Subject: Re: Object Widgets
Posted by davidf on Sun, 07 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Bernard Puc (bpuc@va.aetc.com) writes:

> What's the current status of writing object widgets?

Alive and well. :-)

One of the difficulties with object widgets is handling the events that get generated. It is not possible, of course, to assign object methods as event handlers, so you have to have an event handler that dispatches the events to the appropriate methods.

There are several ways to do this, but one I am starting to use a lot is to use the user value of every widget that will generate an event to store the name of the method that should be called as an event handler for that widget. I write these event handler methods exactly as I do any other event handler, except that I don't have to worry about the info structure, since that is now part of the self object.

I store the self object in the user value of the top-level base (where I previously stored the info structure). This makes it possible for me to write my event handler that dispatches events to the appropriate method in exactly the same way for every object widget I write. (Note that the only case where this doesn't work is when the top-level base generates an event, I.e., a resize event. Thus, I have a special case for this.)

```
PRO MyClass__Events, event
Widget_Control, event.top, Get_UValue=selfObject
IF event.top EQ event.ID THEN thisMethod = 'RESIZE' ELSE $
   Widget_Control, event.ID, Get_UValue=thisMethod
Call_Method, thisMethod, selfObject, event
END
```

So, if I want to add a button, for example, to change the data colors, I can define the button like this:

```
colorsID = Widget_Button(colorsBase, Value='Change Colors', $
   UValue='DataColors')
```

The "DataColors" event handler method is written like this:

```
  PRO MYCLASS::DataColors, event
  thisEvent = Tag_Names(event, /Structure_Name)

  IF thisEvent EQ 'WIDGET_BUTTON' THEN BEGIN

    XColors, NColors=self.ncolors, Group_Leader=event.top, $
      NotifyID=[event.id, event.top]

  ENDIF ELSE BEGIN

      ; Must be an XCOLORS load color table event.
      ; Save color vectors and redraw on 24-bit displays.

    self.r = event.r[0:self.ncolors-1]
    self.g = event.g[0:self.ncolors-1]
    self.b = event.b[0:self.ncolors-1]
    Device, Get_Visual_Depth=thisDepth
    IF thisDepth GT 8 THEN self->Draw

 ENDELSE
 END
```

Since this event handler method is essentially identical to
the way I would write it normally, it makes it easy
to take previously written widget code and turn it into
object code.

But why bother? Well, the primary reason is that
I can make the object do things that I can't make
a widget program do. For example, with an object I
don't really *have* to have a GUI interface, unless
I want one.

I wrote a DrawColors object this weekend that acts
as a combination of my non-GUI program GetColor
and my widget program PickColor. In other words,
I create the object like this:

```
  colors = Obj_New('drawcolors')
```

If I want to load a yellow color at color table
index 200, I can do this:

```
  TVLCT, colors->GetColor("yellow"), 200
```

However, if I want the user to select a color for
color index 200 from the 16 drawing colors I have

available in a graphical user interface (or if
I want them to mix their own drawing color), I
can call the Select method, which puts up a
graphical user interface, like this:

```
thisColor = colors->Select(Cancel=cancelled)
IF NOT cancelled THEN TVLCT, thisColor, 200
```

The first case involves no GUI, the second case
does.

And if I suddenly decide that I want the sliders
in the HSV color system rather than the RBG color
system I originally designed, I can simply write
a new method for the object that changes the
labels on the sliders and the way the sliders
work. I don't have to worry about redesigning
my whole interface.

I hope this gives you some ideas. I'll probably
have the DrawColors object available on my
web page later this week after I add the program
documentation. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155