Subject: Re: Application programming--missing features
Posted by sterner on Fri, 16 Apr 1993 14:26:42 GMT
View Forum Message <> Reply to Message

jdlb@kukui.ifa.hawaii.edu (J-F Pitot de La Beaujardiere) writes:

> Though I love IDL and commend David Stern et al. for their fine work, I feel
> deprived of two important features which are important for application
> programming. I'd like to see them implemented, or failing that to be shown an
> elegant (i.e., non-tedious) way to simulate them.

> 1) Adding a "wrapper" to an intrinsic IDL routine is difficult.
> For example, consider William Thompson's <thompson@serts.gsfc.nasa.gov>
> just-posted routine PLOT_DROP for dropping bad data values when plotting
> data. That procedure, in effect, just adds a single keyword DROP_VALUE to
> the generic PLOT routine.

> In such an application, for each of the usual optional parameters accepted
> by PLOT one must make an entry in the procedure declaration and properly
> pass the parameter to PLOT. This involves either (a) defining defaults for
> each option or (b) tediously building up a command line and passing it to
> the EXECUTE function.

There is an easier way. Its not perfect, but it is much better than then the tedious technique described above (which I've done myself in the past). The key is the IDL execute function, mentioned above. To show the technique I will give an example custom plot routine that just puts a color band behind the plot curve. This routine adds one new keyword to the plot routine:

```
;-----  tplot.pro = test passing plot keywords to a custom plot routine ---
;       R. Sterner, 16 Apr, 1993

        pro tplot, x, y, args, back=back

        if n_elements(back) eq 0 then back=40  ; Color band color.
        if n_elements(args) eq 0 then key = '' else key = ','+args

        i = execute('plot,x,y'+key)   ; Do plot.
        oplot,x,y,thick=8,color=back   ; Plot color band.
        i = execute('plot,x,y'+key+',/noerase')  ; Replot curve.

        return
        end
```

All the plot options are available as far as I know. The reason this technique is not perfect is that the normal plot keywords must all be given inside a text string, unlike the normal plot call.

An example call:

```
IDL> x=findgen(100)
IDL> y=x^2
IDL> tplot,x,y,'linestyl=1,psym=-4,color=255,tit="Test",chars=3,
       xran=[60,80],/ynoz',back=60
```

The above would not fit on one line, but should all be entered on a single line. Note the new keyword, back. Try it with other plot keywords. Its not perfect, but much easier than other methods I've used.

I too would like to get my hands on the original calling line as suggested in the original post. Something like a new keyword: !last_call with the entire calling line.

> 2) User-defined global variables for customizing program behavior do not
> exist.

. . .

> The only two options are to (a) define a new system variable using DEFSYSV
> or (b) use common blocks. Option (a) fails because N_ELEMENTS(!FOO)
> returns an error ("Not a legal system variable") instead of zero if !FOO is
> undefined. Option (b) is very tedious for both programmer and user because
> common blocks are finicky beasts.
> The simplest solution would be to modify IDL such that n_elements(!foo)
> returns 0 if !foo is undefined.

I agree that n_elements(!foo) should give 0 for undefined system variables. I think IDL may be working on that.

I don't mind using commons. They are not so bad if you hide them from the user. I often initialize mine from control files. The user can setup something like .idl_xxx in their home directory with keywords defined inside (like zoom = 4). Comment lines should be allowed (I use both * and ; as comment characters) since options may easily be turned off without loosing track of them altogether. I use commons to share information among a set of related routines. One routine is written to initialize the common from the control file, but provide default values for any or all missing values. Another routine, called from all the others in the set, will check that the common has been initialized and if not call the initialization routine without bothering the user about it. I have found this technique to work very well.

One more way to get around the current limitations of IDL system variables is to define environmental variables. These are easily accessed from IDL: zm = getenv('IDL_ZOOM'), and easily tested for existance: if zm eq '' then zoom = 4.

Ray Sterner sterner@tesla.jhuapl.edu
Johns Hopkins University North latitude 39.16 degrees.
Applied Physics Laboratory West longitude 76.90 degrees.
Laurel, MD 20723-6099