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

Subject: Re: Application programming--missing features
Posted by chase on Fri, 16 Apr 1993 20:58:07 GMT
View Forum Message <> Reply to Message

In article <JDLB.93Apr15173546@kukui.ifa.hawaii.edu>
jdlb@kukui.ifa.hawaii.edu (J-F Pitot de La Beaujardiere) writes:

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.

I agree that a wrapper intrinsic would be very helpful. I quite often try to extend IDL functions and I often end up doing (b) above to build up a command line to pass to EXECUTE. Indeed (b) is very tedious. Typically when I use (b), I end up not allowing for all possible keywords so my new "wrapper" function can not be used as a complete substitute for the original. Additionally, it would never allow use of new functionality added to the wrapped procedures via new keywords without having to be updated. A "wrapper" ability would avoid this. (Gee, it sounds like I want inheritance and overloading as found in object oriented programming).

But how would you implement this? Would IDL allow a person to specify any keyword that isn't defined in your procedure? It seems that it would have to be done as a modification in the IDL kernel. You might not want to allow arbitrary keywords to be given to an arbitrary procedure because it would reduce the error checking ability.

One possible implementation:

On the other hand, if you did allow this, it could be implemented like the UNIX Bourne shell, where additional keyword parameters on a command line become part of the shell environment. In IDL's case additional undefined keyword parameters could be placed in a system variable table reserved for keywords and local to the called procedure. Then your "WRAPPER" intrinsic function could just be another form of EXECUTE that adds those keyword parameters to the command. Actually, new versions of CALL_PROCEDURE or CALL_FUNCTION would be sufficient with an optional keyword dictating that keyword parameters stored in the local "environment" be added to the called procedure or function. Depending on how procedure calls are compiled by IDL this could require substantial changes in the IDL kernel.

> 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 globals ala system variables would be a very useful option. I have felt that using common blocks would not work. In fact, I was not aware previously that (a) was possible. But the n_elements() certainly does make that unuseable.

1) and 2) would indeed be very useful features.

I wonder if people at RSI or PVI monitor this Newsgroup's posts?

Later,
Chris
--
==========
Bldg 24-E188
The Applied Physics Laboratory
The Johns Hopkins University
(301)953-6000 x8529

Subject: Re: Application programming--missing features

Jeff de la Beaujardiere (jdlb@kukui.ifa.hawaii.edu) writes:

> 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.

and Ray Sterner (sterner@tesla.jhuapl.edu) writes:

> 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:

[The routine "tplot" defines a single keyword "args" via which PLOT keywords
can be entered in a character string.]

> 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.

Thanks to the contributors to this thread for an enlightening and
interesting discussion. I have tried to summarise some pros & cons for each
of these 3 methods below. Perhaps people would like to point out anything I've
forgotten or got wrong:

(a) Define defaults for each option:

Tedious--all keywords to be passed through must be listed in 3
places: the definition of the wrapper procedure, setting defaults,
the call to the wrapped routine.

PLOT has approx 64 input keywords (Reference Guide Jan 93 pp 1-159ff)
plus 3 output keywords. If you want to pass all the input keywords
though the wrapper routine, the call to PLOT therefore has 64
keywords. As far as I can tell there is an IDL limit on the number of

keywords that can be passed to a routine of approx 60 (at least there is in IDL for Windows 3.0.1). Of course you could surely leave one or two out, but if you're going to implement most of them it is neater to implement all of them.

For some keywords, finding an appropriate default to set is problematical. For example the PLOT keyword position--see my recent post on "PLOT keywords vs System Variables". (This may not be insuperable but it's certianly irritating.)

(b) Building up a command line and passing it to the EXECUTE function:

Tedious--all keywords to be passed through must be listed in 2 places: the definition of the wrapper procedure, and the code to add each keyword (if defined) to the command string. Arguably it's less tedious than (a) because once the keywords are listed, the code can be built up by cutting & pasting in an entirely mechanical way.

EXECUTE is allegedly inefficient. (I don't think that this is an important issue for the kind of procedure we're considering here.)

Since EXECUTE can't be called recursively the wrapper procedure can't itself be wrapped in the same way.

As far as I am aware, there are no restrictions on the number of characters in a string fed to EXECUTE.

(c) Pass all keywords to the wrapped routine via a string.

MUCH less tedious to program than either of the above.

The calling syntax for the wrapper procedure is non-standard.

Variable names in the keyword-string aren't defined inside the procedure. For example, with Ray Stener's tplot, consider the following:

```
< x=findgen(100)
< y=x^2
< tplot,x,y,'xrange=[2,10]',back=128
 [Nice-looking plot appears]
< a=[2,10]
< tplot,x,y,'xrange=a',back=128
 [Plot appears without axes!]
% PLOT: Variable is undefined: A.
% PLOT: Variable is undefined: A.
```

Of course one could patch the value of a into the keyword string but that's more trouble than it's worth.

Restrictions on EXECUTE apply as for (b).

As far as I am concerned the provisional winner for most purposes is (b),
because it preserves (almost) all the functionality of the wrapped routine.
The wrapper procedure is certainly verbose, but largely transparent--you
don't have to know about the default behaviour for each keyword
to see if it's going to work.

Of course I'd be delighted to be told I've missed the point somewhere
and there's a much better way of doing things.

I concur with Jeff that IDL is a lovely package/language in many ways. To
someone who's done most of his scientific graphing using the NCAR Graphics
Fortran routines, it's a real eye-opener.

 -- ------
| Mark Hadfield   hadfield@wao.greta.cri.nz |
| NIWA Oceanographic (Taihoro Nukurangi) |
| 310 Evans Bay Rd, Greta Point   Telephone: (+64-4) 386-1189 |
| PO Box 14-901, Kilbirnie   Fax: (+64-4) 386-2153 |
| Wellington, New Zealand |
 -- ------

Subject: Re: Application programming--missing features
Posted by jdlb on Mon, 19 Apr 1993 21:47:12 GMT
View Forum Message <> Reply to Message

hadfield_m@kosmos.wcc.govt.nz writes:
> As far as I am aware, there are no restrictions on the number of
> characters in a string fed to EXECUTE.

Actually, there is a miserly 131-character limit, according to the manual.
One of my colleagues here (Hi, Tom!) cleverly works around the limit by
writing a temporary procedure which includes the command string, using
call_procedure to run the thing, and deleting the file.

There is a worse problem with execute: it fills up the code compilation area
if you run it repeatedly. I have noticed this with one of my own routines
and have just repeated it with the following test:

```
IDL> cmd = 'print,"foo" & print,"bar"'
IDL>print, execute(cmd)  ;works fine 1st few times:
foo
bar
1
```

```
IDL> print,execute(cmd)  ;after about 30 repetitions:

print,"foo" & print,"bar"
^
% Program code area full.
0
```

I know the size of the code area can be changed using .SIZE. (I have 64k set aside, which is more than the default.) That's not the issue, because no matter how big the space is made, it's bound to fill up. And once it's full, it seems you can't empty it or do anything useful. (You can issue a .SIZE, which will kill all your variables while it empties the code compilation area.)

Ray Sterner's <sterner@warper.jhuapl.edu> technique of passing keywords to a wrapped routine using a string is clever, and perhaps easiest for the programmer, but difficult for the user because of the non-standard syntax. And Mark Hadfield's <hadfield@wao.greta.cri.nz> comment about variables not being passed because they are merely characters in a string is an important consideration.

Regarding user-defined defaults for customization, I think Ray's suggestion to use environment variables is an excellent one.

--Jeff

% Jeff de la Beaujardiere % jdlb@mamane.ifa.hawaii.edu %
% Institute for Astronomy % 808-956-9843 %
% University of Hawai`i % fax 956-9402 %