Subject: Re: When should objects be used?
Posted by davidf on Thu, 24 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Uh, well, gosh. If people are going to be serious about this...

I have two main criteria I use for creating an object:

1. I'm writing a compound widget. I write these as objects because they are so much easier to control. Getting the value of a compound widget means getting its object reference. Then I can do anything to the compound widget I like by calling a simple object method. When I remember I wanted to do something else to the compound widget, I just write a 3-4 line new method and it's done. No fooling around.

2. I want a "thing" to have some intelligence. For example, I want a plot that knows how to position itself in a window and what linestyle and symbol I want to use. I want a smart image that knows and can remember what kind of processing I have applied to it. I want a contour plot that knows how to position itself on a map projection or not and whether I want a colorbar with it or not. That kind of thing.

Most of the objects I write are of the second type. The real advantage to me of these "smart" objects is that they can be easily controlled by widget programs. For example, the user can choose the plot background and foreground colors from a palette of drawing colors, just by clicking on a pull-down menu item. I don't have to hardcode a couple of colors into my program and listen to how my user doesn't like my aesthetic sensibilities. "Do it yourself", I say.

I do the same thing with smart images. Sure, I have an *idea* how this image should be processed. But what if my user wants to smooth the image *before* the edge enhancement step, when I was certain they would do it *after*. What if they want to smooth twice before this step? My program can accommodate any ol' thing the user is stupid enough to want to try, even though I know better. A smart image can be processed any way at all, and if you don't like what you just did, you can even undo it, because an UNDO method is part of what a smart image is.

The very first object I wrote was a smart window that was simply told what kind of grid it should use to lay things out (I.e., 2 column by 3 row). Given a "plot object" it could lay those things out in its grid. The "plot" could be any old thing at all. The window didn't know or care. This

made it possible for me to write Copy, Cut, and Delete methods for things that were in the window. I had never thought of doing anything like this before. Now the user could set up the window the way *he* wanted it set up. Not the way I imagined he wanted it set up. He could even drag something from this position and put it over there.

I guess this is the thing about objects that has most amazed me. Building objects tends to *generate* more ideas about what to do with them than I ever got when I was writing programs in the old linear way. All of a sudden whole new ways of working and interacting with data is possible. The problem is often not "what would I use an object for", but "how can I stop having ideas so I can stop writing this damn program and get some work done". They have been liberating for me, but then I'm not the world's most innovative programmer, although I've always been a pretty good mimic.

I like objects because they make me feel brighter than I really am. (And tossing the jargon around really intimidates people who ask you a question about IDL you don't know how to answer. :-)

I know because about half the time when I read these object posts in this newsgroup I think to myself, "Fanning, *you* are writing a book about objects!? Who do you think you're kidding!"

Cheers,

David

P.S. Let's just say objects are a whole lot cheaper than cocaine. ;-)

--
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

Subject: Re: When should objects be used?
Posted by John Persing on Thu, 24 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

I have used objects for over a year and spurred some of the recent discussion. I have used two objects, in four different contexts. But

before I start with that, let me make two observations on your programming: 1) it works, 2) and you use reusable programming units. Those are the two more important features of any scientific programming effort.

Use widgets if you want to interact with the data in a random manner to explore different ideas. In this context, your widgets would only "front-end" the codes that you have already written. I agree that you don't want to produce your publication figures with a widget. There are other options, like storing all your variables created by the widget in a structure that can be saved or come up with a mechanism that can store your button clicks, but those are probably not as robust as handwriting a procedure does that things that you would have done on the widget.

The first time I used objects was to create an "editor" in a text widget, but the "editor" was to be a restrictive program development environment. The skinny is that the user would have a lot of liberty to edit the lines of the "editor" in some ways, but with very great restrictions in others. What I wanted to do was put strict curtain around the data edited by the "editor", which is one advantage of objects. Because all changes in the object must be routed through a limited number of methods, you can control strictly how the object is changed. But if you are not coding for a "dumb user" (a programmer should be arrogant enough to claim it is all the other people who are "dumb") then this shouldn't motivate you.

The second object I made was a little more ambitious. I made an object to store any kind of data and I have used this object with three datasets with entirely different structures. The first two datasets were just to test the concept, while I really had the third problem in mind. The first dataset was to read a text file containing the radii and masses of each of the planets and of each of the satellites around each planets, so basically a structure array with an additional mechanism for attributing a satellite to a parent planet. I discovered that the routines I want to attach to the object are those that are well-used and are intimately related to the data itself, such as the utility that attributes satellites to parents. The second dataset was to store the x and y positions of N point vortices over T time steps, so basically it was a multidimensional array. An object is certainly not the best way to deal with this problem either, but it was useful for testing concepts. A side benefit though was that I was able to create a single, master widget interface that could interface with all three datasets by reusing this object. (How I did that would be a topic of another long post.)

But the third problem is where the object is ideally suited. I ingest a bunch of large 4-D arrays (yet small enough for all to stick in memory) of temps, winds, moisture, etc. that is output from an atmospheric forecast model. Then from these fields, I can compute derived fields, such as vorticity, horizontal gradients, etc. But I don't always. What I wanted to do was to create variable structure that would make it easy for me to write

code with. I want to be able to perform gradients or averages on any field, or even any list of fields. I want to do so at will. So I stuff all the fields into an object called "dat". Any new field get added to dat. All the fields are referred to by a string name. The data is stored as a simple structure array, where the first element is the "name" as a string and the second element is the "dat" as a pointer, which can point to anything. The primary methods to the object are "query", "assign", and "delete".

dat->query("u") asks for the u-winds.
dat->assign, "speed", SQRT((dat->query("u"))^2.0 + (dat->query("v"))^2.0)
creates a new variable with the wind speeds
dat->delete, "speed" gets rid of the new field.

Also, there is a mechanism for querying for a slice of a data field. Other methods check to see if a variable exists and to print out a summary list of the names of the fields stored. As you may see, this allows for a free-flowing mechanism for interacting with the data, liberally creating derived data at whim. Most of the mathematics are performed in separate procedures. One example is a code to produce radial profiles (of the cylindrical dataset) of an arbitrary list of data

compute_radial_means, dat, ['vort', 'u', 'v']

where the output will be stored in dat as the new fields "rad_prof_vort", "rad_prof_u", and "rad_prof_v". Because the datafields are referred to by string name and all the data fields are stuffed in dat, this free-flowing approach works. Keywords allow the additional retrieval of asymmetric portions and settings for dealing with masked data points. And again, I can use the same tested widget metaphor used above. This allows me to do one-liners at the IDL prompt with ease and write code quickly.

I have seen the short-comings of this approach, though. If a field isn't in the object, I either have to do alot of robustness coding, or just rely on the errors generated. I generally do the later because, of course, I am not a "dumm user". Also, I wish I would have thought of a mechanism for storing metadata for each field, specifically how to map each of the dimensions of each field to independent variables. I will definetly come up with an additional mechanism for handling bad datapoint masks, which are now not so satisfactorily done by setting a large negative number.

Some advice: Only use an object to solve a specific shortcoming of your present approach or better yet that you anticipate from your future programming effort. Also, implement objects from the start of the programming project. The choice of data structure should always be your first decision. Then you start coding.

--

}3 John Persing }3

http://www.frii.com/~persing persing@frii.com
Half of all Americans earn less than the median income!!!!!!

Subject: Re: When should objects be used?
Posted by davidf on Thu, 24 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

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

> My problem is that so much of the discussion of objects
> seems abstract that I have a hard time figuring out how to
> translate it to the problems I am trying to solve.

To tell you the truth, I read JD's articles five or six
times and I'm *still* not sure I have a clue. :-(

Cheers,

David

P.S. Let's just say that object programming is a whole
lot easier than a lot of people might lead you to believe. :-)

--
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

Subject: Re: When should objects be used?
Posted by mallors on Fri, 25 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <3772CE57.6BF093C8@wellesley.edu>,
"Richard G. French" <rfrench@wellesley.edu> writes:
> Nearly all of my IDL programs started as interactive bits of code, saved
> in
> a journal file and then expanded into programs. The main power of IDL
> for me is to be able to tinker with the code quite easily, try different
> things out, and then finalize the code in the fashion that works best.
> Most of these programs are the old-fashioned kind where you specify the
> conditions at the start (the data to be
> analyzed, the conditions of the analysis, the form of the result) and

[text omitted]

Hi,

I think the experts here in this group have probably given you enough info to peak your interest, but I just want to point out that one of the most useful features of object oriented programming is "code reuse". Unfortunately, for the type of exploratory science analysis you describe, it is sometimes difficult to decide what procedures would (or could) be suitable to be made into objects. I have found that for a lot of data analysis code that no one but yourself will be using, a full blown object-oriented design is not usually acceptable.

On the other hand, keep in mind that if you can write some useful utility objects, you can use them easily within your "old" procedural method of programming. For example, I was finding myself always manipulating my data file names, extracting the path, the file extension, etc. This type of task is perfect for encapsulation into an object, so I created a simple "File" object. This object can then be used anywhere in IDL - you don't have to have an super-duper full object oriented program. Perhaps the task you mention of writing TeX tables might be a good place to start:

```
   Table = OBJ_NEW ('teXTable')   ; Make an instance of a 'teXTable' obj
   Table->setColumns (5)          ; This table will have 5 cols.
   Table->setData (myData)        ; myData might be an array of structures
                                  ;  with 5 tags.  You could even omit the
                                  ;  setColumns() method, and just figure
                                  ;  out the number of columns
   Table->write ('myTable.tex')   ; Write a table
   OBJ_DESTROY (Table)            ; Clean up memory
```

This object could then produce TeX tables for any of your datasets,. Come to think of it, I would like to have this object...why don't you post it when it's done ;-)

Regards,

-bob

By the way, you can take a look at the File object on my web page: http://cspar.uah.edu/~mallozzir/software/idl/idl.html

--
~~~~~~ ~~~~~~~~~~
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: When should objects be used?
Posted by Richard G. French on Fri, 25 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> This posting from Pavel makes me realize that I probably
> didn't emphasize enough that the kinds of plot, contour,
> and image objects I was talking about in my previous article
> are definitely, emphatically NOT object graphics!
>
> I'm was talking about direct graphics objects that have all
> the advantages of object graphics objects, but are fast to
> display and will give you wonderful hardcopy output without
> waiting beside the printer for half the afternoon. :-)
>

OK, you need to help me out here - what is the difference between
direct graphics objects and object graphics objects? Could
you describe a simple example for those of us without brains that
can process abstractions?

Subject: Re: When should objects be used?
Posted by davidf on Fri, 25 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Pavel Romashkin (promashkin@cmdl.noaa.gov) writes:

> I am adding my humble opinion to the words of experts above - to say that I have
> not had a need yet to write my own objects (and, therefore, don't know how to do
> that :-). The only objects I use are graphics objects. The reason I preferred to
> use object graphics was not the ease of positioning, etc. but the need to re-use
> the data plotted. Direct graphics is way faster than RSI supplied object graphics
> to plot or print. However, I came finally to the point where I plotted the data
> that were a result of quite complex pre-processing, and that data were to be
> further modified, but visual inspection was needed. It was in a widget

> application, many intermediate variations were possible. And the data in direct
> graphics is lost, once you plotted it, unless you specifically store it somewhere.

This posting from Pavel makes me realize that I probably
didn't emphasize enough that the kinds of plot, contour,
and image objects I was talking about in my previous article
are definitely, emphatically NOT object graphics!

I'm was talking about direct graphics objects that have all
the advantages of object graphics objects, but are fast to
display and will give you wonderful hardcopy output without
waiting beside the printer for half the afternoon. :-)

Cheers,

David

P.S. I can't imagine anyone EVER using the object graphics
contour object in a real program. At least not in its present
condition. :-(

--
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

Subject: Re: When should objects be used?
Posted by Pavel Romashkin on Fri, 25 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Writing your own objects is another realm I think. Objects provided by RSI are
useful too.
I am adding my humble opinion to the words of experts above - to say that I have
not had a need yet to write my own objects (and, therefore, don't know how to do
that :-). The only objects I use are graphics objects. The reason I preferred to
use object graphics was not the ease of positioning, etc. but the need to re-use
the data plotted. Direct graphics is way faster than RSI supplied object graphics
to plot or print. However, I came finally to the point where I plotted the data
that were a result of quite complex pre-processing, and that data were to be
further modified, but visual inspection was needed. It was in a widget
application, many intermediate variations were possible. And the data in direct
graphics is lost, once you plotted it, unless you specifically store it somewhere.
Storing copies of slices from a 3 mb array is possible but I didn't like it. This
is when I went to objects. In my primitive application, I use different view
objects to display different modes of operation, and can extract the data from

objects tfor analyses - not to just change the appearance of the plots easily.
This way complicated dataset may be used interactively again and again, without
storing it separately.
I can't pretend to advise people on how to use objects. This is how object
graphics work for me.
Good luck,
Pavel

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) wrote in
an e-mail to me:

> I think I have your current version of colorbar__define but
> there is no INDEX keyword allowed in it. I get the
> following error message:
>
> % Keyword INDEX not allowed in call to: COLORBAR::INIT
>
> What is my mistake here?

Sorry. My fault. I accidentally had two versions of the
program in my directory and I grabbed the wrong one when I
tested the code. I think all is straight now and a combined
version is located here:

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

Sorry for the confusion.

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

Subject: Re: When should objects be used?
Posted by Richard G. French on Sat, 26 Jun 1999 07:00:00 GMT

David Fanning wrote:
>
> 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?
>
<snip snip snip>

David - I had a vague idea of the difference, but you made it crystal clear.
Add me to the list of people who will buy your book next month when you've
finished it!
Dick

Subject: Re: When should objects be used?
Posted by davidf on Sat, 26 Jun 1999 07:00:00 GMT

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

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

I wrote in bad syntax
> Perhaps the task you mention of writing TeX tables might be a
> good place to start:
>
> Table = OBJ_NEW ('teXTable') ; Make an instance of a 'teXTable' obj
> Table->setColumns (5) ; This table will have 5 cols.
> Table->setData (myData) ; myData might be an array of structures
> ; with 5 tags. You could even omit the
> ; setColumns() method, and just figure
> ; out the number of columns
> Table->write ('myTable.tex') ; Write a table
> OBJ_DESTROY (Table) ; Clean up memory

>

when of course I should have written

```
Table = OBJ_NEW ('teXTable')
Table->setColumns, 5
Table->setData, myData
Table->write, 'myTable.tex'
OBJ_DESTROY, Table
```

--

~~~~~~ ~~~~~~~~~~
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: When should objects be used?
Posted by Nick Bower on Sun, 27 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

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

i'd encourage anyone to read about OOP before looking at this area with IDL. While no expert, what experience i do have in this area before beginning with IDL a year ago made me realize from the start there more advantages to OOP than is implemented in RSI's attempt to incorporate this "facility". Unfortunately, imho, I think this is where it goes wrong - Objects aren't an add-on but should either be the basis of a language or not. I just really don't think learning how to program with objects in IDL as a first exposure is a good thing. go read some on object design. :) it'll take less than a few nights of reading to understand the basics and things will begin to make sense. but also some things in idl will just not make sense ;)

> 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.)

The first wall I came up against is that I couldn't incorporate a draw widget into a GUI canvas of buttons, menus, etc. It seemed that you had to have a specialized window dedicated for plots. Anyone manage to do

this in object graphics? I'd be interested to know, although I did end
up using direct graphics for maximum speed.

Being familiar with several visual object languages, it wasn't
immediately obvious to me how a widget object packer worked either.
view->scene->window, fine, but i couldn't work out how to arrange
buttons, labels, draw widgets etc into this and there weren't any useful
examples in the documentation that's for sure. maybe it's just my only
dedicating 2 days to this area - but why is there a IDLgrPlot object,
but no textbox, slider or button objects? I could never work this out
and saw this area as somewhat of a sad joke given my previous experience
in Python, c++ and ArcView.

> 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

personally i don't think they _need_ to be this difficult. they've just
been made that way because it's not object designed from the ground-up.
:(there are better ways to learn about objects. Python is the best
I've seen - alot friendlier than Java for starting out.

> 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

i think if we begin using accepted terminology such as classes,
instances, attributes, methods, etc it'll make it easier on the people
trying to pick the stuff up from external-IDL sources. just my opinion
though.

> 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.

i think an important thing here is that we can move away from that darn
concept of attaching structures to the user defined value of widgets.
That's just a work around the problem that gui programming is most
suited to an object approach I feel. when you have some decent objects
to start with that is...

> We can just write our own objects to do whatever we want them
> to do.

but i would cautiously point out that we are limited by the foundation

classes and subsequent class heirachy RSI have provided us with. so my advice isn't to sway anyone here into using objects or not, more to take in wider view of the field, then make up your own mind on RSI's OOP attempt.

cheers, nick

--

Nick Bower
Space Science and Engineering Center
University of Wisconsin, Madison, USA
Phone: (608) 265 8007
Email: nick.bower@ssec.wisc.edu
Web: http://arm1.ssec.wisc.edu/~nickb
