Subject: Re: _Ref_Extra : BUG? (in Win2K 55b) corrected test file
Posted by John-David T. Smith on Fri, 02 Nov 2001 20:06:55 GMT
View Forum Message <> Reply to Message

Martin Downing wrote:
>
> ----- Original Message -----
> From: "Mark Hadfield" <m.hadfield@niwa.cri.nz>
>>
>> So the routine that's crashing your machine (and mine) is as follows
>> (shortened)...
>>
>> pro test_ref_extra1,_REF_EXTRA=e
>> test_ref_extra2, _REF_EXTRA=e ; Here be dragons!
>> end
>>
>> This isn't merely bad practice, it's wrong. The _REF_EXTRA keyword is only
>> used in routine definitions. When *calling* a routine, the acceptable
> forms
>> are "_EXTRA" and (as of 5.5) "_STRICT_EXTRA".
>>
>> Still, it shouldn't crash IDL. It should be detected as a syntax error.
>>
> Hi Mark
>
> It seemed logical to me that if arguments have to be passed through many
> preprocessing routines to a later routine, that the best way would be by
> reference to avoid copying (as would be the case if the Keyword was
> specified), if the final routine insists on a local copy (by value) why
> should that matter?
> Well I guess I just didnt understand the manual (no change there!). Are you
> saying that this is determined purely by the definition, and that calling
> with _REF_EXTRA as below
>> test_ref_extra2, _REF_EXTRA=e ; Here be dragons! (indeed)
> is supposed to be a syntax error? That was just not clear to me from the
> help (and for that matter the programmer who wrote the syntax parser!!!)
> I am concerned as to which causes passing by reference/value, the definition
> or the call?
> This is almost as horrible as C++ inheritance simple in principle but boy
> can some strange things happen when things get complicated - for now I guess
> will have to stick to _extra which at least is easy to understand :(
>
> roll on coffee time!

As someone unaturally familiar with the intimate details of _REF_EXTRA
(having raised the complaint which led to its creation), let me give a
bit of background in defense of this (admittedly confusing) behavior.

I initially envisioned changing the _EXTRA mechanism to support natively the default passing behavior of arguments (normal or keyword), which had been in existence in IDL since time immemorial: everything would be passed by reference, except indexed arrays, structure field members, and a few other by-value items known well to anyone more than a casual IDL user. I reasoned that this default-by-reference behavior was very familiar to all IDL programmers, and should not provide any technical or conceptual barriers for extension to *inherited* keyword parameters... in fact, it was how it should have been implemented to begin with.

The last comment notwithstanding, there was already in place a specific by-value inherited keyword passing mechanism which provided the inherited arguments in the form of an _EXTRA structure to intermediate routines. People started to rely on the specifics of this _EXTRA structure (yes, this includes me), to do all kinds of interesting and/or perverse things. The fact that this structure exists as it does, and is used as it is, meant that no single modification could simultaneously provide by-reference behavior and access to the old, familiar _EXTRA structure. So, they settled on an uneasy compromise: an additional keyword.

However, converting routines to use by-reference keyword inheritance would then be very awkward: a special-purpose keyword (_REF_EXTRA), would need to be used everywhere. You'd have to keep track of all arbitrary chains of inheritance to make sure you keep the correct "flavor" all along. The RSI developers discovered a trick, however: they could isolate the required change only to the routine definition, and allow all the calling routines (including existing code) to continue to use _EXTRA as-is. And some routines might just do both! Yes, you can pass on a by-reference inherited keyword set to a by-value inheriting routine: the conversion is automatic. (And no, you can't go the other way!).

Had it been done this way from the beginning, we'd have only one mechanism, and it would look and feel like the other argument passing mechanisms in IDL. Such is not the case, and what you see is the compromise which resulted.

So, bottom line rule of thumb: always use _EXTRA in your calls (or at least, never use _REF_EXTRA -- see below). The fact that _REF_EXTRA *could* be used in calls was something of a slip-up, reflecting the later inclusion of the "only-change-in-one-place" feature. The fact that using it crashes IDL remains, of course, a bug.

JD

P.S. With IDLv5.5, yet another _EXTRA keyword appears to muddy the waters: _STRICT_EXTRA. Be assured that it is *very* different in

flavor, and as far as I can discern from the single-paragraph "What's New" entry, is meant to be used only in routine *calls*, and never in definitions.

_STRICT_EXTRA simply weeds out extraneous keywords, and signals an error if they occur. This is useful to guard against simple misspellings sneaking in, and not manifesting themselves, e.g.:

myroutine,DOMYNICECALCULATION=1

vs.

myroutine,DOMENICECALCULATION=1

neither would raise an error if called through:

```
pro myroutine, _EXTRA=e
  mycalc,_EXTRA=e
end
```

even though presumably mycalc doesn't recognize the second one. You may not get the result you expect, or worse, you may get a result you erroneously believe, blissfully unaware of your fat fingers. With _STRICT_EXTRA you can check. Let's just hope it works for both by-value and by-reference inherited keywords.
