Subject: Re: Keyword precedence
Posted by davidf on Thu, 24 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield (m.hadfield@niwa.cri.nz) writes:

> The documented behaviour for IDL--the behaviour to which Jeff was
> referring--is specified in the following quote from "Building IDL
> Applications"
>
> Note that keywords passed into a routine via _EXTRA override
> previous settings of that keyword. For example, the call:
>
> PLOT, a, b, COLOR=color, _EXTRA={COLOR: 12}
>
> specifies a color index of 12 to PLOT.
>
> Contrary to what I wrote a month ago, I think this is usually the desired
> behaviour, because it makes it easy to write wrapper routines.

I absolutely agree that this is the desired behavior. It's
just that it buggers your programs occasionally and makes you
wish for the opposite behavior. :-(

I'm referring, of course, to those occasions when you
absolutely, positively DON'T want the damn color to
be mucked around with. Then you have to go fishing
for the COLOR keyword in the extra structure. It would
be OK if you could do something like this:

```
fields = Tag_Names(extra)
index = Where(fields EQ 'COLOR', count)
IF count GT 0 THEN extra.color = 127
```

But, of course, the user didn't use COLOR as the keyword.
They used C, CO, COL, COLO, or some other thing, and
you have to fish those things out as well.

I say it's easier to write somewhere in the program
documentation:

"And another thing. Don't touch the friggin' COLOR keyword!!!!"

But this comes up so rarely (I'm really easy with respect to
color and tolerate a lot of diversity), that I don't mind the
current behavior at all.

> For example,

> taking the PLOT example, one can imagine a MY_PLOT routine, a wrapper for
> PLOT, that specifies its own default for colour:
>
> pro my_plot, a, b, _EXTRA=extra
> plot, a, b, COLOR=127, _EXTRA=extra
> end
>
> MY_PLOT will plot data in color 127 unless the caller overrides it by
> specifying a COLOR keyword. If we can't rely on the documented behaviour
> then we have to make MY_PLOT more complicated, thus:
>
> pro my_plot, a, b, COLOR=color, _EXTRA=extra
> if n_elements(color) eq 0 then color = 127
> plot, a, b, COLOR=color, _EXTRA=extra
> end
>
> Anyone who has looked at the code on my WWW page will see many examples of
> the latter style. I would prefer to use the former!

Oh, I don't think so! Maybe you *think* you prefer the
former, but we have already established you might be
confused. I'd say this is the clincher. :-)

I would much prefer the latter, for this reason. The user
of the program understands that COLOR might be important
because there is a whole keyword devoted to it. It's documented,
it's up front, he knows if he uses it something appropriate
is going to happen.

With _Extra he doesn't know what to do. Should he use COLOR?
Will it do anything? What other keywords can he get away with?
If you point him in the documentation to some other routines:

  _Extra -- Picks up all the defined keywords for FOOBAR

the chance of him looking up FOOBAR would be just about nil,
I'd guess.

I think if a keyword is important, define it, and define
a default value for it. I wouldn't change any of your fine
code a bit, Mark.

> I mentioned an anomaly. This is illustrated by set of routines in the
> following .PRO file:
>
> http://katipo.niwa.cri.nz/~hadfield/gust/software/idl/ ->
> mgh_example_keywords.pro
>

> Reference inheritance appears to be broken.
>
> % MGH_EXAMPLE_KEYWORDS: Calling color-print routine directly
> % MGH_EXAMPLE_KEYWORDS_PRINT_COLOR: COLOR is 12
> % MGH_EXAMPLE_KEYWORDS: Calling color-print routine via value wrapper
> % MGH_EXAMPLE_KEYWORDS_VALUE_WRAPPER: Passing along EXTRA keywords in
> structure:{ 12}
> % MGH_EXAMPLE_KEYWORDS_PRINT_COLOR: COLOR is 12
> % MGH_EXAMPLE_KEYWORDS: Calling color-print routine via reference wrapper
> % MGH_EXAMPLE_KEYWORDS_REFERENCE_WRAPPER: Passing along EXTRA
keywords by
> reference
> % MGH_EXAMPLE_KEYWORDS_PRINT_COLOR: COLOR is 0

Actually, I think this code is working exactly the way
it is suppose to work. (Someone is going to have to pry
JD away from this thesis for the definitive answer. I'm
slightly confused about it too.)

But my understanding of how _Ref_Extra works is that inside
the routine that defines _Ref_Extra there is no possibility of
seeing what is in the extra structure. In other words, the extra
structure "passes through" that routine. I think what you are
looking at in MGH_EXAMPLE_KEYWORDS_REFERENCE_WRAPPER is the
definition of the extra structure and not the particular
instance of the extra structure itself.

(I happen to be re-reading Zen and the Art of Motorcycle
Maintenance at the moment, and I am struck by how much
that last sentence sounds like Pirsig's metaphysical argument
that "Quality is the *cause* of the subject and the object,
which are then mistakenly presumed to the the cause of
the Quality."

Huh!?

Anyway, I believe the last routine to get the extra structure
has to receive it via an _Extra keyword and NOT an _Ref_Extra
keyword. The _Ref_Extra is just the wormhole for getting the
damn value back to where you really want it, to put it in
Star Trek terms.

Hope that clears up any confusion. :-)

> Returning to Jeff's proposal, does anyone else see a need for a _DEFAULT
> formal keyword parameter.

I don't.

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
