Subject: Re: Keyword precedence
Posted by Mark Hadfield on Sun, 27 Aug 2000 23:23:50 GMT
View Forum Message <> Reply to Message

"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