Subject: Re: New Object Method Invocation Syntax Brokenness
Posted by Paul Van Delst[1] on Mon, 16 May 2011 21:03:36 GMT
View Forum Message <> Reply to Message

Crikey. I haven't yet verified any weird behaviour of the sort you describe in our IDL codes, but thanks for the heads up.

Your post has been saved to my desktop (in 12years, I've only done that 4 times) for immediate reference.

End-of-last year we went through a relatively big transformation of regular old functional code to an object library.
The code in question is the start of our processing chain so it's, you know, sorta important that that library work
correctly. And reliably so. In the background. The plausible-but-wrong answer is what keeps me up at night.

If things start going pear shaped, at least I have a prototype for conversion of said IDL object library to Fortran2003....

<Sigh>

cheers,

paulv

JDS wrote:
> A few years back, we had long discussions about the trouble the new "dot" syntax for method invocation would bring
> about. I've just stumbled across a new and unexpected case. Not only does IDL 8 interpret "." as equivalent to
> "->", in certain uses it also goes the OTHER WAY, recasting "->" as "." and overriding a method invocation with
> structure variable dereferencing, *even when the arrow operator is used*. This is a major issue for any pre-existing
> class which contains a method-function and structure member with identical names, which is quite common.
>
> Consider this simple class:
>
> pro DOT_ARROW::try ;compile_opt idl2 print,"GOT: ",self->item([1,2,3,4]) end
>
> function DOT_ARROW::item, p return, size(p,/DIMENSIONS) end
>
> pro dot_arrow__define st={DOT_ARROW,$ item:0.0} end
>
> IDL> d=obj_new('dot_arrow') IDL> d->try GOT: 0.00000 0.00000 0.00000 0.00000
;; RUNS without

> ERROR, but is WRONG!!!
>
> Now with IDL2 enabled (or in IDL 7):
>
> IDL> d->try GOT: 4 ;; RIGHT!!!
>
> IDL 8 has gone one step further than introducing a new ambiguity between "." and "->" in the name of Python/JS-esque
> cosmetics, it's completed that ambiguity -- essentially enforcing it on us -- by making it work both ways. In
> essence, IDL8 is interpreting:
>
> self->item([1,2,3,4])
>
> as
>
> self.item[[1,2,3,4]]
>
> (no indexing error since you can't go out of bounds with an indexing vector by default). If I say "->", I mean
> invoke a method, no matter what. However, as new programmers adopt "." for method invocation, this ambiguity will
> never go away.
>
> And this is not a simple matter of troublesome syntax errors. What's really scary about this is, there are plenty of
> imaginable cases in which a->b(c) is interpreted by IDL8 in a completely different way from earlier versions of IDL,
> yet it can run through without error, silently corrupting your output. There is no warning against or detection of
> the use of identical names for a method-function and a class structure variable.
>
> Chris Torrence noticed this ambiguity could cause trouble while IDL 8 was being designed; I don't recall what the
> final decision was. But in any case, my understanding was this would be a new ambiguity, affecting new code which
> uses the dot operator for method invocation. Never was it discussed that IDL 8 would render inoperable (or worse,
> operable but incorrect) existing, functioning code by re-defining dereferencing post facto. I can scarcely see how
> this marketing-driven syntax change was worth it.
>
> JD