Subject: Re: widget_control and group_leader
Posted by nrk5 on Mon, 25 Dec 2000 05:18:23 GMT
View Forum Message <> Reply to Message

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

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'.

```
>       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'?

```
>         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] ?

```
>         else *self.signup=(*self.signup)[valid]
>      endif
>    endif
>
>    ;; Add it to the list, if necessary
>    if keyword_set(rm) then return
>
```

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

```
>    list_item={BROKER_SIGNUP,Object:obj,Method:Method}
>    if ptr_valid(self.signup) then begin ; append item
>      *self.signup=[*self.signup, list_item]
>    endif else $                ;create list with item
>      self.signup=ptr_new(list_item,/NO_COPY)
>  end
>
>  pro broker_handler, ev
>    widget_control,ev.top, get_uvalue=self
>    self->Handler
>  end
>
>  pro Broker::Handler, ev
>    ;; Send it to A
>    widget_control, self.A_ID, SEND_EVENT=ev
>
>    ;; Send it to all the B's
>    if ptr_valid(self.signup) eq 0 then return
>
>    for i=0,n_elements(*self.signup)-1 do begin
>      call_method, (*self.signup)[i].Method, $
>                (*self.signup)[i].Object, ev
>    endfor
>  end
```

```
>
>  pro Broker::Cleanup
>    ptr_free,self.signup
>  end
>
>  pro Broker::Init, A_ID
>    ;; We will pre-process A's events
>    widget_control,A_ID, event_pro= 'c_handler'
>  end
>
>  pro Broker__Define
>    struct={Broker,A_ID: A_ID, signup:ptr_new(), ...}
>
>    ;; A convenience struct for the signup list
```

-->Again, you use the term 'Broker_Signup'. What is that?

```
>    list_struct={BROKER_SIGNUP, object:obj_new(), method:''}
>  end
>
```

> This is just an outline, and I haven't tried it. But it gives you an
idea of
> what I had in mind. You can see how I caught A's events before it
does, and
> then send them on to A (via the standard IDL event flow), and also to
the B's
> (via the method/object they signed up for). You could obviously add
more
> intelligence to this dispatch process (e.g. only button events to B1,
etc.)
>

>
> So, the one thing I didn't specify is when the B's signup for events.
 I.e. how
> do they know they are on? Two possibilities:
>
> 1. They are always on, i.e. you start them from the command line, and
they
> immediately sign up:
>

```
>  a=A_widget(lots_of_args)
>  c=obj_new('Broker',a)
>  b=obj_new('Cool_foreign_helper',BROKER=c)
```

This seems like a good choice. I really dont want to mess with A.

>

> and in b's Init:
>
> function Cool_foreign_helper::Init, BROKER=brk
>   brk->Signup, self, 'MyHandler'
>   return, 1
> end
>
> 2. They get turned on by A (which means you'd have to modify A to at
least have
> this ability).

> 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.

Thanks much :)

Nidhi

--

Nidhi Kalra
nrk5@cornell.edu

Sent via Deja.com
http://www.deja.com/