Subject: Re: When should objects be used?
Posted by davidf on Sat, 26 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Richard G. French (rfrench@wellesley.edu) writes:

> OK, you need to help me out here - what is the difference between
> direct graphics objects and object graphics objects?

There are two graphics systems that you can use in IDL.
One, the direct graphics system, is what you are using now
and has been a part of IDL from the beginning. The other,
the object graphics system, was introduced in IDL 5.

The two systems are completely separate and different. You
cannot mix and match. You use one or the other. In fact, the
way they are implemented is that you create either a
direct graphics window (I.e., Window) or an object graphics
window (I.e., window = Obj_New('IDLgrWindow')). You cannot
display direct graphics in an object graphics window and
visa versa.

When RSI introduced the object graphics system, they also
introduced an object class library of low-level objects
that could be used to produce object graphics. In general,
this library is much harder to use than direct graphics,
although you have a great deal more power and control
over how things are displayed than you ever did in direct
graphics. (There are also things you *can't* do in object
graphics that can easily be done in direct graphics. For
example, you can't currently label a contour line with
its value in object graphics.)

One of the things that is holding object graphics back,
in my opinion, is not that they are so hard to use (it
takes about a page and a half of code to reproduce the
direct graphics PLOT command, e.g., see XPLOT), but that
they take so darn long to print. The object graphics
system is a true 3D system. And *everything* is done in
3D, even 2D line plots. So every time you do something
with an object graphic, you have to carry around a LOT
of information. That is what makes simple PostScript files
of a PLOT command appear in the 10-20 MByte range. (RSI
is making efforts to reduce the size of these files.)
As a result, object graphics can be slow and printing
can be extremely frustrating unless you have a coffee
machine nearby.

Now, it turns out that an awful lot of what we do is NOT done in a 3D space, but in a 2D space. Line plots, contour plots, image displays, etc. Using a 3D graphics system for these 2D requirements is overkill. It not only slows these things down, but it makes them hard to print, etc.

But on the other hard, objects are really NICE! Each object's properties are "sticky" if you like. If I create a plot object, for example, and tell it I want it to draw its data in a green line with red symbols, it is going to draw its data like that forever. I don't have to set a COLOR keyword every time I want it done that way. I just tell the object to draw itself and it knows what it should do. And, of course, each object of that type that I create has its own sticky parameters. So I can create these plot objects and each can plot its data in different colors, linestyles, etc. And each can remember its own state. (Sound like a widget to you?)

Well, one of the side benefits of RSI introducing the object class library, for their new object graphics system, was that they had to create and introduce a way to create this new object data type. So, alright, the object graphics system is a nightmare to learn and we don't need 3D graphics anyway. Everything we do is in 2D space. What can we gain from objects?

Well...everything!

We can just write our own objects to do whatever we want them to do. I happen to like plots that can remember than I want a charcoal background and yellow axes and green data lines with red symbols that can *always* go into the same location in a display window, whether that window is on my display or in my PostScript file, or in the Z-buffer or whatever it is. So I can build one.

And you can use it because I am going to publish the "methods" which are the procedures and functions that you can use to interact with my plot object. For example, I might tell you that to change the axes color you use the method ChangeAxisColor like this:

```
myPlotObject->ChangeAxisColor, 'beige'
```

Now anytime you draw that plot in a display window:

```
myPlotObject->Draw
```

it is going to be using a beige color for the axes.

But here is the thing. You don't know *how* I am actually drawing the plot, or even setting the color of the axes. Nor do you care much, as long as it works. *How* it is done is left up to me and I can be as creative (or not) as I want to be.

> Can you describe a simple example for those of us without
> brains that can process abstractions?

Well, I've already done it. I've even gone to the trouble of documenting it *extremely* heavily. But no one seems to be looking at it. :-)

You can find the direct graphic colorbar object on my web page:

http://www.dfanning.com/programs/colorbar__define.pro

A colorbar is a 2D sort of thing. I often use colorbars with filled contour plots. I already mentioned that I can't imagine anyone using the object graphics contour plot, because you can't label contour lines. So I built this direct graphics colorbar object to go with my direct graphics contour plot object. (Although I give a LOT of stuff away, I don't give *everything* away. You will have to pay me for the contour plot object, it's that nice. :-)

To see how this works, let's open a window and display an image. You will have to get my TVImage and LoadData programs to run the code below, or you can use your own image:

http://www.dfanning.com/programs/tvimage.pro
http://www.dfanning.com/programs/loaddata.pro

Here we go:

```
image = LoadData(7)
Window, XSize=500, YSize=500
TVImage, image, Position=[0.25, 0.1, 0.75, 0.75]
```

You create the colorbar object like this:

```
cbar = Obj_New("COLORBAR", Index=5)
```

Here I create a colorbar with color table 5. I've left
other parameters (Position, Range, Color, Title,
Fontsize, etc., etc.) to be set to their default values.
By default, this is a horizontal colorbar positioned
in the upper part of a window, so to see it I can do this:

```
cbar->Draw
```

Suppose that is not really where I wanted it. Suppose I
really wanted a vertical colorbar. Then I could do this:

```
; Erase the current colorbar.

cbar->Erase

; Make it vertical.

cbar->SetProperty, Vertical=1

; Redisplay it in the window.

cbar->Draw
```

Whoops! I wanted it on the left side of the window, not the
right and resized for the image. As it happens, I wrote the
SetProperty method so that I can automatically erase and
redraw the colorbar object when I make a change to it. So,
to move the colorbar for the right side of the window to
the left and make it the right size, I do this:

```
cbar->SetProperty, Position=[0.15, 0.1, 0.22, 0.75], /Draw, /Erase
```

Ever see a direct graphics program do that!? Without moving
the image?

No, me either. But that is what is possible with objects. :-)

The code is all there. If you want more details, I can put
you on the list to be the first one (actually about 15th now)
to buy my new book. :-)

Cheers,

David

P.S. Whoops! Objects are persistent, so we have to get rid
of them when we are finished. If we don't, we will have
memory leaking everywhere. :-)

Obj_Destroy, cbar

P.S.S. Works in PostScript *just* like it works here!

--
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