Subject: Communication between top-level bases.
Posted by Imanol Echave on Fri, 29 May 1998 07:00:00 GMT
View Forum Message <> Reply to Message

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?

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

David Foster (foster@bial1.ucsd.edu) writes a lot of excellent advice about widget programming, including this:

```
>  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
```

I couldn't have written this part of the article any better myself, and (as Dave mentions) I suggest something very similar in my IDL Programming Techniques book. But I developed those ideas nearly two years ago now, and I find myself using something quite a bit different today. Poor Dick, who knows I am working on ideas for the next programming book, calls me about every other day to see what our Current Best Practice is now. When I am in the writing mode, it changes frequently. :-)

But what I do today is put the state structure (I often call it the "info" structure) in an IDL pointer variable, like this:

```
   statePtr = Ptr_New( {image:image, drawID:drawID, ...}, /No_Copy )
```

By placing the pointer in the UValue of the top-level base (TLB), or by passing the pointer to other widget programs, I can assure that any program module that has the pointer has the latest and greatest information that the pointer points to.
The only tricky part is de-referencing the pointer. My event handler code might look like this:

```
PRO JUNK_EVENT, event
Widget_Control, event.top, Get_UValue=statePtr
Widget_Control, (*statePtr).drawID, Get_Value=windex
WSet, windex
TV, (*statePtr).image
END
```

The pointer has to be de-referenced to a structure by the
use of parentheses before I can de-reference a field in the
structure.

If my image field were itself a pointer (it usually is) defined
like this:

```
statePtr = Ptr_New( {image:Ptr_New(image), drawID:drawID, ...} )
```

Then the image de-referencing would look like this:

```
TV, *((*statePtr).image)
```

It makes my IDL code look more and more like C code, but
it's not so bad when you get used to it. :-)

Using a pointer for the state or info structure has several
advantages. First, I've eliminated all the No_Copy keywords
in getting them into an out of the event handler. Second,
I don't have to worry about putting them "back" in the UValue
of the TLB, since if I make changes to the thing pointed to,
I change the thing itself. Third, I don't have to worry about
whether the info structure is "checked in" when I call another
event handler directly from within my event handler code. This
gives me more flexibility in how I handle events.

The only thing I have to remember to do is destroy the pointer
when the widget program dies. I always do this with a widget
cleanup routine, assigned with the CLEANUP keyword on the
XMANAGER call:

```
XManager, 'junk', tlb, Cleanup='JUNK_CLEANUP'
```

The cleanup routine is where all the pointers, objects,
pixmaps, etc. are cleaned up in my program. The cleanup
routine is called when the TLB is destroyed. This simple
cleanup routine might look like this:

```
PRO JUNK_CLEANUP, tlb
Widget_Control, tlb, Get_UValue=statePtr
```

```
   Ptr_Free, (*statePtr).image
   Ptr_Free, statePtr
   END
```

If I ever find the time to write the 2nd Edition of my
book, this is how I'll explain things then. :-)

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438
Coyote's Guide to IDL Programming: http://www.dfanning.com/

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
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~
```

Subject: Re: Communication between top-level bases.
Posted by Struan Gray on Tue, 02 Jun 1998 07:00:00 GMT
View Forum Message <> Reply to Message

mirko_vukovic@notes.mrc.sony.com writes:

  [widget state as object in uvalue]

> But does anyone else see any other practical advantages?

    Make it a 'state' object within a container. That way external
users who want to store information in the user value can do so too.
They have to learn your programming convention for getting and setting
info in the container object but that's better than forbidding them to
touch the uvalue because you've monopolised it.

    Pre objects I used to make the uvalue a pointer (or handle) to the
start of a linked list for just this reason. I'm swapping to objects
but haven't settled on a definite plan as yet. Of course, another big
benefit of using heap variables of any kind is that they are globally
available if you can find some way of passing the variable reference
around. It's a nice way to keep track of display preferences for
example.

Struan

Subject: Re: Communication between top-level bases.
Posted by mirko_vukovic on Tue, 02 Jun 1998 07:00:00 GMT
View Forum Message <> Reply to Message

In article <MPG.fdce647de8322e89897c4@news.frii.com>,
  davidf@dfanning.com (David Fanning) wrote:
>
> David Foster (foster@bial1.ucsd.edu) writes a lot of excellent
> advice about widget programming, including this:
>
>> 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
>
> I couldn't have written this part of the article any better
> myself, and (as Dave mentions) I suggest something very similar

> in my IDL Programming Techniques book. But I developed those
> ideas nearly two years ago now, and I find myself using
> something quite a bit different today. Poor Dick, who knows
> I am working on ideas for the next programming book, calls
> me about every other day to see what our Current Best
> Practice is now. When I am in the writing mode, it
> changes frequently. :-)
>
> But what I do today is put the state structure (I often call
> it the "info" structure) in an IDL pointer variable, like this:
>
>   statePtr = Ptr_New({image:image, drawID:drawID, ...}, /No_Copy)
>

:-)
I couldn't have written Dave's contribution any better (now does that sound familiar or what), as I dumped the State into Heap, oh say last Friday. As Dave pointed out, using /no_copy is an invitation for programming lapses.
:-)

I played for a minute or two (most of those due to editing and un-doing the edits, the other couple of secons on thinking), about putting State (which I call something else) into an *object*. A jump from a structure to an object is a small one.

This again would involve cleanup upon exit, but one benefit is a cleaner pointer de-referencing syntax in the code. It would be able to provide for some
processing, very related to the base widget, but for which I do not see any need right now (I am only doing my second widget application to-date).

However, the whole concept of the State Object is eerily similar to that of Plot Objects that David has been aluding (sp?) to.   It is essentially an object related to the widget.

I do not see any huge (other than syntactical) advantages as yet, and thus am not implementing it.
But does anyone else see any other practical advantages?

cheers,

Mirko