Subject: Re: Modal base without group leader.
Posted by davidf on Mon, 15 Jun 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Imanol Echave (ccaeccai@sc.ehu.es) writes:

> I want to create a modal top-level base without a group leader. I think that is
> possible because other IDL widget routines do it. DIALOG_MESSAGE is an example,
> this widget is modal and doesn't need to specify his group leader. How can I do
> it?

Presumably if you are writing C code and linking it to IDL with LINKIMAGE you can write anything you like, including making calls to system-level routines like DIALOG_MESSAGE does. But if you are like the rest of us you will have to live with certain limitations. This is one of them.

IDL 5 changed the way modal widgets were created, presumably to bring the code into compliance with accepted standards that are available across all platforms. An admirable goal in theory, but a bit of a pain in the neck in practice. In particular, to make a modal widget you used to set the MODAL keyword on XManager, but in IDL 5 you had to set it on the top-level base. BUT, to do that correctly you also had to give the top-level base a valid group leader.

This caused some panic for me because I had quite a few useful routines for doing things like selecting and reading files that I needed to make modal, but I also wanted to call these routines both from within widget programs *and* from the IDL command line, where a group leader wasn't available. Was it possible to write a program that stopped and waited for user input when called both from within a widget program and from the IDL command line?

Yes. But first I had to learn the fine distinction between a "blocking" widget and a "modal" widget. Those of you who wrote widget programs in IDL 4 are familiar with blocking widgets. These widget programs, when run, "block" the IDL command line (similar to any other IDL program). The program must be destroyed before the command line comes back. This is similar, but not the same, as a modal widget, which must also be destroyed before the user can interact with other widget programs.

To create a blocking widget today, you just call XManager without using the No_Block keyword. BUT--and this is the most important point--it is only the first

widget program registered with XManager as a blocking
widget that blocks the command line. If that blocking
widget program called other blocking widget programs,
those other blocking widget programs would NOT block!

Another way to say this is that those other widget
programs would have to be written as "modal" widgets
if you expected them to stop and wait for user input.
But, again, this is not a problem. Since these other
programs are being called from within the blocking
widget program, there is a valid group leader available:
the top-level base of the blocking program.

So, back to my problem. How should these blocking/modal
widget programs be written? Take, for example, a program
named GetImage that is available on my web page. This
is a program for requesting information from the user
about the name, size, data type, etc. of a data file
that the program opens and reads, returning the image
data to the caller of the program:

```
    image = GetImage()
```

GetImage is written as a blocking widget. That is to
say, the XManager call looks like this:

```
   XManager, 'getimage', tlb, Event_Handler='GETIMAGE_EVENT'
```

So if I call it from the IDL command (the command line
must be unblocked if I can call it) like I do above, then
it "blocks" and waits for me to fill out the proper fields
and hit the "Accept" button, which causes the program to
open the appropriate file and read and return the image data.

However, if I call GetImage from within a widget program
(I often do), then I must use a Parent (equivalent to a
Group_Leader) keyword. (I'm not too excited about this,
because it makes the Parent keyword a *required* parameter
in this case, which sort of goes against the point of
keywords. I could make it a positional parameter, of
course, but this is how I chose to implement it.)

```
   image = GetImage(Parent=event.top)
```

How is this implemented? Like this:

```
   IF N_Elements(parent) EQ 0 THEN $
     tlb = Widget_Base(Column=1, Title='Read Image Data') ELSE $
```

```
tlb = Widget_Base(Column=1, Title='Read Image Data', $
   Modal=1, Group_Leader=parent)
```

Now, notice that if I forget to use the Parent keyword and
I call GetImage from within a *non-blocking* widget program,
I have no problems at all. When the XManager call in GetImage
is executed, the program blocks. I only have problems if
I forget to use the Parent keyword and I am calling GetImage
from a *blocking* widget program. (Which I hardly ever write,
thank goodness!) In that case, without the Modal keyword
being set on the top-level base, GetImage would not block
and wait for user input. Thus, it would fail to execute
correctly.

How can I know internally if GetImage is being called from
a blocking or non-blocking widget program? I don't know.
There may be a way, but I haven't explored it. I've chosen
to document the requirements for the program and live with
a bit of uncertainty.

Sorry for the long and windy explanation. This is a widely
misunderstood topic and one that can't be explained easily
in a few sentences. There is a more complete explanation
of this topic in my book.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438
Coyote's Guide to IDL Programming: http://www.dfanning.com/
