure. This was
the first thing I did on my way to #1. Make sure that the structure is
anonymous or you'll run into the same problem as you did with common
blocks.

Good luck,
Phil
--

```
/*********************************************************** ********/
 Phil Williams, Ph.D.
 Research Instructor
 Children's Hospital Medical Center    "One man gathers what
 Imaging Research Center                another man spills..."
 3333 Burnet Ave.                      -The Grateful Dead
```

Cincinnati, OH 45229
email: williams@irc.chmcc.org
URL: http://scuttle.chmcc.org/~williams/
/*** ********/

Subject: Re: Common block conumdrum
Posted by brian.jackel on Tue, 04 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

In article <5ffnnr$o0g@gap.cco.caltech.edu> williams@skrymir.srl.caltech.edu (Daniel Williams) writes:

> When developing programs in IDL, I often work with common blocks.
> However, if I try to change the definition of an existing common
> block, I get an error because it IDL thinks that I am trying to use a
> pre-existing common block in a naughty manner. According to the
> user's manual (v3.6 p6-10) this is what IDL should do.
>   However, when creating new programs, I often want to change the
> definition of the common block as the program developes. How can I do
> this, short of exiting IDL and starting over? Is there some "delvar"
> equivalent for deleting common blocks or programs from memory?

For a quick hack, define the common block as

```
COMMON MYBLOCK,var1,var2,var3,dum1,dum2,dum3,dum4,dum5
```

with a bunch of dummy assignments tacked on. Then, when you
have new variables, just rename one of the dummy variables. This,
combined with the use of structures, is what I do when I'm writing
something new and the common block keeps changing. Not quite
what you were looking for, but it usually does the trick.

Brian Jackel

Subject: Re: Common block conumdrum
Posted by Phil Williams on Wed, 05 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

John-David Smith wrote:
>
> Struan Gray wrote:
>
>>   This works but has it's own problems. I often need to have the
>> state information when the widget is killed so that I can save
>> parameters and variables to disk or pass a response to another widget.

>>   David's technique works well if you can be sure that the user will
>> only ever kill the widget with 'close'/'quit'/'cancel' buttons, but if
>> the widget can be killed another way (with the xmanager tool or via a
>> close box provided by the operating system's window manager) I find
>> that by the time my widget knows it is being killed only the top level
>> base remains and my state information is lost.
>
> XMANAGER's CLEANUP callback mechanism gives you control of your dying
> widget after almost all of it has been killed. This is problematic,
> since you often can't then get to your state info.
>
> A technique I often use to prevent this behaviour is the KILL_NOTIFY
> mechanism. Its use is strongly discouraged in various IDL references,
> but the thrust of the warning is "don't assign a kill_notify procedure

<snip>

Figured I'd add my 2 cents again. They way I handle this is to have a
handle that would have any "global" data. Then any additional "global"
data that I needs I simply make a child handle of that main handle. I
then share this handle with any of my widgets that need it. If a popup
then dies the "main" widget still knows where the handle is and is also
responsible for it's cleanup when it dies.

It is my understanding that with IDL 5.0 that we are going to be more
careful with issues such as these since you will be able to run widgets
and still have access to the command line.

Good luck,
Phil
--
```
 /********************************************************* ********/
  Phil Williams, Ph.D.
  Research Instructor
  Children's Hospital Medical Center    "One man gathers what
  Imaging Research Center                another man spills..."
  3333 Burnet Ave.                      -The Grateful Dead
  Cincinnati, OH 45229
  email: williams@irc.chmcc.org
  URL: http://scuttle.chmcc.org/~williams/
 /********************************************************* ********/
```

Subject: Re: Common block conumdrum
Posted by J.D. Smith on Wed, 05 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Struan Gray wrote:

>   This works but has it's own problems. I often need to have the
> state information when the widget is killed so that I can save
> parameters and variables to disk or pass a response to another widget.
>   David's technique works well if you can be sure that the user will
> only ever kill the widget with 'close'/'quit'/'cancel' buttons, but if
> the widget can be killed another way (with the xmanager tool or via a
> close box provided by the operating system's window manager) I find
> that by the time my widget knows it is being killed only the top level
> base remains and my state information is lost.

XMANAGER's CLEANUP callback mechanism gives you control of your dying widget after almost all of it has been killed. This is problematic, since you often can't then get to your state info.

A technique I often use to prevent this behaviour is the KILL_NOTIFY mechanism. Its use is strongly discouraged in various IDL references, but the thrust of the warning is "don't assign a kill_notify procedure for a top level base". As long I avoided this, I haven't had problems. The key recognition is that the callback procedure specified by KILL_NOTIFY only has access to the user value of that widget for which it was specified -- no other widgets are gauranteed to be alive, so widget_control doesn't let you even try to access them. The specified widget can't be the top level base, since XMANAGER wouldn't like this. What is needed is a widget that contains the state info but is *not* the top level base. I therefore use the the base's first child's uvalue to store my state structure (the de facto for compound widgets), and assign the callback procedure to the *child* widget. E.g. I might say:

```
   widget_control,widget_info(base,/CHILD),SET_UVALUE=state,/NO _COPY, $
      KILL_NOTIFY='WIDGET_KILL_PROCEDURE'
```

The strength of this technique lies in the fact that when the callback procedure is called, the state is still defined, since it's in the user value of the widget... so you can free handles, save info, etc. This works whether the user exits with the DONE button, or otherwise.   In fact, it's also convenient for application development, since even a crashed widget (after you retall and xmanager) will call its kill_notify, freeing handles and preventing memory leakage.

Perhaps there is substance to the warnings, and problems with using XMANAGER jointly with KILL_NOTIFY do exist, but I haven't experienced them, as long as I stayed away from the TLB.

JD

Subject: Re: Common block conumdrum
Posted by Struan Gray on Wed, 05 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

David Foster replies to:

> Daniel Williams:
>>
>> However, when creating new programs, I often want to
>> change the definition of the common block as the program
>> developes. How can I do this, short of exiting IDL and
>> starting over?
>
> I would strongly suggest that you follow the example
> presented in the User's Guide, Ch. 21 "Writing a Compound
> Widget", and store the "state" information in the uvalue of
> the first child of your main base.

This works but has it's own problems. I often need to have the state information when the widget is killed so that I can save parameters and variables to disk or pass a response to another widget. David's technique works well if you can be sure that the user will only ever kill the widget with 'close'/'quit'/'cancel' buttons, but if the widget can be killed another way (with the xmanager tool or via a close box provided by the operating system's window manager) I find that by the time my widget knows it is being killed only the top level base remains and my state information is lost.

I get round this (and the original problem of wanting to change data structures in common blocks) by keeping common variables and state information in linked lists of handles. IDL handles have lots of built in funtions to define trees and lists so it is really easy to build and modify data structures on the fly.

I tend to use what I call a 'parameter', which is a pair of handles, one pointing to the data and another pointing to a string which contains the parameter name. The data handle is declared as a child of the name handle and the name handle is part of a linked list or a tree of handles all linked back to a master handle whose value is stored in a variable of type long. I have a bunch of utility routines to create, delete, change and move parameters, all of which accept the name and some data so the code is really easy to read. One nice side effect is that the entire list of parameters can be deleted and their associated memory reclaimed just by freeing the master handle.

If the parameters are needed by lots of program units (for example working directories, user information and graphics defaults) I create a system variable for the master handle. Otherwise a common block can

hold one or more master handles, which has the advantage that the
'common' statements in the code make dependencies explicit.

To get round the widget problem I use the user value of the main
base widget as a master handle, which can be retreived and the state
information read even in the cleanup routine called by the xmanager.
Other widgets who want to use the user value for their own data have
to use the paramater mechanism too, but that places no real
restrictions on what data they can store there so for me the 'bad'
coding is worth doing.

Struan

Subject: Re: Common block conumdrum
Posted by david.eldon on Fri, 16 May 2014 21:07:29 GMT
View Forum Message <> Reply to Message

On Monday, March 3, 1997 12:00:00 AM UTC-8, Daniel Williams wrote:
> When developing programs in IDL, I often work with common blocks.
> However, if I try to change the definition of an existing common
> block, I get an error because it IDL thinks that I am trying to use a
> pre-existing common block in a naughty manner. According to the
> user's manual (v3.6 p6-10) this is what IDL should do.
> However, when creating new programs, I often want to change the
> definition of the common block as the program developes. How can I do
> this, short of exiting IDL and starting over? Is there some "delvar"
> equivalent for deleting common blocks or programs from memory?
>
> Sincerely,
> Daniel Williams
> +--- -----------------+

I know this is an old discussion, but it still appears in searches, so I'd like to add this: There is now
(and may not have been in 1997) an executive command to do this: .reset_session
http://www.exelisvis.com/docs/_RESET_SESSION.html
http://www.physics.nyu.edu/grierlab/idl_html_help/symbols8.h tml#wp983175