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

Subject: Re: widget_control and group_leader
Posted by Richard French on Sat, 23 Dec 2000 14:26:32 GMT
View Forum Message <> Reply to Message

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

This is a piece of advice I've been trying to follow for the past few years, and it really came back to help me this past week, when a research
colleague came to visit for four days of hard work. We were able to process 75 Gbytes of radar observations of Saturn's rings, using very slight modifications of the code we wrote for last year's data set, even though this year's observations had very different formats.
In four days, we don't have time to write fancy widget programs, but we use IDL the old-fashioned way - direct graphics and no objects.

Still, last year's effort in taking the small amount of extra time
to avoid hard-wiring in numerical constants that 'couldn't possibly
change', and keeping all of the user-defined variables near the very
top of the code, made it much easier to adapt the programs to our
unexpectedly different circumstances this year.
 I mention this only because I used to approach IDL as though
the I in Interactive meant 'interactive programming' - I'd start
a journal file, fiddle with the observations and analysis and display,
edit the journal file, and call it a program. I still take this approach
for rush projects, but taking the few minutes to annotate the code
and reorganize it so that it can be used again is now a high priority
for me.
 Although I have written some widget programs over the years,
I still find myself quite often using IDL in this seat-of-the-pants
mode when I start a new project. I'd be curious to know how many
other readers of this news group use IDL primarily in this way.
Dick French

Subject: Re: widget_control and group_leader
Posted by Pavel A. Romashkin on Sat, 23 Dec 2000 17:32:34 GMT
View Forum Message <> Reply to Message

Hi Nidhi,
I guess, the use of a common block is justified in this case.
What it boils down to, A does not need to be aware of B. It is the *
foreign_event_handler* that does. So, create a new common block in
B::init with one variable, SELF (or BtopID so that ou can use widget
control on it easily). Then let the foreign_event_handler access that
new common block and gain access to B from on an event received from A.
Trying to be elegant by avoiding common blocks is ok I guess, but then
again, if you think about it, Xmanager itself uses them, so why should
we be ashamed of it?
If you insist on not following "the evil path of Commons", and can
inspect the code for A, try to see if AtopID.uvalue or
AtopID.child.uvalue are not being used. Then, let B "worm" into A
without modifying the code of A at all, at the same time you do
Widget_control, AtopID, group_leader=BtopID. Just set the Uvalue of
AtopID or AtopID.child to BtopID and you will be able to gt it from
A-generated event.
Now, the last and the cleanest, I'd think, would be, at the time you do
widget_control, BtopID, group_leader=AtopID, is to create your own,
useless for A, *named* widget like this:

```
link_key = widget_base(AtopID, uname='Link_base_for_B', $
 map=0, xsize=1, ysize=1, uvalue=BtopID)
```

Then, from any event generated by A, you can call (although searching

for a widget is not my favorite way of programming, it can be justified
by those who hate common blocks)

link_key = widget_info(event.top, $
 find_by_name='Link_base_for_B')

and use link_key to get to B.self.

Hope this helps.
Cheers,
Pavel

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.
>
> Thanks much,
>
> -- Nidhi
> ----------
> Nidhi Kalra
> nrk5@cornell.edu
>
> Sent via Deja.com
> http://www.deja.com/

Subject: Re: widget_control and group_leader
Posted by nrk5 on Sun, 24 Dec 2000 07:34:07 GMT
View Forum Message <> Reply to Message

In article <3A43E6DF.98E40A55@astro.cornell.edu>,
JD Smith <jdsmith@astro.cornell.edu> wrote:

>
> Hi Nidhi, how's the weather in Fargo? Glad to see you didn't take my
> advice and are hard at work.

Such is the life of a lowly college student. Besides, I live in Fargo.
What else can I do? :)

Thanks for explaining the project, btw.

> 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
>

Let me paste in a bit of the code from program A. In the event handler
that I am mostly concerned with, the user sets the mode. mousemode
cases 0-3 were already there, so I added 4 for uniformity. When 4 is
selected, events on the draw_widget are sent to the foreign event
handler.

pro a_event, event

```
; Main event loop for atv top-level base, and for all the buttons.
...
widget_control, event.id, get_uvalue = uvalue
...
case uvalue of
   'mode': case event.index of
      0: widget_control, state.draw_widget_id, $
        event_pro = 'atv_draw_color_event'
      1: widget_control, state.draw_widget_id, $
         event_pro = 'atv_draw_zoom_event'
      2: widget_control, state.draw_widget_id, $
         event_pro = 'atv_draw_blink_event'
      3: widget_control, state.draw_widget_id, $
         event_pro = 'atv_draw_phot_event'
      4: widget_control, state.draw_widget_id, $
        event_pro = state.foreign_event_handler $
        + '_event'
      else: print, 'Unknown mouse mode!'
   endcase
```

> 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.

The functionality I'm going for is that the user can decide when to use external event handlers and when to let program A run 'naturally'. At the moment, I have tried to keep foreign_event as general as possible. Each B can do whatever it pleases with its own particular foreign_event handler. The two things (now) registered with A are the foregn event handler to use and a widget_ID to use. Whatever that needs to be.

> 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.

The object method part makes sense, but "signing up" is a bit confusing.

> 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).

How do I register event/object pairs? Ok. So here I'm a little lost
(Caution: Newbie IDL-er at work). A
registers the following event handlers:

widget_control, top_menu, event_pro = 'topmenu_event'
widget_control, state.draw_widget_id, event_pro = 'draw_color_event'
widget_control, state.draw_base_id, event_pro = 'draw_base_event'
widget_control, state.keyboard_text_id, event_pro = 'keyboard_event'
widget_control, state.pan_widget_id, event_pro = 'pan_event'

And everything in these main bases is differentiated by uvalues (as you
can see from the above code). So I'm a bit confused about how to go
about differentiating the "events requested" and how the reigstering
in "call_method,method,obj, ev" works.

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
>

> 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.

Thats good advice, and its prettymuch why I am being so fussy about this right now. The requirements Jim has given me really dont require much, but I would really hate to have to rewrite everything (or anything, for that matter) later.

So, ideally, here's the functionality im looking for. On "foreign" mode, all events go to foreign_event_handler. If foreign event handler wants to do something with it, wonderful. If not, the event goes back to where it would go on non-foreign mode.

The quick and dirty way is to put in a simple statement in each of the four event handlers:

if (foreign) send_event, foreign_event_handler, event (or whatever).

hmm...waitaminit. what if i register foreign_event_handler as the event handler for the top level base? what would that do? Would all events then go to foreign_event_handler and then bubble up/down?

I think I'll need to sleep on this.

Thanks much

>
> JD
>

--

Nidhi Kalra
nrk5@cornell.edu

--

Nidhi Kalra
nrk5@cornell.edu

Subject: I for Interactive Programming? (was: widget_control and group_leader)
Posted by Jaco van Gorkom on Wed, 03 Jan 2001 11:13:00 GMT
View Forum Message <> Reply to Message

Richard G. French wrote:
> ... I used to approach IDL as though
> the I in Interactive meant 'interactive programming' - I'd start
> a journal file, fiddle with the observations and analysis and display,
> edit the journal file, and call it a program. I still take this approach
> for rush projects, but taking the few minutes to annotate the code
> and reorganize it so that it can be used again is now a high priority
> for me.
> Although I have written some widget programs over the years,
> I still find myself quite often using IDL in this seat-of-the-pants
> mode when I start a new project. I'd be curious to know how many
> other readers of this news group use IDL primarily in this way.
> Dick French

Hi Dick,

Count me in, I do this all the time. Usually forgetting to start a
journal
in time and thus cutting and pasting from the log window. I find that it
often leads to faster and more 'creative' data analysis, although it is
sometimes hard to really take those minutes at the end of the day for
reorganizing and annotating the code.

Jaco

Jaco van Gorkom gorkom@rijnh.nl
FOM-Instituut voor Plasmafysica Rijnhuizen