Subject: Re: Cleanup routines
Posted by J.D. Smith on Fri, 25 Apr 1997 07:00:00 GMT
View Forum Message <> Reply to Message

William Daffer wrote:
>
> Hello all;
>
> I just wanted a confirmation on a point. Is it true that by the time
> a cleanup routine specified by the 'cleanup' keyword to xmanager is
> called, the widget is already destroyed? Therefore, you can't store
> stuff like handle ids in any info structure residing in the user value
> of the top level widget since it won't be there for your cleanup
> routine to free. Seems like one is forced to use commons.
>
> Does the 'kill_notify' keyword to widget_base work differently? Not that I
> want to use it, since it's ignored in routines managed by Xmanager.
>

Not quite true...

There was a discussion on this topic in the newsgroup about a 1 1/2 months ago. The reality is that most of your widget application is killed up to *but not including* your top level base when you get to your CLEANUP routine (and in actuality, "killed" might need to be replaced with "hidden"). This is basically implemented in XManager by setting the kill_notify function of your top level base (i.e. the id you pass to XManager) to XUnregister(), a procedure which chains to *your* CLEANUP procedure (after taking care of some registration junk). So, the CLEANUP routine *will* have access to the TLB's user value... but *only* this user value. If your state is not entirely encapsulated in this, you'll be out of luck. Fortunately, there are several ways out of this dilemma. I'll repost my discussion relating to this point from the former postings:

**********BEGIN REPOST**********

XMANAGER's CLEANUP callback mechanism gives you control of your dying widget after almost all of it has been killed. This is problematic, since you often can't then get to your state info.

A technique I often use to prevent this behaviour is the KILL_NOTIFY mechanism. Its use is strongly discouraged in various IDL references, but the thrust of the warning is "don't assign a kill_notify procedure for a top level base". As long I avoided this, I haven't had problems. The key recognition is that the callback procedure specified by KILL_NOTIFY only has access to the user value of that widget for which

it was specified -- no other widgets are gauranteed to be alive, so widget_control doesn't let you even try to access them. The specified widget can't be the top level base, since XMANAGER wouldn't like this. What is needed is a widget that contains the state info but is *not* the top level base. I therefore use the the base's first child's uvalue to store my state structure (the de facto for compound widgets), and assign the callback procedure to the *child* widget. E.g. I might say:

```
widget_control,widget_info(base,/CHILD),SET_UVALUE=state,/NO _COPY, $
  KILL_NOTIFY='WIDGET_KILL_PROCEDURE'
```

The strength of this technique lies in the fact that when the callback procedure is called, the state is still defined, since it's in the user value of the widget... so you can free handles, save info, etc. This works whether the user exits with the DONE button, or otherwise. In fact, it's also convenient for application development, since even a crashed widget (after you retall and xmanager) will call its kill_notify, freeing handles and preventing memory leakage.

Perhaps there is substance to the warnings, and problems with using XMANAGER jointly with KILL_NOTIFY do exist, but I haven't experienced them, as long as I stayed away from the TLB.

**********END REPOST**********

As a final note, it is interesting that IDL's own XManager implements a KILL_NOTIFY, but they warn you against its use. I think this is basically because they're afraid users will forget and set the TLB's KILL_NOTIFY. I now believe that if you avoid this, there are no problems whatsoever with my technique (other than possibly style). All KILL_NOTIFY's will be called, except one for the top_level base. XManager only ignores (by overriding) this particular callback.

Hope this helps,

JD