Subject: Re: widget_control and group_leader
Posted by John-David T. Smith on Fri, 22 Dec 2000 23:42:23 GMT
View Forum Message <> Reply to Message

nrk5@cornell.edu wrote:
>
> Lets say I have two widgets, A and B. There are two links between the
> two:
> 1) A.top and B.top are eachother's groupleaders, and
> 2) A uses common blocks and has a variable 'foreign_event_handler' that
> is set by B.
>
> So, when an event is generated by A and the 'Use Foreign Event Handler'
> option is set in the widget, events generated by A go to whatever B set
> 'foreign_event_handler' using:
>
> widget_control, id, event_pro=foreign_event_handler
>
> Things to note:
> 1) A can't be modified at all. Nothing added or changed. (ie. no more
> variables)
> 2) B is an object widget and needs to set its structure variables to
> variables in the events generated by A.
> 3) In B::init, B.top has a uvalue of self.
>
> The question is, how can I use foreign_event_handler to get to 'B self'
> from an event generated by A? My thought was:
>
> PRO foreign_event_handler, eventFromA
> widget_control, eventFromA.top, get_Group_Leader = BtopID
> widget_control, BtopID, get_Uvalue = objectReferenceToB
> ...
> END
>
> And now I would be in business. But, is there such as thing as
> get_group_leader? Is there another way to do this?
>
> I know that not being able to change A doesn't help, else there would be
> a million solutions, but its not my program. The only minor change I
> might be able to make is to create a generic variable in A's common
> block that could be set to whatever, but then I would have to define it
> as a string or a long, and that would restrict its use.

Hi Nidhi, how's the weather in Fargo? Glad to see you didn't take my advice and are hard at work. Since I know a little bit about this project, I'll explain this for everyone:

A. is a premade, heavily common-block oriented display program.

B. is an extension to A. (or more than one) which will receive events from A. (mostly widget_draw???). Rewriting A. is not time-productive.

Obviously, the first reaction out of this crowd will be "Common Blocks!!! Off with her head, accursed practice of vile witchery!" But here the goose came fully stuffed, so even if you don't like turnips, you'll have to make something of it.

The first thing to note is that there needn't be just one event handler. Sure, there is one and only one place where IDL will automatically deliver an event for you, but that event can be forwarded around anywhere you wish, and changed along the way, if you so desire. There's nothing to say a single motion event can't simultaneously display a zoomed image, update a data/coordinate status line, and stretch a colormap, all at once, even from within different entire widget trees or programs. You obviously have to be a bit careful throwing all these events around, but in practice it's no problem. This means, you never have to use:

widget_control, event_pro=foo

You can just process and dispatch events from within the already existing widget handler. This also obviates your "Foreign Event Handler" button, as this can all be automatic, and you can be using those events all over the place, whenever appropriate.

What I would recommend in this case is set up a foreign event handler *method*, since the foreign widget is an object. That is, have a routine to sign up for events from A. from within B., like this:

a_signup, self, "Handle_A_Events", /Button, /TRACKING

or some such. Then, each "foreign" object can sign up for whatever events it wants. This can obviously be static or dynamic (i.e., objects can register for certain events, and change that during runtime). All you'd need to add to A. is code to manage this "signup" list (add, delete entries -- a pointer on A.'s common block would be most flexible here), and a small function which uses:

call_method,method,obj, ev

to dispatch the event from within A's standard event handler, based on the events requested (B would turn on and turn off the event spigot when appropriate). If you'd like to make it quite simple (e.g. no need to expand it later to more than one type of foreign object widget), dispense with the optional events, and just send them all. So, at the most basic level, it's the same as having your foreign_event_handler,

but just as foreign_event_method instead (which necessitates storing an object on which to call the method).

The important point to remember is that when you are explicity redirecting events from the standard IDL up-the-widget-tree widget_event() handling, there is no benefit had by keeping the same event handler format (e.g. events are not "swallowed" or "passed on" outside the tree in which they originated). So you may as well use events however is convenient. Just to appease David I should note that you can use "widget_control, id, SEND_EVENT=event" to effectively splice two widget trees together, and royally confuse yourself.

One more wrinkle: What if you didn't want to modify A's code at all? So you could drop in new versions as they become available, for instance. All you allow yourself to do is change the event handler for A, after it sets itself up (how you get A's TLB ID is up to you). In this case, a special purpose event broker (call it C.) could sit between A and the rest of the world. It could interpose it's own procedure as the primary event handler, and feed both A., and all the B.'s. It could also serve as a proxy for A. when signing up different types of events, etc. (i.e., the B's sign up with C., not A.!)

Whatever you do, make it 1 notch more general than you think you need, and you'll thank yourself later.

JD
