Subject: Re: Ranting and Raving and getting back to global variables
Posted by J.D. Smith on Mon, 20 Apr 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Martin Schultz wrote:
>
> David Fanning (davidf@dfanning.com) wrote in a reply to J.D. Smith
> (jdsmith@astrosun.tn.cornell.edu) who, in another elegant
> and well-reasoned article, this time on main-level variables,
> ends the article by writing this:
>>
> [...]
>>
>>
>> Having spent considerable time afield (and I say this with
>> considerable humility and respect for the efforts of people
>> trying to learn IDL), I believe that more often RSI errs
>> on the side of *overestimating* the user's ability to harness
>> and control the power of IDL.
> True enough: I still remember my first months' struggle trying to
> convince myself that the effort of learning IDL may pay off one day. I
> strongly agree with David's suggestion that the IDL folks should
> make an effort to consolidate the powerful system they provide before it
> may one day become totally incomprehensible (just imagine David would
> retire one day ! ;-)
>
> Coming back to the point of global variables etc.: I still can't fully
> understand the need to create main variables in a subroutine. My
> philosophy (may be a bit antique ?) is that you should know what to
> expect from a subroutine when you are calling it, hence you can pass it
> a parameter which would in your example return the image that you loaded
> and manipulated (if you don't like to restrict yourself to an image, you
> can pass a structure and stuff it with everything you like). So what is
> the point of typing (at a minimum two !) additional characters to a call
> like
>
> my_fancy_widget_routine_that_does_everything_anybody_has_eve r_dreamed_of
>
> (simply add ,a and you can get back to the main level any kind of
> data, images, etc. you like).
>
> E.g in my case; I am very often working with 2D data sets that have an
> additional variable_names array associated with them. So almost all of
> my subroutines can be called as
> routine,data,header
> or sometimes
> routine,data=data,header=header
> and I don't even have to think about this any more when I type it. So

> why should I bother and call
> routine
> bring_to_main_level,data,header
> (or something alike) ?
>
> Should I sign with Joe Farmer now?
> Best regards,
> Martin
>

Allow me to elaborate on the situation which would require a more flexible mechanism for importing and exporting main level variables. For applications in which a fixed data structure is being dealt with, as in the case of your 2D one-at-a-time data sets, it is simple to pass data between programmatic levels with appropriately crafted arguments. This mechanism is perfectly adequate in many instances, and I use it extensively. However, if the data being created or manipulated does not have some a priori defined structure, size, or organization, the argument passing paradigm fails. This presumably is the very reason Insight chooses to do real importing/exporting from the $MAIN$ level: it is attempting to be a general purpose analysis tool that is not tied to any one specific format of the data it deals with. As David, I also don't know the details of how Insight was written, but it is apparently written in IDL (and is restored from a save file). I am uncertain how they could achieve this flexibility without special built-in functions which they're not telling us about.

Of course it is possible by building large cumbersome structures to achieve some approximation of this flexibility via called arguments, but this is a markedly less-than-ideal mechanism for user interaction. Imagine having to distentangle a large, varying-format structure everytime you wanted to pop back to the command line for some quick analysis. And another weakness of the argument-based mechanism is its inability to support the active command line; a widget based program must exit entirely before its arguments are exported to $MAIN$.

The real issue here is the duality of IDL environment engendered by an increasingly powerful set of widget-based programming tools. It becomes realistic to create elaborate and self-contained programs in IDL which strain the simple partitioned data space format -- a format conceived in the days when procedural units were much less complex, and most of the work in IDL was done on the command line. The next step in this evolution is of course the creation of IDL-based programs which obviate the command line interface altogether. Such programs are possible to design currently, but for what I do, I find that a mix of both interfaces is most efficient.

And as for the philosophical question of greater power vs. consolidation

and organization, I see it as a non-issue. I argue that if the introduction of new features and flexibility makes a program less accessible, they were not correctly implemented. The common backbone of all good programs I've encountered is the hierarchical organization of functionality: a gentle learning curve whose gentleness nonetheless does not impose arbitrary limits on how high the curve goes. I realize this is difficult to implement in the real world, but I don't see this as an excuse. Take as an example the IDL Advanced Development tools for linking with external programs, and even embedding IDL within a custom program. These tools are certainly above the heads of most IDL users (including myself, for the most part), but they are eminently useful and powerful. Most users, however, can be perfectly productive without knowing anything about them.

I believe IDL *should* focus on consolidating and cleaning their interface, but I don't think they should delay or inhibit the introduction of new features to help achieve this consolidation. As we all know, the simplest program is the one which does nothing at all.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-4083
206 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|