Subject: Re: Common block conundrum
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