Subject: Re: Keyword precedence
Posted by John-David T. Smith on Fri, 25 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> 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.)

OK, I'll bite.

The basic problem Mark is having is trying to manipulate the familiar _EXTRA structure in the context of _REF_EXTRA, which, though not strictly forbidden, won't do what you imagine it will. The reason is this: the data structure with which _REF_EXTRA deals is not the whole story. And it's not even a structure!. If you examine it within KEYWORDS_REFERENCE_WRAPPER, you'll see it is simply a list of string keyword names. This provides *part* of the by reference keyword inheritance functionality, with the other part being invisible to us, and inaccessible for our modification.

So, what is happening is as follows: _REF_EXTRA absorbs the first COLOR keyword, since the definition of KEYWORDS_REFERENCE_WRAPPER doesn't include it. It stores the name of the keyword in question, and *invisibly to us*, associates that name with a variable in both the caller and the callee. In this case it's simply a temporary variable which holds "0" -- so using _REF_EXTRA is somewhat of a waste. There is also another bit of functionality in _REF_EXTRA: the ability to seamlessly absorb regular _EXTRA structs as if it were simply an _EXTRA! (You can actually see this happening if you throw an "print, arg_present(color)" into KEYWORDS_PRINT_COLOR.) So this happens, and yet another variable is listed in the by-reference keyword inheritance list. This is the only reason you didn't get an error, trying to manipulate the inherited struct as you've done. Of course, only one of these can be used, and as it happens, the fully by-reference variable is chosen. I think generating an error in the case of multiple by-reference keyword variables passed would be preferrable.

This is actually spelled somewhat out in the manual, the relevant subsection of which I'll quote:

****
Accepting Extra Keyword Parameters

While you must choose whether a routine will pass extra keyword parameters by value or by reference when defining the routine (specifying both _EXTRA and _REF_EXTRA as formal parameters will cause an error), routines that accept extra keyword parameters can use either the _EXTRA keyword or the _REF_EXTRA keyword. However, it is not possible to both have access to the keyword values and pass them along to called routines by reference within the same routine. This means that any routine that needs access to the passed keyword parameters must use _EXTRA in its definition statement, or define the keyword explicitly itself.
****

The bottom line is that the use of _REF_EXTRA prohibits (for all intensive purposes) the use or manipulation of a standard _EXTRA structure. This prohibition is not explicit, since it was made to be as backwards compatible as possible (for those who do manipulate the _EXTRA structure). This might have been a mistake, since it encourages people to think of _EXTRA and _REF_EXTRA as fully interchangeable, though they in fact are not (as also described in the manual in some detail). It *does* allow for simple in-place modifications of the extra structure when there is a _REF_EXTRA somewhere in the inheritance chain. However, this is not a technique I recommend, due to the ambiguities you've discovered.

In order to achieve what I think you're after, _REF_EXTRA would need to consider overriding passed variables.... i.e. something like:

```
pro mypro,_REF_EXTRA=e
 myprocol=0
 another_pro,COLOR=myprocol,_EXTRA=e
end

IDL> maincol=12
IDL> mypro,COLOR=maincol
 or
IDL> mypro
```

So that anotherpro would have by-reference access either to $MAIN$ level variable "maincol", if the keyword were used, or mypro-level variable "myprocol", if not.

Then the question becomes, how usefule and how confusing is it to have a runtime-changeable location in which to store results passed through by-reference keyword inheritance? Either you want it on a given level, or you

want it on the level above.

Anyway, hope all this rambling clears some things up, or at least gets the brain cogs in motion.

Final synopsis: You want to play with the _EXTRA structure? You've got to use the _EXTRA mechanism.

JD

--

J.D. Smith /*\ WORK: (607) 255-6263
Cornell University Dept. of Astronomy */ (607) 255-5842
304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
Ithaca, NY 14853 */