Subject: Re: Communication between top-level bases.
Posted by David Foster on Mon, 01 Jun 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Imanol Echave wrote:
>
> Hi people:
>
>   I have a widget program with a top-level base which is the group leader of some
> other top-level bases. The events produced in the "child" top-level bases must
> be communicated to the "parent" top-level base. Do you know an "elegant" way to
> do this?

What you mean by "communicated" is a bit vague, but here's my input.

First, if you want the events to be handled by the event handler
for your parent widget, just use the EVENT_HANDLER keyword in
the XMANAGER calls for your child TLB's, eg.:

```
  xmanager, widget_name, widget_id, $
          event_handler=parent_event_handler, $
          group_leader=state.base, /no_block
```

where

  WIDGET_NAME is name of child TLB widget
  WIDGET_ID is widget ID of child TLB
  PARENT_EVENT_HANDLER is name of event handler routine defined for
 parent TLB widget
  STATE.BASE is parent widget ID stored in STATE structure (see below)

In this way, whenever you register a new widget with XMANAGER you
can control which event handler processes the widget's events.

However, it is generally considered better programming style to
have the events for each widget handled by their own event handler.

If by "communicate" you mean that you want data shared between
the widgets, this is a more complicated issue. The first solution
which seems almost inevitable for beginning programmers is to use
common blocks. This is a very easy way to accomplish this sharing
of data, but I recommend that it be avoided.

It would be well worth your efforts to learn to do this the "right"
way from the beginning, so your programs don't suffer from the
shortcomings of common blocks (one being that only one copy of the
program may be running at any one time).

A much more elegant way to share data *within* a widget is to create a STATE structure (I always call it STATE{} to avoid confusion) which contains the data you want to share. Then, before you register it with the XMANAGER, copy this structure into the UVALUE of the top-level widget:

```
 widget_control, base, set_uvalue=state, /no_copy

 xmanager, 'program_name', base, /no_block
```

For reasons that you'll see below, be sure to include the field BASE which is the widget ID for the TLB.

If you want to share this structure with other child top-level modal widgets that are created within the first "parent" program, you can do the following:

1. When you create a new child TLB widget, set the UVALUE of the TLB's first child to be the widget ID of the parent TLB.

2. When you process an event from one of these child TLB's, get the UVALUE of EVENT.TOP's first child; this will be the widget ID of the parent TLB. Then get the STATE information from *that* widget's UVALUE. So at the beginning of the event handler for the child TLB, put:

```
 stash = widget_info(event.top, /CHILD)
 widget_control, stash, get_uvalue=main_base
 widget_control, main_base, get_uvalue=state, /no_copy
```

This technique is intended for modal child widgets only. You would have to make some minor changes if you wanted to generalize for non-modal child widgets.

Here are some guidelines to follow:

1. Whenever you copy the STATE information within an event handler from a UVALUE, always use the /NO_COPY keyword to make the operation faster (otherwise your program may slow down a *lot*).

2. Always be sure you put STATE *back* into the UVALUE at the end of your event handling routines, since the /NO_COPY keyword makes the UVALUE undefined.

3. Since all widgets will be sharing the same STATE structure, be careful to update the TLB's UVALUE whenever STATE has been updated. Also, be sure that the copy of STATE that you pass

back to the main event handler is the updated one. In
particular...

Be careful about situations in which you call a function to
create a child TLB widget, and you pass STATE to this function.
After you call XMANAGER in this function, you should copy
the UVALUE of the main TLB back into STATE. This is because
the STATE structure was probably updated by the created widget,
and will contain information that is "new" with respect to the
local copy of STATE known to the function. Otherwise, when this
function returns, it would return the local copy, and not the
updated copy stored in the TLB's UVALUE.

Basically, here is how I would write such a function to create
a new child TLB widget:

```
FUNCTION create_widget, state

state.child_base = WIDGET_BASE(group_leader=state.base, /modal)

label = WIDGET_LABEL(state.child_base, value='')  ; For TLB ID

<create the rest of the widget heirarchy>

WIDGET_CONTROL, state.child_base, /realize

; Put Main TLB ID into first child of this popup widget, and
;  update STATE info in uvalue of main TLB

widget_control, label, set_uvalue=state.base
widget_control, state.base, set_uvalue=state

; Call XMANAGER to start the widget. Events from this
;  widget will cause STATE to be updated in the UVALUE
;  of the parent's TLB. Since this is a modal widget,
;  processing continues once the widget is destroyed.

xmanager, 'child_widget_name', state.child_base, $
      event_handler='child_event', group_leader=state.base

; Now get STATE info back from TLB uvalue, so when passed
;  back it doesn't overwrite what we've just updated in the
;  CHILD widget.

widget_control, state.base, get_uvalue=state

return, 0
END
```

Notice that in this function when I copy STATE from and to
the UVALUE, I don't use /NO_COPY. This is ok, since these
operations only happen when the widget is created, and don't
slow things down much. It's a different story however if you
do this in an event handler!

This stuff can get a bit tricky. I strongly suggest you get
David Fanning's "IDL Programming Techniques" book. You can order
it from his web site: http://dfanning.com (I'm not trying to sell
his book...I just find it very useful).

If what I've written doesn't make sense, please feel free to email
me and I'll try to explain it more completely.

I hope this helps.

Dave
--

```
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~
  David S. Foster         Univ. of California, San Diego
   Programmer/Analyst     Brain Image Analysis Laboratory
   foster@bial1.ucsd.edu  Department of Psychiatry
   (619) 622-5892         8950 Via La Jolla Drive, Suite 2240
                          La Jolla, CA  92037
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~
```