Subject: Re: Erasing common block variables
Posted by davidf on Tue, 24 Sep 1996 07:00:00 GMT
View Forum Message <> Reply to Message

Runar Jorgensen writes:

> Neither the command .run nor .rnew erases
> the common block variables (2-11U). So whenever I need to
> change my common blocks I need to terminate IDL and
> restart IDL again. Very unpractical.
>
> The same problem arises when dealing with structures.
> Is there a way around this?

Well, the same problem arises with *NAMED* structures.

Anonymous structures are much easier to handle (pun intended) in this regard, which is why they are often used to hold program information in widget programs, whose widget IDs often act as handles.

Would you believe that I *never* write a program with a common block in it? Well, all right, almost never. It *is* true, however, that I make a point of never writing a *widget* program with a common block in it. :-)

In a widget program the ID of the top-level base is often used as a handle or pointer to a storage location, i.e. the UValue of the top-level base.

For example, the top-level base is created like this:

```
tlb = WIDGET_BASE(Title='Pointer Example')
```

The variable "tlb" is the "pointer" or handle. Now, you create an *anonymous* structure to store all the information you previously stored in a common block. (You use an anonymous structure because you are going to forget something and you will want to add it later. See above on why common blocks are a pain in the neck!)

```
info = { data:data, drawWidgetID:drawWidgetID, $
   otherStuff:otherStuff}
```

You save the info structure in the UValue of the top-level base:

```
WIDGET_CONTROL, tlb, SET_UVALUE=info
```

Now, when you want the information you have to de-reference it. In widget programs the variable "tlb" is usually named something else when it's time to de-reference the handle. It's usually named "event.top" or something like that. So the de-referencing looks like this:

```
WIDGET_CONTROL, event.top, GET_UVALUE=myInfo
```

If you want to plot the data, do this:

```
PLOT, myInfo.data
```

This whole business can be done just as easily with handles; the commands you use are just slightly different. For example, to create the "pointer" or handle that will point to the location where you store the data, you type:

```
ptr = HANDLE_CREATE()
```

Get your "info" structure (or whatever) and store it there:

```
info = { data:data, drawWidgetID:drawWidgetID, otherStuff:otherStuff}
HANDLE_VALUE, ptr, info, /SET
```

Later, you want to de-reference it, like this:

```
HANDLE_VALUE, ptr, myInfo
```

The beauty of handles and anonymous structures is that you can make changes to either at any time and IDL doesn't much care.

Just one caveat. You can't change the definition of an anonymous structure field without redefining the entire structure. Let me give you an example.

Suppose the "data" in my anonymous info structure above started off as a FLTARR(100) and later I have another variable, say "newdata", that is a FLTARR(200). I could not do this:

```
info.data = newdata
```

This is illegal because the data field of the info structure is set up to hold only 100 floating values. There isn't enough room. But I could redefine the data field in the info structure, like this:

```
info = {data:newdata, drawWidgetID:info.drawWidgetID, $
  otherStuff:info.otherStuff}
```

Now I have the same info structure as before, except that the data field is twice the size it used to be. Be sure you free up your handles (and the data stored there) when you are finished with them, or you will have memory leaking from your programs.

```
HANDLE_FREE, ptr
```

Hope this gives you some ideas.

Cheers!

David

--
David Fanning, Ph.D.
Phone: 970-221-0438
Fax: 970-221-4728
E-Mail: davidf@fortnet.org
