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

Subject: Re: Keyword precedence
Posted by Mark Hadfield on Fri, 25 Aug 2000 04:03:36 GMT
View Forum Message <> Reply to Message

"David Fanning" <davidf@dfanning.com> wrote in message
news:MPG.140f8856ff9c2a24989bf1@news.frii.com...
> 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.

Thanks for your comments, David. It's nice to know that somebody reads my raves.

Why not

```
pro my_plot, COLOR=color, _EXTRA=extra
   if n_elements(color) gt 0 then $
      message, /INFORM, "I'm plotting this in color index 127 whatever you
say!"
   plot, COLOR=127, _EXTRA=extra
end
```

or my preference

```
pro my_plot, COLOR=color, _EXTRA=extra
```

```
    if n_elements(color) gt 0 then $
       message, "Don't touch the friggin' COLOR!"
    plot, COLOR=127, _EXTRA=extra
end
```

> 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 already have done in one or two places. It got clearer, in my opinion. But it's a matter of taste.

There's nothing stopping me from writing in the comments header:

COLOR: As for plot, but default is 127.

> 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.)

At the moment when I write

foobar, COLOR=0, _EXTRA={color:12}

then if foobar is a wrapper for some other function (and foobar itself doesn't have a COLOR keyword) I have to know whether foobar uses reference or value inheritance to know what the effect will be. This can't be right!

Of course I don't actually write the above command, not unless I'm messing about trying to understand inheritance mechanisms. But that's the effect of some perfectly reasonable wrapper designs.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Keyword precedence
Posted by davidf on Fri, 25 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

J.D. Smith (jdsmith@astro.cornell.edu) writes:
>
> Final synopsis: You want to play with the _EXTRA structure? You've got to use
> the _EXTRA mechanism.

JD, I know you are working on your thesis, so here is a tip for you. Put this kind of information in an Executive Summary at the TOP of your article or thesis, not at the very bottom. It will save a lot of head scratching and your thesis advisors will appreciate already understanding whatever it is you are tying to say. :-)

Cheers,

David

P.S. I have to say, you never disappoint me with your understanding of these issues. I really appreciate it. Thanks.

--
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

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 */

Subject: Re: Keyword precedence
Posted by Mark Hadfield on Sat, 26 Aug 2000 01:27:41 GMT
View Forum Message <> Reply to Message

"Mark Hadfield" <m.hadfield@niwa.cri.nz> wrote in message news:967252103.29393@clam-ext...

>
> With the third one, 12 is printed in every case! My interpretation: IDL's
> handling of keyword abbreviations is such that an abbreviated keyword
takes
> precedence over a non-abbreviated one, and this overrides the "fully
> by-reference first" rule.

This isn't quite true either. Further experimentation, left as an exercise
for the reader, establishes that the "fully
by-reference first" rule only applies when the conflicting keywords have
identical names. If the user-supplied keyword name to MGH_EXAMPLE_KEYWORDS
is shorter *or* longer than the default it's supposed to override, then the
user-supplied value wins every time.

I guess the point is that the reference inheritance mechanism only has to
choose between conflicting keywords when they are represented by
(case-insensitively) identical strings. Otherwise it just passes them both
through to the next level, where they are dealt with properly.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Keyword precedence
Posted by John-David T. Smith on Sat, 26 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield wrote:
>
> "J.D. Smith" <jdsmith@astro.cornell.edu> wrote in message
> news:39A6B82E.2533B5A@astro.cornell.edu...
>> .....
>
> Whew! Thanks for that explanation, JD. I think I understand it now.
>
> First an executive summary for David:
>
> * IDL's current behaviour regarding precedence of explicit keywords vs those
> passed through from above by inheritance is too inconsistent to be useful.
> It requires the caller to know details about inheritance mechanisms further
> down the chain, which is highly undesirable.
>
> Now a few questions/comments:
>
>> You can actually see this happening if you throw an "print,

>> arg_present(color)" into KEYWORDS_PRINT_COLOR.
>
> I did. It prints 0 in all 3 cases. What does this tell me? ARG_PRESENT is
> only supposed to return 1 if a named variable is supplied. I didn't supply
> one.

I guess I neglected to say if you do pass a named variable in addition to the
explicit _EXTRA={}.

>
>> Of course, only one of these can be used, and as it
>> happens, the fully by-reference variable is chosen.
>
> This is the crux, isn't it? Could & should this design choice be changed? If
> it were, would this lead to any other surprising results?

I commented briefly before to indicate that really this should not be changed,
but _REF_EXTRA should be changed to detect duplicate keyword passings and make
it an error. Without this error, the behaviour is confusing, I agree. RSI went
to far to ease acceptance of _REF_EXTRA, I belive.

>
> This is my main point and I could leave it there but I'll add another
> tidbit:
>
> I modified MGH_EXAMPLE_KEYWORDS & played with it a
> bit more. I've put the modified code on the WWW page and attached it to this
> message. Now the main routine accepts _EXTRA keywords from its caller. Its
> usage now reflects the intention of the whole exercise better. The idea is
> that we want to pass COLOR=0 through to the printing routine but let the
> user override it. (I hope nobody thinks I ever actually *write* code like
> "some_procedure, COLOR=0, _EXTRA={color:12}".)
>
> You might like to try the following set of calls:
>
> mgh_example_keywords
> mgh_example_keywords, COLOR=12
> mgh_example_keywords, COL=12
>
> No surprises with the first one: the default value of 0 is passed through
> and printed.
>
> With the second one we get the result that's being discussed, the user's
> value of 12 overrides the default and is printed except when the "reference
> wrapper" is involved in the calling chain, in which case 0 is printed.
>
> With the third one, 12 is printed in every case! My interpretation: IDL's
> handling of keyword abbreviations is such that an abbreviated keyword takes

> precedence over a non-abbreviated one, and this overrides the "fully
> by-reference first" rule. Which certainly suggests that the "fully
> by-reference first" rule is not cast in stone.

I too played with this and discovered the same property, which I negelected to mention to avoid furthing muddying the waters. I didn't want to imply there was any such "fully by-reference first" rule, but rather that there is no rule at all (and should in fact be an error, I believe). The progression of this unfortunate situation probably had to due with RSI's discovery that _REF_EXTRA is faster than _EXTRA, so why not use it for plain by value passing too, they thought (so long as you're not going to unintentionally modify the value). I think this was a slippery slope, since overriding a *value* is very different than overriding a *variable*. The former is relatively more straightforward. If RSI had stuck to a _REF_EXTRA used only for returning values up inherited keywords in chains of calls, we wouldn't have this ambiguity... it really doesn't make sense to override the *location in memory* associated with a given inherited keyword variable at runtime. But, now that the worms are out of the can, I suppose we'll have to deal with it. Since IDL can tell whether _REF_EXTRA is being used by value or by reference (or by some combination), it can use sensible rules for overriding. I'd reccommend:

1) 2 or more by-value and 0 by-reference(arg_present==0): Default to the standard _EXTRA rules for overriding and abbreviations (Longest match first).

2) 1 or more by-value and 1 by-reference: always pass the variable, by-reference. Useful for modifying a value *and* returning a result.

3) 0 or more by-value and more than 1 by-reference: generate an error. Not too burdening since you have to go out of your way to achieve this situation.

In any case, just beware of mixing your _EXTRA metaphors in the meantime.

JD

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 */

Subject: Re: Keyword precedence
Posted by Mark Hadfield on Sun, 27 Aug 2000 23:23:50 GMT

"John-David Smith" <jdsmith@astro.cornell.edu> wrote in message
news:39A874F3.D91EC14F@astro.cornell.edu...
> Mark Hadfield wrote:
>> "J.D. Smith" <jdsmith@astro.cornell.edu> wrote in message
>> news:39A6B82E.2533B5A@astro.cornell.edu...
>>> .....
>> Whew! Thanks for that explanation, JD. I think I understand it now.

Well, I didn't, but I'm getting there.

> ... I think this was a slippery slope, since overriding a
> *value* is very different than overriding a *variable*.
> The former is relatively more straightforward.
> If RSI had stuck to a _REF_EXTRA used only for returning values
> up inherited keywords in chains of calls, we wouldn't have
> this ambiguity... it really doesn't make sense to override the
> *location in memory* associated with a given
> inherited keyword variable at runtime...

You'll be relieved to hear that I've finally grasped this point. I was focussed entirely on passing information *inwards* through the chain of calls.

But before we give up entirely, let's consider the following again:

IDL> mgh_example_keywords, COLOR=12
% 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 by value:
COLOR{ 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: COLOR COLOR
% MGH_EXAMPLE_KEYWORDS_PRINT_COLOR: COLOR is 0

A key difference between the value wrapper and the reference wrapper is that the former passes only one keyword (which it sees as the structure {COLOR: 12}) down to the next level, whereas the the reference wrapper passes both, (which it sees as the string array ['COLOR', 'COLOR']).

Now that you've explained it, I see that that the reference wrapper retains both because either of them might point to a variable to be modified on output. By the way, in IDL 5.1 and before, the value wrapper also passed

both keywords as {COLOR: 0, COLOR: 12} and the color-print routine chose the first of them. But this was changed in 5.2 to the current behaviour.

So far this is all OK and in line with your explanation, but we agree there has to be a rule to choose which of the keywords the reference wrapper will pass to the color-print routine. By abbreviating the keyword name in the call to MGH_EXAMPLE_KEYWORDS (the "user keyword") and/or the name of the keyword specified inside MGH_EXAMPLE_KEYWORDS (the "default keyword") we can establish empirically that IDL behaves according to the following rules:

* Inside the reference wrapper the keywords are represented in the EXTRA string array in the order [<default>, <user>].

* If both keywords are represented by identical strings (case-insensitively) then the first of them (default) is passed to the color-print routine. If the keyword-strings are not identical then the second of them (user) is passed on.

The second rule is a pretty weird one, I think you'll agree. Maybe it's just a special case of some logical general rule--but I doubt it.

Re your proposed rules:

> 1) 2 or more by-value and 0 by-reference(arg_present==0): Default to the
> standard _EXTRA rules for overriding and abbreviations (Longest match first).
> 2) 1 or more by-value and 1 by-reference: always pass the variable,
> by-reference. Useful for modifying a value *and* returning a result.
> 3) 0 or more by-value and more than 1 by-reference: generate an error. Not too
> burdening since you have to go out of your way to achieve this situation.

First, as an aside to rule 1, I now think that abbreviation does *not* affect precedence.

But to the main point, assigning precedence based on arg_present would mean that this...

```
color = 0
mgh_example_keywords_reference_wrapper, COLOR=color, _EXTRA=extra
```

could give a different result from this...

```
mgh_example_keywords_reference_wrapper, COLOR=0, _EXTRA=extra
```

even though the programmer's intention in both cases might be identical. Scary!

I think the rule should be, where a choice has to be made between duplicate keywords, always choose the last one in the list.

Can we think of by-reference keywords as a set of named pipes through IDL memory space, linking different levels in the calling stack? Inheritance lets a bundle of these pipes pass through a routine's scope without the routine having to worry about their names. I think that the behaviour I am expecting is that each routine takes a bundle from its caller and can add additional pipes on the inside of the bundle (the beginning of the EXTRA list) or take pipes by name off the outside of the bundle (the end of the EXTRA list).

> In any case, just beware of mixing your _EXTRA metaphors in the meantime.

I don't think this has anything to do with mixing by-value and by-reference inheritance. (Proof: change _EXTRA to _REF_EXTRA in the procedure definition for MGH_EXAMPLE_KEYWORDS. By-value inheritance is not involved at all when the reference-wrapper is called but the behaviour is exactly the same.)

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Keyword precedence
Posted by John-David T. Smith on Mon, 28 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield wrote:

>>> Whew! Thanks for that explanation, JD. I think I understand it now.
>
> Well, I didn't, but I'm getting there.
>
>> ... I think this was a slippery slope, since overriding a
>> *value* is very different than overriding a *variable*.
>> The former is relatively more straightforward.
>> If RSI had stuck to a _REF_EXTRA used only for returning values
>> up inherited keywords in chains of calls, we wouldn't have
>> this ambiguity... it really doesn't make sense to override the
>> *location in memory* associated with a given
>> inherited keyword variable at runtime...
>
> You'll be relieved to hear that I've finally grasped this point. I was
> focussed entirely on passing information *inwards* through the chain of
> calls.

>
> But before we give up entirely, let's consider the following again:
>
> IDL> mgh_example_keywords, COLOR=12
> % 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 by
value:
> COLOR{ 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: COLOR COLOR
> % MGH_EXAMPLE_KEYWORDS_PRINT_COLOR: COLOR is 0
>
> A key difference between the value wrapper and the reference wrapper is that
> the former passes only one keyword (which it sees as the structure {COLOR:
> 12}) down to the next level, whereas the the reference wrapper passes both,
> (which it sees as the string array ['COLOR', 'COLOR']).
>
> Now that you've explained it, I see that that the reference wrapper retains
> both because either of them might point to a variable to be modified on
> output. By the way, in IDL 5.1 and before, the value wrapper also passed
> both keywords as {COLOR: 0, COLOR: 12} and the color-print routine chose the
> first of them. But this was changed in 5.2 to the current behaviour.
>
> So far this is all OK and in line with your explanation, but we agree there
> has to be a rule to choose which of the keywords the reference wrapper will
> pass to the color-print routine. By abbreviating the keyword name in the
> call to MGH_EXAMPLE_KEYWORDS (the "user keyword") and/or the name of the
> keyword specified inside MGH_EXAMPLE_KEYWORDS (the "default keyword") we can
> establish empirically that IDL behaves according to the following rules:
>
> * Inside the reference wrapper the keywords are represented in the EXTRA
> string array in the order [<default>, <user>].
>
> * If both keywords are represented by identical strings (case-insensitively)
> then the first of them (default) is passed to the color-print routine. If
> the keyword-strings are not identical then the second of them (user) is
> passed on.
>
> The second rule is a pretty weird one, I think you'll agree. Maybe it's just
> a special case of some logical general rule--but I doubt it.
>
> Re your proposed rules:
>

>> 1) 2 or more by-value and 0 by-reference(arg_present==0): Default to the
>> standard _EXTRA rules for overriding and abbreviations (Longest match
> first).
>> 2) 1 or more by-value and 1 by-reference: always pass the variable,
>> by-reference. Useful for modifying a value *and* returning a result.
>> 3) 0 or more by-value and more than 1 by-reference: generate an error.
> Not too
>> burdening since you have to go out of your way to achieve this situation.
>
> First, as an aside to rule 1, I now think that abbreviation does *not*
> affect precedence.
>
> But to the main point, assigning precedence based on arg_present would mean
> that this...
>
>   color = 0
>   mgh_example_keywords_reference_wrapper, COLOR=color, _EXTRA=extra
>
> could give a different result from this...
>
>   mgh_example_keywords_reference_wrapper, COLOR=0, _EXTRA=extra
>
> even though the programmer's intention in both cases might be identical.
> Scary!

I disagree. The programmers intentions are ambiguous. Does he wish a return value out? Is he passing a value in? Both? More scary is the notion of:

mgh_example_keywords_reference_wrapper, COLOR=color, _EXTRA=extra

putting a return value somewhere other than in the variable "color"... maybe not even on this level... maybe n levels up somewhere. This value-return paradigm is what _REF_EXTRA was intended for, and if we have to choose between scary by-value behaviour and scary by-reference behaviour, I choose the former. I think you are imagining cases in which you don't have control over the inheritance chain, and may not know you are using _REF_EXTRA. This is a danger I suppose, but it seems to an uncommon situation. Just as with normal positional parameters, the programmer must be sure to define in advance how each will be used: for input values, for return values, or for both. This affects their usage! You can't say:

mypro, retval

and expect

mypro, 1

to work the same if the first positional parameter is being used to return a

value... no matter what the user intends. Now, _REF_EXTRA is more complicated since more than two calling level are involved, but if you simply regard it as a tunnel through intermediate levels, the same rules apply.

> Can we think of by-reference keywords as a set of named pipes through IDL
> memory space, linking different levels in the calling stack? Inheritance
> lets a bundle of these pipes pass through a routine's scope without the
> routine having to worry about their names. I think that the behaviour I am
> expecting is that each routine takes a bundle from its caller and can add
> additional pipes on the inside of the bundle (the beginning of the EXTRA
> list) or take pipes by name off the outside of the bundle (the end of the
> EXTRA list).

This is a good analogy. Each individual pipe is accessible at each end only. One addition: all pipes originate at a given level (think of a bunch of flowers having their stems cut as a group), but may end any level up. This allows, for example, a chained series of GetProperty methods traversing the (class) inheritance tree up 10 levels to "stop-off" at each appropriate level and collect the relevant information. New pipes cannot originate mid-level, in contrast with _EXTRA. The power gained in this tradeoff is, of course, that _REF_EXTRA pipes flow in both directions!

>> In any case, just beware of mixing your _EXTRA metaphors in the meantime.
>
> I don't think this has anything to do with mixing by-value and by-reference
> inheritance. (Proof: change _EXTRA to _REF_EXTRA in the procedure definition
> for MGH_EXAMPLE_KEYWORDS. By-value inheritance is not involved at all when
> the reference-wrapper is called but the behaviour is exactly the same.)

Unfortunately your test is not sufficient proof, since _REF_EXTRA has hidden _EXTRA (read: by-value) functionality built inside of it (though in a perverse way). The way to see this is to put a "print, arg_present(color)" in your mgh_example_keywords_print_color procedure, and change the calling keyword of the reference wrapper to COLOR=col, where col is a local variable defined in mgh_example_keywords. Now you can change whether the by-reference or by-value form of _REF_EXTRA gets invoked by calling mgh_example_keywords with either no keyword/exactly "COLOR" or some abbreviation of "COLOR". Definitely not a good situation. If the rules I suggested were followed, you'd always get the reference variable here.

In any case, hopefully an RSI person or two will get the basic notion that this needs to be straightened up. And for those of you who have resolved never to include _REF_EXTRA in your programs, please be assured that this really affects only a very limited subset of cases of use.

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 */

Subject: Re: Keyword precedence
Posted by Martin Schultz on Mon, 28 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield and John-Cavid Smith had a long discussion in which
Mark wrote at some point:

>
> I think the rule should be, where a choice has to be made between duplicate
> keywords, always choose the last one in the list.
>

now for the slow-movers: does this mean that
wrapper, color=12, _Extra=e
and
wrapper, _Extra=e, color=12

should yield different results if there is a color tag in the
extra structure? I could see some benefits to this, but it would
alter the whole idea of keywords being position independent as
opposed to parameter arguments.

Although this may seem overly formal, I would argue that the
"convenience" gained by overriding keywords that are explicitely
set with values in the extra structure does not aid in writing
"clean" programs. If you are messing with any particular property
in your routines, this property should always be identified by
having its own keyword in my opinion. So, if you want a default
color, the procedure header should be something like:

```
pro wrapper, color=color, _EXTRA=e

  IF N_Elements(color) EQ 0 THEN color = 12
  plot, x,y, color=color, _EXTRA=e

end
```

and the "real" color keyword should be used (which I think and
hope it gets).

Conclusions (these tend to come near the end of an article ;-):

I agree with JD that IDL should be stricter and issue an error if
it finds more than 1 keyword of the same name in a _ref_extra
list, and perhaps it should even be possible to generate an error
if the two identical keywords appear in the _extra list (another
compile_opt flag?).

Cheers,
Martin

--

[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie
[[
[[Bundesstr. 55, 20146 Hamburg
[[
[[phone: +49 40 41173-308
[[
[[fax: +49 40 41173-298
[[
[[martin.schultz@dkrz.de
[[
[[[[[[[[[

Subject: Re: Keyword precedence
Posted by Craig Markwardt on Mon, 28 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"Mark Hadfield" <m.hadfield@niwa.cri.nz> writes:

> It's interesting that David Fanning and Martin Shultz have both recommended
> the following idiom for establishing overridable defaults
>
> pro my_plot, COLOR=color, _EXTRA=extra
>   if n_elements(color) eq 0 then color = 12
>   plot, COLOR=12, _EXTRA=extra
****   Ooops ^^   ****
> end
>
> This has the effect, unintended and normally irrelevant, that if the
> following call is made with the COLOR keyword set to an undefined variable
>
> my_plot, COLOR=color

>
> then this variable is set to 12 on output. It isn't too hard to imagine a
> situation (successive calls to different routines with different default
> colours) where this will bite an unwary programmer, though in several years
> of using this idiom I have seldom thought about this side-effect and have
> very seldom been bitten.

I have had a difficult time keeping up with this thread. Whew! I
often do my keyword passing with the following draconian but safe
technique.

```
pro my_plot, COLOR=color0, _EXTRA=extra
   if n_elements(color0) eq 0 then color = 12 $
   else color = floor(color0(0))
   plot, COLOR=color, _EXTRA=extra
end
```

COLOR0 is the value passed in, which is distinct from the value of the
local variable COLOR. I agree. It's ugly.

Craig

--

---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: Keyword precedence
Posted by davidf on Mon, 28 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

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

> Now to control (say) the X axis the caller just sets xaxis_properties equal
> to a structure containing the appropriate keyword:value pairs, e.g.:
>
> my_visualisation, XAXIS_PROPERTIES={notext:1, minor:0}
>
> Now this approach obviously relies on the "_properties" structures
> overriding the defaults.

This is how I've been configuring, for example, the PostScript
device for just about forever:

```
  ps_device_keywords = PSConfig()
  Set_Plot, 'PS'
```

```
   Device, _Extra=ps_device_keywords
```

It works great in a nice, controlled environment like
PSConfig, where I know *exactly* what keywords are
going to be in the structure coming back from it.
If I want to make sure a value is set (Color is an
obvious one), I can always force it:

```
   ps_device_keywords = PSConfig()
   ps_device_keywords.color = 1
   Set_Plot, 'PS'
   Device, _Extra=ps_device_keywords
```

But I have been much more wary of letting the user
create inherited structures, for some of the same
reasons you and JD so elegantly describe.

But I appreciate the discussion and the time everyone
has devoted to figuring this out. It has been extremely
helpful. Thank you.

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

Subject: Re: Keyword precedence
Posted by Mark Hadfield on Mon, 28 Aug 2000 22:07:03 GMT
View Forum Message <> Reply to Message

"Martin Schultz" <martin.schultz@dkrz.de> wrote in message
news:39AA2492.6ADB8E0@dkrz.de...
> Mark wrote at some point:
>
>> I think the rule should be, where a choice has to be made between
duplicate
>> keywords, always choose the last one in the list.
>
> now for the slow-movers: does this mean that
> wrapper, color=12, _Extra=e
> and
> wrapper, _Extra=e, color=12

>
> should yield different results if there is a color tag in the
> extra structure?

You mean different results from each other, don't you? Definitely not! The "list" here is the list of "extra" keywords inside wrapper (where--to reiterate--wrapper uses reference inheritance and doesn't have a color keyword itself). The information passed in via _extra always goes onto the end of this list. Position in the call doesn't matter (I checked).

> Although this may seem overly formal, I would argue that the
> "convenience" gained by overriding keywords that are explicitly
> set with values in the extra structure does not aid in writing
> "clean" programs. If you are messing with any particular property
> in your routines, this property should always be identified by
> having its own keyword in my opinion. So, if you want a default
> color, the procedure header should be something like:
>
> pro wrapper, color=color, _EXTRA=e
>
> IF N_Elements(color) EQ 0 THEN color = 12
> plot, x,y, color=color, _EXTRA=e
>
> end
>
> and the "real" color keyword should be used (which I think and
> hope it gets).

David Fanning said the same thing. I see your point, but I think I still favour the overriding form:

```
 pro wrapper, _EXTRA=e
   plot, x,y, color=12, _EXTRA=e
end
```

because it's shorter (much shorter if several properties are overridden) and therefore clearer. However I've been happily writing code in the style you advocate for the last couple of years, so it can't be all that bad.

Actually my reason for wanting to rely on keyword precedence is a little more complicated, and a little more compelling, than what I described in my previous posts. For the last week--while I haven't been writing long posts on keyword precedence--I have been preparing figures and animations for a conference presentation. I have been using widget applications that I wrote some time ago to visualise hydrodynamic model output. These applications create various object graphics elements, e.g.:

pro my_visualisation

```
   theview = new_view(UNITS=2, DIMENSIONS=[10,10])
   thexaxis = new_axis(DIRECTION=0, TICKFORMAT=...)
   theyaxis = new_axis(DIRECTION=1, TICKFORMAT=...)
   ; Add the axes to a model & add that to the view
   ; Get data
   thesurface = new_surface(STYLE=2, DATAX=...)
   ; add the surface to the model
   new_window, GRAPHICS_TREE=theview, RETAIN=2 ; (sorry Randall)
end
```

where the "new_" functions and procedures are wrappers of some sort for the object-creation functions.

This has been fine for visualisation purposes, but for presentation I wanted to tweak the results: adjust the size and colours for the output medium, get rid of extraneous tick labels and reduce the size of the margins, stuff like that. But how to get all that information into the application without adding a huge number of keywords to the routine? I found the solution in an example provided with IDL 5.4 beta, but I see that it's also used by the LIVE_STYLE routine: for each element foo add a single keyword, foo_properties, thus:

```
pro my_visualisation $
   , VIEW_PROPERTIES=view_properties $
   , XAXIS_PROPERTIES=xaxis_properties $
   , YAXIS_PROPERTIES=yaxis_properties $
   , SURFACE_PROPERTIES=surface_properties $
   , WINDOW_PROPERTIES=window_properties

   theview = new_view(UNITS=2, DIMENSIONS=[10,10], _EXTRA=view_properties)
   thexaxis = new_axis(DIRECTION=0, TICKFORMAT=...,
_EXTRA=xaxis_properties)
   theyaxis = new_axis(DIRECTION=1, TICKFORMAT=...,
_EXTRA=yaxis_properties)
   ; Add the axes to a model & add that to the view
   ; Get data
   thesurface = new_surface(STYLE=2, DATAX=..., _EXTRA=surface_properties)
   ; add the surface to the model
   new_window, GRAPHICS_TREE=theview, RETAIN=2, _EXTRA=window_properties
end
```

Now to control (say) the X axis the caller just sets xaxis_properties equal to a structure containing the appropriate keyword:value pairs, e.g.:

```
my_visualisation, XAXIS_PROPERTIES={notext:1, minor:0}
```

Now this approach obviously relies on the "_properties" structures overriding the defaults. When I first tried this I found that the

xaxis_properties keyword was not working and that this was because I had happened to use inheritance by reference in new_axis. The rest (as they say) is history...

> I agree with JD that IDL should be stricter and issue an error if
> it finds more than 1 keyword of the same name in a _ref_extra
> list, and perhaps it should even be possible to generate an error
> if the two identical keywords appear in the _extra list (another
> compile_opt flag?).

Well, I think that raising an error is a little harsh and that, as I've said, some form of consistent precedence rule would be better. But raising an error is certainly better than IDL's current behaviour, which is silently inconsistent.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Keyword precedence
Posted by Mark Hadfield on Mon, 28 Aug 2000 23:32:06 GMT
View Forum Message <> Reply to Message

"J.D. Smith" <jdsmith@astro.cornell.edu> wrote in message news:39AA8AFE.CDBAB7E6@astro.cornell.edu...

> ...Just as with normal
> positional parameters, the programmer must be sure to define in advance how each
> will be used: for input values, for return values, or for both. This affects
> their usage!...

It's interesting that David Fanning and Martin Shultz have both recommended the following idiom for establishing overridable defaults

```
pro my_plot, COLOR=color, _EXTRA=extra
   if n_elements(color) eq 0 then color = 12
   plot, COLOR=12, _EXTRA=extra
end
```

This has the effect, unintended and normally irrelevant, that if the following call is made with the COLOR keyword set to an undefined variable

```
my_plot, COLOR=color
```

then this variable is set to 12 on output. It isn't too hard to imagine a situation (successive calls to different routines with different default colours) where this will bite an unwary programmer, though in several years of using this idiom I have seldom thought about this side-effect and have very seldom been bitten.

My point: in many situations IDL programmers are pretty relaxed about whether values are modified on output because it has no effect on how their programs operate. As far as possible the language should avoid punishing them for this.

> ...More scary is the notion of:
>
> mgh_example_keywords_reference_wrapper, COLOR=color, _EXTRA=extra
>
> putting a return value somewhere other than in the variable "color"... maybe not
> even on this level... maybe n levels up somewhere.

Yes, that would be the effect of my proposals for precedence. And it is the current situation (which acts exactly the way I am proposing except in a specific, albeit common, case). I think you have to accept that by putting _EXTRA or _REF_EXTRA in your code you are passing power, and responsibility to the next level up.

> ...I
> think you are imagining cases in which you don't have control over the
> inheritance chain, and may not know you are using _REF_EXTRA.

Yes. And I was bitten in a case where I did have control over the inheritance chain but had forgotten which particular mechanism I had used.

> In any case, hopefully an RSI person or two will get the basic notion that this
> needs to be straightened up. And for those of you who have resolved never to
> include _REF_EXTRA in your programs, please be assured that this really affects
> only a very limited subset of cases of use.

Agreed.

I can't resist adding another tidbit: CALL_PROCEDURE passes keywords through in both directions, but it doesn't behave like my "reference wrapper". It always gives precedence to extra keywords, irrespective of whether the names match exactly.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Keyword precedence
Posted by Martin Schultz on Tue, 29 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
>
> "Mark Hadfield" <m.hadfield@niwa.cri.nz> writes:
>
>> It's interesting that David Fanning and Martin Shultz have both recommended
>> the following idiom for establishing overridable defaults
>>
>> pro my_plot, COLOR=color, _EXTRA=extra
>>   if n_elements(color) eq 0 then color = 12
>>   plot, COLOR=12, _EXTRA=extra
> ****   Ooops ^^   ****
>> end
>>
>> This has the effect, unintended and normally irrelevant, that if the
>> following call is made with the COLOR keyword set to an undefined variable
>>
>> my_plot, COLOR=color
>>
>> then this variable is set to 12 on output. It isn't too hard to imagine a
>> situation (successive calls to different routines with different default
>> colours) where this will bite an unwary programmer, though in several years
>> of using this idiom I have seldom thought about this side-effect and have
>> very seldom been bitten.
>
> I have had a difficult time keeping up with this thread. Whew! I
> often do my keyword passing with the following draconian but safe
> technique.
>
> pro my_plot, COLOR=color0, _EXTRA=extra
>   if n_elements(color0) eq 0 then color = 12 $
>   else color = floor(color0(0))
>   plot, COLOR=color, _EXTRA=extra
> end
>
> COLOR0 is the value passed in, which is distinct from the value of the
> local variable COLOR. I agree. It's ugly.
>

> Craig
>
> --
> -- --------------
> Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
> Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
> -- --------------

No, it's not ugly, it's utmost correct ;-) This is what I do whenever I get caught by the situation that Mark points out - once I discover that my return value is changed *and* that this leads to undesired consequences (which most often it does not, rather the opposite), then I change color to color0 or whatever. To give you an example, where I rely on setting the keyword value if it is undefined:

```
pro whatever, filename

   read_data, filename, data
   if n_elements(data) eq 0 then return
   print, ' Read data from file '+filename
   plot,data.x,data.y
end
```

Here, read_data will receive an undefined value if you pass no filename to whatever. It then sets filename to a default search pattern (e.g. '*.nc') and calls the dialog_pickfile to select a file. The name of the file that is selected is stored in filename for future reference (in this example, the print statement). Alternatively, if you pass a fully qualified filename, the file is opened with no further questions, or if you know a little more about the file, you can pass a search mask like '/home/myself/data/global*.nc' that will be used as filter for dialog_pickfile. I find that this works very nicely and veeeery conveniently in 99% of all cases - just occasionally if I want to loop over several files at once and have them all "hand-picked", then I will have to re-initialize filename each time.

Cheers,
Martin

--
[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie
[[
[[Bundesstr. 55, 20146 Hamburg

[[
[[phone: +49 40 41173-308
[[
[[fax: +49 40 41173-298
[[
[[martin.schultz@dkrz.de
[[
[[[[[[[[[[

Subject: Re: Keyword precedence
Posted by Martin Schultz on Tue, 29 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield wrote:
>
> for each element foo add a single keyword,
> foo_properties, thus:
>
> pro my_visualisation $
> , VIEW_PROPERTIES=view_properties $
> , XAXIS_PROPERTIES=xaxis_properties $
> , YAXIS_PROPERTIES=yaxis_properties $
> , SURFACE_PROPERTIES=surface_properties $
> , WINDOW_PROPERTIES=window_properties
>
> theview = new_view(UNITS=2, DIMENSIONS=, _EXTRA=view_properties)
> thexaxis = new_axis(DIRECTION=0, TICKFORMAT=...,
> _EXTRA=xaxis_properties)
> theyaxis = new_axis(DIRECTION=1, TICKFORMAT=...,
> _EXTRA=yaxis_properties)
> ; Add the axes to a model & add that to the view
> ; Get data
> thesurface = new_surface(STYLE=2, DATAX=..., _EXTRA=surface_properties)
> ; add the surface to the model
> new_window, GRAPHICS_TREE=theview, RETAIN=2, _EXTRA=window_properties
> end
>
> Now to control (say) the X axis the caller just sets xaxis_properties equal
> to a structure containing the appropriate keyword:value pairs, e.g.:
>
> my_visualisation, XAXIS_PROPERTIES={notext:1, minor:0}
>
> Now this approach obviously relies on the "_properties" structures
> overriding the defaults. When I first tried this I found that the
> xaxis_properties keyword was not working and that this was because I had
> happened to use inheritance by reference in new_axis. The rest (as they say)
> is history...

If you follow this approach, you might almost as well pass the structure explicitely instead of relying on the _extra passing mechanism. Things like this prompted me a while ago to write my little ChkStru routine, where you test (a) if the thing you test is a structure, and (b) if it contains what you are looking for.
An example for setting axis properties could look like:

```
   IF ChkStru(xaxis_properties, 'DIRECTION') THEN
thexaxis->SetProperty,  $
       direction=xaxis_properties.direction
   IF ...
```

But I realize, this may get a bit lengthy from time to time ;-)

Thanks for the thoughtful discussion,

Martin

--
[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie
[[
[[Bundesstr. 55, 20146 Hamburg
[[
[[phone: +49 40 41173-308
[[
[[fax: +49 40 41173-298
[[
[[martin.schultz@dkrz.de
[[
[[[[[[[[[