Subject: Re: static variables
Posted by Peter Mason on Tue, 01 Oct 1996 07:00:00 GMT
View Forum Message <> Reply to Message

On 30 Sep 1996, Esfandiar Bandari wrote:
> I am newcomer to idl and this news group! I was wondering if
> there is any clean way of having static variables inside files or
> functions (as in C or C++). These are variables that keep their value
> from function call to function call and are local to the function.

The variables you define inside a function (or procedure) get undefined
once the function is exited. (In IDL you (re)define a variable just by
(re)assigning a value to it.) So there is no way to have static local
variables, and:
** you have to resort to some sort of global kludge **

One way is to use common blocks. For example, you could define a common
block just for the function - you'd define and use the common block in
the function and nowhere else. (Common blocks in IDL are like the ones in
Fortran. They're really meant for providing global access to "common"
variables. Here we're using a side effect of theirs - they're static.)
This is probably the closest you'll get to local static vars.

Another way is to pass all the variables into the function. (Perhaps you'd
keep the function's "static" variables in a structure so that you only have
to pass one thing.) This would mean that you have to pass this stuff all
the way down from the main level to the function concerned. Not that good.

You might also want to check out "handles" and "widget uvalues" in the docs,
although they might be out of context here. I'm thinking that if the
function in which you need static locals is called from a widget event handler
of yours, then uvalues are actually ideal for this sort of thing. Basically,
the event handler would provide your function with a couple of widget IDs
including the ID of a "parent" widget - an ideal place to stash "static"
stuff.
(Programmers often store a structure of "state" variables in a parent
widget's uvalue. It's a very convenient method as a widget's uvalue
stays alive (and just the way you left it) as long as the widget is alive,
and the ID of the parent widget is available in "event.top", where "event" is
a scalar structure variable which is automatically passed to your event
handler. You could even store your state in one of the "child widgets" of
the parent - check out /CHILD and /SIBLING under WIDGET_INFO(), and have a
look at some of the "CW_*.PRO" programs in the IDL distribution. Handles
work similarly, except that you have to keep track of handle IDs on your
own.)
Just a last note on handles and widget uvalues:
These work similarly to C pointers (pointers to anything). The difference is
that you can't reference things indirectly - you have to actually retrieve the

stuff pointed to first, before you can work with it. There are two ways to retrieve. The standard way (no special switches) COPIES stuff from the pointer, leaving the pointer still pointing to a copy of its own; if you change the stuff then you have to put it back to "update" the pointer's copy. The efficient way involves the use of the /NO_COPY keyword - it leaves the pointer pointing to nothing, and you always have to replace the stuff when you're finished working with it.
(The "efficient" way doesn't incur wastage of memory due to extra copies of variables - something which can become an issue in IDL programs.)
I hope this all makes sense to you.

Peter Mason
