Subject: Passing info and destroying widgets...
Posted by dirk on Mon, 21 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Here's something i can't figure out... I'm following the advice of DF and communicating between my widgets with info structures. All is well, but now i want to pass the info structure from the event handler back to the parent widget with

WIDGET_CONTROL, event.top, SET_UVALUE=lines, /NO_COPY

and then destroy the widget. But you can't do this, because WIDGET_CONTROL (i think) dereferences event.top so that

WIDGET_CONTROL, event.top, /DESTROY

fails since it doesn't know where to look. (you can't even put in a dummy to hold the event.top number, the widget itself is gone from that id)

Unfortunately, you can't /DESTROY the top widget first and expect to set it's UVALUE later, either. So what do i do here? I tried putting a flag in my info structure to trigger the base widget destruction back in the widget definition level (not in the event handler), but i can't figure out when the program would be able to look at that newly inserted flag.

Thanks for your help. - Dirk

Subject: Re: Passing info and destroying widgets...
Posted by Liam Gumley on Mon, 21 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning <davidf@dfanning.com> wrote in message
news:MPG.11d85d6ea2cc37d9897de@news.frii.com...
> Liam Gumley (Liam.Gumley@ssec.wisc.edu) gives us an
> example of a program that can record the last instance
> of a button push in a non-blocking, non-modal widget
> when he writes:
>> Here's an example which works in non-blocking mode:
>
> No question it works. But I would argue that it works
> for all the wrong reasons and is a *terrible* programming
> practice in almost every instance. I mean, you can write
> an object method that returns a data pointer too, but
> by doing so you violate every tenet of good object programming
> practice, in which the data should be encapsulated and
> unseen by the outside world. Sucking the pointer out of
> a widget program, except perhaps in the hands of just the

> best programmers, is a practice that is guaranteed, it
> seems to me, to get most of the rest of us in a hell of
> a lot of trouble.

Well I guess I'll have to say that my example only demonstrates that it *can* be done, not that it necessarily *should* be done.

> If you are going to recommend this, at the very least
> teach people how to use HEAP_GC at the same time because
> I'll bet a ton of money there will be leaking memory
> right and left!

As the saying goes, there was good news and bad news in the release of IDL5.0:

The good news was, it included pointers.

The bad news was, it included pointers.

My personal programming philosophy is to only use pointers where absolutely necessary (e.g. for retaining information when a widget dies, or for storing structure elements which are of unknown size and type until runtime). I've never used HEAP_GC in any of my programs; I rely on simple wrappers like those I posted at http://cimss.ssec.wisc.edu/~gumley/pointers.html to prevent me from getting into memory leakage problems.

> As for me, I'm sticking to widget programs that clean
> themselves up and don't leave the user holding the bag,
> er, pointer. :-)

I agree that widget *users* (e.g. of David's fine XCOLORS routine) don't want to know (and shouldn't be allowed to touch) the internals of "shrink-wrapped" widget routines. But I think any IDL application developer will eventually run across cases where items like information structures need to be shared between widgets which are performing closely related application-specific tasks.

Cheers,
Liam.

Subject: Re: Passing info and destroying widgets...
Posted by R.Bauer on Tue, 22 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Liam Gumley wrote:

> "Robert S. Mallozzi" wrote:

>> I believe you must use XMANAGER in blocking mode for
>> this technique to work.
>
> Here's an example which works in non-blocking mode:
>
> ;---cut here---
> PRO TEST_EVENT, EVENT
>
> ;- Get pointer from top level base, then the info structure
>
> widget_control, event.top, get_uvalue=ptr
> info = *ptr
>
> ;- Handle the widget which caused this event
>
> widget_control, event.id, get_uvalue=name
> case 1 of
>   name eq 'Button 1' or name eq 'Button 2' : info.name = name
>   else : widget_control, event.top, /destroy
> endcase
>
> ;- Update the info structure
>
> *ptr = info
>
> END
>
>

I aggree to this method because sometimes I have to handle a lot of MBytes of data.
But in difference to your solution I am using a string in the tagname to identify pointers. This gives me the possibility
to destroy the pointers before I destroy the widget.
I have posted the routine get_tagname as attachement.

```
free_ptr=get_tagname(map,'PTR*')
FOR i=0,N_TAGS(free_ptr)-1 DO PTR_FREE,free_ptr.(i)

WIDGET_CONTROL,map.base_0,/destroy
```

The number of pointers are not unlimeted, so they should carefully be destroyed after usage. If you don't free pointers you are caught sometimes later not knowing which to destroy.

Another important difference is that I don't copy the pointer into a normal

variable. All changes are done directly on the pointer.
e.g.:

(*map.ptr_drawids)[i]=drawid

R.Bauer

```
;
; Copyright (c) 1998, Forschungszentrum Juelich GmbH ICG-1
; All rights reserved.
; Unauthorized reproduction prohibited.
; This software may be used, copied, or redistributed as long as it is not
; sold and this copyright notice is reproduced on each copy made.  This
; routine is provided as is without any express or implied warranties
; whatsoever.
;
;+
; NAME:
; get_tagname
;
; PURPOSE:
;   This function generates a new structure from a given structure by a search string
;
;
; CATEGORY:
;   PROG_TOOLS/STRUCTURES
;
; CALLING SEQUENCE:
;    Result=get_tagname(structure,search_string,[without=without] )
;
;  INPUTS:
;   structure:   The structure which should be scaned
;   search_string:      The string which should be used for searching in the structure
;
;  OPTIONAL INPUTS:
;   without:     A string which should not included in the result
;
; OUTPUTS:
;   The return value is a structure
```

```
;   defined by the search parameter
;
; EXAMPLE:
;   Result=get_tagname(inhalt,'pi*')
;   gives back all tags starting with pi
;
;   ** Structure <11867f8>, 1 tags, length=8, refs=1:
;      PI_NAME         STRING    'Peter Mustermann'
;
;   Result=get_tagname(inhalt,'*name')
;   gives back all tags ending with name
;
;   ** Structure <1186108>, 4 tags, length=32, refs=1:
;      PI_NAME         STRING    'Peter Mustermann'
;      NAME            STRING    'sin0'
;      PARAM_LONG_NAME STRING    'SIN(X)'
;      TIME_LONG_NAME  STRING    'time'
;
;   Result=get_tagname(inhalt,'*name*')
;   gives back all tags including name
;
;   ** Structure <1171ce8>, 5 tags, length=112, refs=1:
;      PI_NAME         STRING    'Peter Mustermann'
;      GPARAM_NAMELIST STRING    Array[10]
;      NAME            STRING    'sin0'
;      PARAM_LONG_NAME STRING    'SIN(X)'
;      TIME_LONG_NAME  STRING    'time'
;
;    Result=get_tagname(inhalt,'*name*',without='gparam_namelist' )
;   gives back all tags including name without gparam_namelist
;
;   ** Structure <1180108>, 4 tags, length=32, refs=1:
;      PI_NAME         STRING    'Peter Mustermann'
;      NAME            STRING    'sin0'
;      PARAM_LONG_NAME STRING    'SIN(X)'
;      TIME_LONG_NAME  STRING    'time'
;
;   Result=get_tagname(inhalt,'name')
;   gives back the tag name named 'name'
;
;   ** Structure <118ab68>, 1 tags, length=8, refs=1:
;      NAME            STRING    'sin0'
;
;
; MODIFICATION HISTORY:
;  Written by: R.Bauer (ICG-1), 1998-07-30
;-
```

```
FUNCTION get_tagname, struct,search_string,$
  without     = without


  IF N_PARAMS() LT 2  THEN BEGIN
    MESSAGE,'result=get_tagname(struct,search_string)',/info
    RETURN,' '
  ENDIF

  n_search_string = N_ELEMENTS(search_string)-1
  FOR o=0,n_search_string  DO BEGIN
    search = (search_string[o])(0)


    IF STRPOS(search,'*') EQ -1 THEN BEGIN
      only=1
      begining=1 ; removed ;nn
      tag_such=search
    ENDIF


    IF STRPOS(search,'*') GT 0 THEN BEGIN
      begining=1
      only=0
      tag_such=STRMID(search,0,STRPOS(search,'*'))
    ENDIF

    IF STRPOS(search,'*') EQ 0 THEN BEGIN
      begining=0
      only=0
      IF STRPOS(reverse_text(search),'*') NE 0 THEN BEGIN
        tag_such=STRMID(search,1,STRLEN(search)-1)
        ende=1
        ENDIF ELSE BEGIN
        ende=0
        tag_such=STRMID(search,1,STRLEN(search)-2)
      ENDELSE
    ENDIF
    CATCH,errvar

;if errvar ne 0 then goto , help

    tag_such=STRUPCASE(tag_such)

    tags=TAG_NAMES(struct)
```

```
    tcount=   STRPOS(tags, tag_such)
    count=WHERE(tcount NE -1,m)

    IF m EQ 0 THEN RETURN,-1

    IF KEYWORD_SET(begining) THEN BEGIN
      count =  WHERE(tcount EQ 0,m)
      IF m EQ 0 THEN RETURN,-1
    ENDIF

    IF KEYWORD_SET(only) THEN BEGIN
      a=WHERE(tags(count) NE tag_such,zaehler)
      IF zaehler GT 0 THEN BEGIN
        count(a)=-1
        mcount=WHERE(count NE -1,m)
        IF m EQ 0 THEN RETURN,-1
        count=count(mcount)
      ENDIF
    ENDIF

    IF KEYWORD_SET(ende) THEN BEGIN
       en=WHERE(STRPOS(reverse_string(tags[count]),reverse_string(t ag_such)) EQ
0,count_en)
      IF count_en GT 0 THEN count=count[en] ELSE RETURN,-1

    ENDIF

    IF N_ELEMENTS(without) GT 0 THEN BEGIN
      without=STRUPCASE(without)
      a=WHERE(tags(count) EQ without,zaehler)
      IF zaehler GT 0 THEN BEGIN
        count(a)=-1
        mcount=WHERE(count NE -1,m)
        count=count(mcount)
      ENDIF
    ENDIF

    IF m GT 0 THEN BEGIN

      keyws=tags(count)

      FOR i=0 , N_ELEMENTS(count)-1 DO $
          IF N_ELEMENTS(antwort) EQ 0 THEN
antwort=CREATE_STRUCT(keyws[i],struct.(count[i])) ELSE $
          antwort=CREATE_STRUCT(antwort,keyws(i),struct.(count[i]))
    ENDIF
```

```
  ENDFOR
  IF N_ELEMENTS(antwort) EQ 0 THEN BEGIN
    MESSAGE,'nothing found like '+tag_such,/cont
    RETURN,antwort
  ENDIF
  RETURN, antwort


END
```

File Attachments

1) get_tagname.pro, downloaded 90 times

Subject: Re: Passing info and destroying widgets...
Posted by Martin Schultz on Tue, 22 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Stein Vidar Hagfors Haugan wrote:
>
> [A very interesting article in a very interesting thread ...]
>
> The memory inefficiency is of course related to the fact that if
> you're not using pointers, you need to provide a copy of the data
> to the outside whenever the outside (user) needs to look at it,
> like this:
>
>   IDL> data = obj->getdata()
>   IDL> print,sigma(data)
>

Hmmm. To avoid that whole /NOCOPY thing, I would probably
store the data as a pointer within the object, so
   obj->getdata()
would return the pointer anyway. No need to call
   obj->setdata(pointer)
afterwards if you are not manipulating the data itself. Here, of course
one probably has to differentiate between Stein's three user types
again: don't trust a scientist! He/she is most likely to manipulate the
data and not tell the object about it. Hence, there should probably be a
   obj->CheckIntegrity
method which retrieves type and dimensions of the data
before the objects accesses it (for plotting or whatever else).
Pun: you shouldn't call this method obj->IsInteger ;-)

Martin.

--

|||||||||||||\\\\\\\\\\\\\-------------------//////////// //|||||||||||||
Martin Schultz, DEAS, Harvard University, 29 Oxford St., Pierce 109,
Cambridge, MA 02138 phone (617) 496 8318 fax (617) 495 4551
e-mail mgs@io.harvard.edu web http://www-as/people/staff/mgs/

Subject: Re: Passing info and destroying widgets...
Posted by Liam Gumley on Tue, 22 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Stein Vidar Hagfors Haugan wrote:
[many thoughtful comments removed]
> I must say I agree a lot more with Liam than David here.

That makes it all worthwhile!

I guess if I can spark a good discussion involving people who (unlike
me) actually know something, it's not all in vain.

Cheers,
Liam.

--
Liam E. Gumley
Space Science and Engineering Center, UW-Madison
http://cimss.ssec.wisc.edu/~gumley

Subject: Re: Passing info and destroying widgets...
Posted by Struan Gray on Tue, 22 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Liam Gumley, Liam.Gumley@ssec.wisc.edu writes:
>
> David Fanning <davidf@dfanning.com> wrote in message
>
>> Liam Gumley (Liam.Gumley@ssec.wisc.edu) gives us an
>> example of a program that can record the last instance
>> of a button push in a non-blocking, non-modal widget
>>
>> No question it works. But I would argue that it works
>> for all the wrong reasons and is a *terrible* programming
>> practice in almost every instance.
>
> Well I guess I'll have to say that my example only
> demonstrates that it *can* be done, not that it

> necessarily *should* be done.

I've been writing generic helper widgets which behave like the tool palettes and pattern swatches found in drawing programs. Because these often manage properties that can be changed elsewhere in the application, and because they can be left floating about ready for use at any time, other widgets need to be able to get and set information about the helper widget's state.

Liam's technique works, but it is ugly (sorry Liam :) and opens an economy-sized can of worms. My concern is less that users will create memory leaks, but rather that they will come to depend on a particular info structure or tag name being present, which makes it hard for me to revise the helper widget later.

My old solution was for both the main and the helper widget to send custom events to each other (in the same way that David's colour table pickers can update draw widgets on 24-bit displays). The helper widget was defined in a function that returned its own widget ID instead of a pointer to the info structure.

This works, and is consistent with the overall widget methodology, but it also causes code-maintainence problems. I find I lose track of where the custom event structures are defined (sometimes IDL does too) and they have a habit of proliferating to an alarming extent as the functionality of the helper widget increases. Also, coding discipline is demanded to ensure that event handlers of widgets using the helper in a simplistic way don't lose or trip up on events they are not interested in.

The new (actually, THE ONE TRUE) way objectifies the widget as described by Mark Rivers. This simplifies event handling (and debugging) because the info structure is now referenced by SELF.xxxxx. It never needs to be fetched or restored so it can't go missing. The helper widget creation function returns its object ID and the other widget(s) can get and set information through methods. Standard events can be sent to the encapsulated widget by a DO_EVENT method, and as I mentioned in my earlier post the user now has a way to prioritise events, bypassing the very crude event management available in WIDGET_CONTROL. Code becomes more readable since you avoid having to write (*infoptr).xxxxx everywhere, and I find that procedures and definitions tend to end up in more logical places on my hard disk.

Why bother? Here's an example: I have a generic viewer widget for 3D model objects. The viewing angle can be set by an embedded trackball, or by choosing menu items which nudge the object a few degrees about various axes, or by setting the whole thing spinning continuously. More exact angular movements, as well as control of the

spin axes and rate, are specified in a helper widget.
Objectification (reification?) makes it much easier to handle the multiple ways of setting and displaying the viewing angle and letting all the interested widgets know that changes have occurred. I can prioritise user commands like 'stop spinning' or 'reset angle' and I don't have to worry about them being lost from the event queue when I remove piled-up timer events after long redraws. Finally, I can do gee-whiz things while spinning, such as simultaneously changing background colours or even editing the structure of the model object in yet another widget.

Sorry this is a bit long, but I'm enthused.

Struan

Subject: Re: Passing info and destroying widgets...
Posted by steinhh on Tue, 22 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <7kn0h5$s0c$1@news.doit.wisc.edu> "Liam Gumley" <Liam.Gumley@ssec.wisc.edu> writes:

> David Fanning <davidf@dfanning.com> wrote in message
> news:MPG.11d85d6ea2cc37d9897de@news.frii.com...
[....]
>> ... I mean, you can write
>> an object method that returns a data pointer too, but
>> by doing so you violate every tenet of good object programming
>> practice, in which the data should be encapsulated and
>> unseen by the outside world.
[....]

So what good is it, then, if it cannot be seen :-) See below.

[Liam Gumley:]
> My personal programming philosophy is to only use pointers
> where absolutely necessary (e.g. for retaining information when
> a widget dies, or for storing structure elements which are of
> unknown size and type until runtime). I've never used HEAP_GC
> in any of my programs; I rely on simple wrappers like those I
> posted at http://cimss.ssec.wisc.edu/~gumley/pointers.html to
> prevent me from getting into memory leakage problems.

>> As for me, I'm sticking to widget programs that clean
>> themselves up and don't leave the user holding the bag,
>> er, pointer. :-)

> I agree that widget *users* (e.g. of David's fine XCOLORS
> routine) don't want to know (and shouldn't be allowed to touch)
> the internals of "shrink-wrapped" widget routines. But I think
> any IDL application developer will eventually run across cases
> where items like information structures need to be shared
> between widgets which are performing closely related
> application-specific tasks.

I must say I agree a lot more with Liam than David here.

Although users in general should not be bothered with pointers to object data, there are (at least) 3 different types of users with different types of needs: users, damned users, and scientists.

Hey, some of them can even write IDL programs :-)

What do you do if your objects contain *large* arrays that need to be manipulated, analyzed or plotted - in a fashion that's completely impossible to foresee when you write the object methods in the first place.

There are basically two ways to do it "David's way", without slipping a pointer out to the user:

1. "Just write an object method to do it".
2. Use objects like handles

No. 1 is the "right thing", but requiring a user to write an object method in order to plot his data in a new and exciting way seems a tad, well, optimistic. We're supposed to *hide* the internals of the object from the user, not let him loose on the inside of the object, for heavens sake.

No. 2 is, I suspect, David's favourite way, but it is *either* memory inefficient *or* includes some awkward handle-like syntax plus "security risks", just like giving out a pointer to the user does (in some respects larger, in other respects smaller).

The memory inefficiency is of course related to the fact that if you're not using pointers, you need to provide a copy of the data to the outside whenever the outside (user) needs to look at it, like this:

```
IDL> data = obj->getdata()
IDL> print,sigma(data)
```

And let's say you write a widget program to display the data from two such objects alongside each other - it would have to make a

copy of each of the data sets every time part of the data needs
to be redisplayed.. And so on.

Or, you could use something like NO_COPY to avoid this but once
again it comes with an extra "security risk":

```
pro my_tiny_analysis_program,obj
   data = obj->getdata(/no_copy)
   print,sigma(data)
end
```

Bye bye data!

Actually, I'd say that providing a *pointer* is a lot less dangerous, since you're more aware of the fact that you're messing with something that points to the data itself, it's not just a normal, dynamic variable.. The example above would also be totally benign if "data" was returned as a pointer.

The one point where pretending an object is a handle would give you some benefits, is to protect the user from e.g. changing the dimensions/type etc of object data without the object detecting it (just check for it in the "obj->setdata,data" call).

But how do know it's time to remind him you want your data back..

And why should we go back to handle notation? Do you *really* want to write this simple action with three lines of code:

```
IDL> data = obj->getdata(/no_copy)
IDL> print,sigma(data)
IDL> obj->setdata,temporary(data)
```

instead of just:

```
IDL> print,sigma(*obj->getdata())
```

So, I hope that not everyone listens too carefully to David's advice - maybe I could even persuade David to at least mention these issues in a future book...?

Regards,

Stein Vidar

Subject: Re: Passing info and destroying widgets...

Posted by rmlongfield on Tue, 22 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <7kn0h5$s0c$1@news.doit.wisc.edu>,
"Liam Gumley" <Liam.Gumley@ssec.wisc.edu> wrote:
> David Fanning <davidf@dfanning.com> wrote ...
>> Liam Gumley (Liam.Gumley@ssec.wisc.edu) gives us an
>> example of a program that can record the last instance
>> of a button push in a non-blocking, non-modal widget ...(stuff cut)
but
>> by doing so you violate every tenet of good object programming
>> practice, in which the data should be encapsulated and
>> unseen by the outside world. Sucking the pointer out of
>> a widget program, except perhaps in the hands of just the
>> best programmers, is a practice that is guaranteed, it
>> seems to me, to get most of the rest of us in a hell of

This is really funny. As I was reading Liam's suggestion, I was thinking that this was a fantastic idea and something that I would certainly try to implement. Then, I go on to read David Fanning's comments and am dismayed. Well, you can be sure that I will think think carefully if I ever decide to use it.

I've recently been forced to deal with too many dangling pointers because my image tool gave me errors after I was creating (and not destroying) too many of them. I also now have two delete event handlers for each top level widget I create. One is called with a quit button and explicitly deletes all the pointers I have created. The second one is a cleanup routine (learned from DWF's book) which destroys the toplevel ID:

```
PRO Image_panel_cleanup_event,lasteventID
help,lasteventID,/structure
print,'At the very end of Image cleanup'

WIDGET_CONTROL,lasteventID,GET_UVALUE=image_infoPtr
       PTR_FREE,image_infoPtr
END
```

This removed the errors related to too many pointers. Using HEAP_GC just got me into trouble because I didn't really know what I was deleting or why. I feel happier that my explicit deletion works well and I know why.

Actually, using Liam's pointer solution might enable me to avoid using the COMMON statements I just put into my programs to make my data accessable to other programs. It's a difficult choice....

Rose

Sent via Deja.com http://www.deja.com/
Share what you know. Learn what you don't.

Subject: Re: Passing info and destroying widgets...
Posted by J.D. Smith on Thu, 24 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Struan Gray wrote:

> The idea of a objectified widget I owe to Mark Rivers. Deja News
> has a thead with a neat discussion of his technique, plus a few
> refinements - search on his name and 'objects'. My widgets follow his
> scheme, with a few inherited properties that I like all my
> program-oriented objects have (such as a unified way of handling
> global and user preferences) and generalised information
> sharing/passing methods (the above, plus the ability to handle
> conventional events).
>
> At present the parts work, but the whole looks like it's in the
> middle of open heart surgery. I'm building a disperse set of
> data-objects, widget-objects and plot/analysis-objects and at present
> I'm playing around with different ways of distributing basic
> behaviours among them. I'm not sure when it will be ready for public
> consumption, but I promise to make what I have freely available when
> it is.

I have developed something along these lines and have been using it for over a year in all my new widget applications. I'd be interested in seeing what you have and how it compares. I really think object widgets (Obgets, for short) are the way to go; so much so, in fact, that I've toyed with rewriting XManager to be more object friendly. No reason I shouldn't be able to say:

XManager,"MyApp:"+self.name, self.base, /NO_BLOCK, OBJECT=self,
Method='thisEventHandler'

But even this doesn't go far enough. Taking this progression even further, XManager could be scrapped altogether in favor of an event-handling, message passing class for obgets. Though my framework, contained in a class called "ObjMsg", does not directly manage the widget events of its obget progeny (requiring XMananger still to be used), it does implement a generic, feature-based message passing protocol, which has proven very useful. ObjMsg objects can "publicize" the features or services they offer, and others can "subscribe" to those services, all of which can be modified dynamically during runtime. Some of these features might be just the passing of plain widget events being generated in a widget an obget contains, but more commonly, they are messages distilled from one or more widget events, or independent of widget events altogether. Here's the first paragraph of the doc/blurb:

```
;+
; NAME: ObjMsg
;
; PURPOSE: A superclass to define a common prescription for event-
;       driven object communication.  The events will include those
;       which arise from widget activity within the objects, but the
;       formalism is extensible to any generic 'object events'.  Both
;       kinds of events are encapsulated by the term 'messages', and
;       are referred to as "object messages" when handled by the
;       protocol defined in this class.
```

Basically, once you have a generic message passing protocol, you are free to implement whatever message flow structure is convenient, not just the "up-the-widget-tree" technique implicit in normal widget programming.

With this freedom comes complexity, but also power. For instance, one nice feature is the ability to debug your large application by "snooping" on the messages being passed. Since they all travel through the ObjMsg class, you can for instance, do things like "show all messages of types [msg1,mgs2,msg3] from objects of type X, and show who is sending them to whom, and the calling sequence which resulted in their being sent".

I would like to make several improvements, and I think some kind of obget programming super-class could be very useful, potentially even part of the IDL distribution. Things I would like to do include:

1. Handle the widget events of the obget progeny -- i.e. obviate XManager. While XManager is a "private" RSI routine, it really doesn't do too much at all, just calling widget_info, widget_control, and widget_event, albeit with a few undocumented keywords. ObjMsg wouldn't have to provide backward compatibility for "fake" Modal widgets, and non-blocking widgets aren't really handled by XManager anyway, so this changeover could be accomplished without too much fuss.

2. Make the service "publication" and "subscription" more robust, uniform, and easier to understand. Currently each ObjMsg object is responsible for organizing it's own services for publication, which leads to different conventions for subscription to different class

services. Ideally, anyone could write an class which inherits ObjMsg, name and describe the services it provides, and, without seeing any source code, someone else could subscribe to and make use of those services.

3. Update the debugging features mentioned above.

Anyway, it seems like quite a few of you are converging on this type of idea, and I bet we could come up with something very useable if we put our heads together.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: Passing info and destroying widgets...
Posted by mallors on Thu, 24 Jun 1999 07:00:00 GMT

View Forum Message <> Reply to Message

In article <376E938D.86AAD3B6@ssec.wisc.edu>,
Liam Gumley <Liam.Gumley@ssec.wisc.edu> writes:
\> "Robert S. Mallozzi" wrote:
\>> I believe you must use XMANAGER in blocking mode for
\>> this technique to work.
\>
\> I've used this technique successfully in blocking and non-blocking
\> modes. As long as the main widget procedure creates a new pointer for
\> each invocation, there is no problem.
\>

Uhh, I stand corrected. Heck I've even used this somewhere before...Lesson for the day: if you've spent the last 2 months with C++, *try* the IDL code before you hit the "Send" button :-)

--
~~~~~~ ~~~~~~~~~
Robert S. Mallozzi 256-544-0887
Mail Code SD 50
Work: http://gammaray.msfc.nasa.gov/ Marshall Space Flight Center
Play: http://cspar.uah.edu/~mallozzir/ Huntsville, AL 35812
~~~~~~ ~~~~~~~~~

Subject: Re: Passing info and destroying widgets...
Posted by philaldis on Mon, 28 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

On 21 Jun 1999 16:25:04 GMT, Struan Gray <struan.gray@sljus.lu.se>
wrote:

> Interesting discussion snipped

Well while we're all talking about the various styles of widget programming that we use, I thought I may as well throw my hat into the ring.

At the moment I'm writing a direct graphics version of IDL's insight. Those of you who've seen insight will know that there's a data manager and a visualisation manager. The data manager has to hold all the data objects and the visualisation mamnager has to hold all the visualisation objects. It is essential that these two, and the main program can all communicate.

The only solution is that all the widgets are objects. They are all, though, very separate entities, and so I make them kind of compound/object widgets. i.e. I call
 datMan = dataManager(PARENT=mainWID)
dataManager then creates its own tlb, and then creates an object of the class dataManager. Both the visualisation manager and data manager are subclassed from DERA_Container, which is essentially just like IDL_Container. They contain all the objects.

The UVALUE of one the menu buttons is set to the object so it can be got at. I then use the FUNC_GET_VALUE to create a function which when called returns the object reference. This means that the other object can then invoke relevenat methods and so on. When an event is received, the object reference is got, and then an object event handler is called with teh event passed in. This event handler calls all the relevant method.

I find this method is by far the best way to create complex widget programs where lots of different elements interact. The fact that other objects can call other objects methods is so powerful. For example, in my main program I have a button which says
'Import file...'
When button is clicked, it simply does:-

Widget_Control, self.datMan, dataObj
dataObj->inportFile

This is the sort of integrated power that can be achieved.

Cheers,
Phil

Subject: Re: Passing info and destroying widgets...
Posted by Struan Gray on Mon, 28 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

J.D. Smith, jdsmith@astrosun.tn.cornell.edu writes:

> Anyway, it seems like quite a few of you are converging
> on this type of idea, and I bet we could come up with
> something very useable if we put our heads together.

I'm game. A semi-standardised set of objects and methods for handling basic functionality will make it much easier for us to share 'real' code later. Usenet has a fine tradition of awarding victory to the least lazy, so I suggest that those of us with working models and ideas should stick them on websites for others to peruse and comment on, and we'll see if we can interate to a solution.

Sadly, despite my recent rash of posts I am genuinely too busy until the end of July to do more than snipe from the sidelines. I promise to expose my own code to a cruel world Real Soon Now.

Struan