Subject: Re: widget_control and group_leader
Posted by John-David T. Smith on Fri, 29 Dec 2000 18:33:38 GMT
View Forum Message <> Reply to Message

Nidhi Kalra wrote:
>
> In article <3A4652F8.D47410F8@astro.cornell.edu>,
> John-David Smith <jdsmith@astro.cornell.edu> wrote:
>
> Thanks JD. Lots of good info here. Having looked at the code, I realize
> that I'm unlikely to have more than one object widget behaving in the B
> position. So, rather than having the list revolve around objects, I
> thought it might be cute to have it use the event.id as the key.
>
> What I mean is that the first thing when you create the Broker is that
> it does a
>
> widget_control, id, event_pro = broker_event
>
> to all the widgets. Then, B registers with the broker items in the
> signup_list of the following type:
>
> item: Event_ID ;The ID to match
> Object B ;The object owning the method
> Method ;The method to be called
> Call_Before ;A boolean value indicating when to call the method
>
> So, the event handler gets changed to:
> 1. Find all items in the list such that item.Event_ID = event.id
> 2. Of these, find those where Call_Before = 1
> 3. Call each of these methods
> 4. Send the object back to where it came from (widget control, send..)
> 5. Find the remaining items where Call_Before = 0
> 6. Call each of these methods

I presume in 4 you mean send the *event* back to where it came from? This might
be more work than you need... see below.

>
> The principle is the same, the details are a bit different. I also have
> some technical questions about your code. Things I couldn't find in the
> help.
>
>>
>> pro Broker::signup, obj, method, REMOVE=rm
>> if obj_valid(obj) eq 0 then return
>> if ptr_valid(self.signup) then begin
>

> --> I couldn't find the keyword COMPLEMENT documented in the call to
> 'where.' It appears to return those items that are not in 'wh'.

Sorry, that's a 5.4ism, and your guess is correct. In older days I would have used:

```
wh=where((*self.signup).object ne obj,cnt)

if cnt lt n_elements(*self.signup) then ...
  ....
  (*self.signup)=(*self.signup)[wh]
```

before adding the new one on.

>
>> wh=where((*self.signup).object eq obj,cnt,COMPLEMENT=valid)
>>
>> ;; Rid list of obj, if it's already on there
>> if cnt ne 0 then begin
>
> -->Assuming I'm right about what 'valid' is, does valid[0] = -1 if there
> are no items in the list that arent in 'wh'?

You are correct. The equivalent test is cnt eq 0 vs. cnt eq n_elements(*self.signup). I.e. none of them, or all of them. If none remain valid, we free the list.

>
>> if valid[0] eq -1 then ptr_free, self.signup $
>
> -->I am not sure what (*self.signup)[valid] does. Reissue self.signup to
> be valid? [valid] ?

It's just an array indexing of the dereferenced list pointed to by self.signup. I.e. self.signup is a pointer to a 1-d array of {BROKER_SIGNUP} structs. *self.signup is that list. (*self.signup)[0] is the first element of that list. Etc.

Precedence rules with pointer dereference in IDL are goofey (or rather, C rules are goofey but most people are used to them).

> -->Why does list_item have 'BROKER_SIGNUP'? What does that do/why is it
> there?

That is a named structure, which is defined in the class definition, along with the class itself. It doesn't actually serve to define the class in any way, but is an auxiliary helper structure. The reason to use named structures is the ability to concatenate them together (vs. anonymous structures with the same

fields/data sizes). Hence managing the list is easier.

>> Just because RSI publishes a manual describing standard event
>> processing doesn't mean you can't innovate beyond that (especially in
>> unusual cases like yours).
>
> True. Sometimes you just cant follow the herd. Moooo.

One more piece of advice. If you're sure there's only one B, you don't need a list at all! In fact, you don't even need a Broker! You just make B itself intercept the events, like (extra bits ommitted):

```
pro B_Event, ev
  widget_control, ev, get_uvalue=self
  self->Event, ev
end

pro B::Event, ev
  ;; Process the event as it relates to B

  ;; Also Send back to A
  widget_control, self.A_ID, EVENT_PRO=self.A_EVENT,SEND_EVENT=ev
  widget_control, self.A_ID, EVENT_PRO='B_Event'
  ;; or use:
  ; call_procedure, self.A_EVENT, ev
end

function B::Init, A_ID
  self.A_ID=A_ID
  self.A_EVENT=widget_info(A_ID,/EVENT_PRO)
  widget_control,self.A_ID, SET_UVALUE=self, EVENT_PRO='B_Event'
  return, 1
end

pro B__Define
  struct={B, A_ID:0L, A_EVENT:''}
end
```

You get the idea. Much simpler. Less flexible, but easier to code. Simply intercept the event. Process it locally for B's own devious purposes, and forward it on to A. By the way... there was an error in my prior logic. Sending an event to A via SEND_EVENT will bring it right back to the Broker, making an endless event loop, unless you temporarily reset the event handler. That's shown here. Also you really needn't use SEND_EVENT, which normally would use the swallow vs. non-swallow event tree paradigm. In this case, since all event_pro's swallow events (no return value), it's exactly equivalent (and possibly faster?) just to say:

call_procedure, self.A_EVENT, ev

to send them back to A. How's that for simple, David?

JD
