Subject: Re: Passing info and destroying widgets...
Posted by Struan Gray on Tue, 22 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Liam Gumley, Liam.Gumley@ssec.wisc.edu writes:
>
> David Fanning <davidf@dfanning.com> wrote in message
>
>> Liam Gumley (Liam.Gumley@ssec.wisc.edu) gives us an
>> example of a program that can record the last instance
>> of a button push in a non-blocking, non-modal widget
>>
>> No question it works. But I would argue that it works
>> for all the wrong reasons and is a *terrible* programming
>> practice in almost every instance.
>
> Well I guess I'll have to say that my example only
> demonstrates that it *can* be done, not that it
> necessarily *should* be done.

I've been writing generic helper widgets which behave like the tool palettes and pattern swatches found in drawing programs. Because these often manage properties that can be changed elsewhere in the application, and because they can be left floating about ready for use at any time, other widgets need to be able to get and set information about the helper widget's state.

Liam's technique works, but it is ugly (sorry Liam :) and opens an economy-sized can of worms. My concern is less that users will create memory leaks, but rather that they will come to depend on a particular info structure or tag name being present, which makes it hard for me to revise the helper widget later.

My old solution was for both the main and the helper widget to send custom events to each other (in the same way that David's colour table pickers can update draw widgets on 24-bit displays). The helper widget was defined in a function that returned its own widget ID instead of a pointer to the info structure.

This works, and is consistent with the overall widget methodology, but it also causes code-maintainence problems. I find I lose track of where the custom event structures are defined (sometimes IDL does too) and they have a habit of proliferating to an alarming extent as the functionality of the helper widget increases. Also, coding discipline is demanded to ensure that event handlers of widgets using the helper in a simplistic way don't lose or trip up on events they are not interested in.

The new (actually, THE ONE TRUE) way objectifies the widget as described by Mark Rivers. This simplifies event handling (and debugging) because the info structure is now referenced by SELF.xxxxx. It never needs to be fetched or restored so it can't go missing. The helper widget creation function returns its object ID and the other widget(s) can get and set information through methods. Standard events can be sent to the encapsulated widget by a DO_EVENT method, and as I mentioned in my earlier post the user now has a way to prioritise events, bypassing the very crude event management available in WIDGET_CONTROL. Code becomes more readable since you avoid having to write (*infoptr).xxxxx everywhere, and I find that procedures and definitions tend to end up in more logical places on my hard disk.

Why bother? Here's an example: I have a generic viewer widget for 3D model objects. The viewing angle can be set by an embedded trackball, or by choosing menu items which nudge the object a few degrees about various axes, or by setting the whole thing spinning continuously. More exact angular movements, as well as control of the spin axes and rate, are specified in a helper widget.
Objectification (reification?) makes it much easier to handle the multiple ways of setting and displaying the viewing angle and letting all the interested widgets know that changes have occurred. I can prioritise user commands like 'stop spinning' or 'reset angle' and I don't have to worry about them being lost from the event queue when I remove piled-up timer events after long redraws. Finally, I can do gee-whiz things while spinning, such as simultaneously changing background colours or even editing the structure of the model object in yet another widget.

Sorry this is a bit long, but I'm enthused.

Struan
