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
