Subject: Re: New Object Method Invocation Syntax Brokenness
Posted by JDS on Mon, 16 May 2011 22:35:02 GMT
View Forum Message <> Reply to Message

My understanding is compile_opt idl2 alleviates this particular undesirable feature of the newly ambiguated syntax, though I'm not eager to go back and insert it for thousands of methods, or develop methods to discover potential brokenness. I suppose talk of enabling idl2 by default died off; I'd be in favor of it. It would break old code, but in a well-controlled and presumably well-advertised way.

By the way, this behavior seems to have been introduced in 8.0.1. I believe the short lived IDL 8.0 respected the arrow operator.

I discovered it when attempting to debug a piece of older formerly functioning object code. I set a breakpoint, and tried to step into the relevant function method that was returning nonsense (named, in my case 'Width'). I could never get IDL to step into the method, which I eventually realized was because IDL was parsing the method call:

```
width=self->Width(pt)
```

as the simple array dereference:

```
width=self.width[pt]
```

In my case, this threw an error, because width was now a two-element vector, and it was supposed to be a scalar. You can imagine many cases in which no error whatsoever is thrown, and, perhaps less commonly, cases where "plausible-but-wrong" values are returned, which, without some diligence, would simply never be discovered.

In principle, IDL could 1) always respect the arrow operator as a method call (in which case only the foolhardy would use ".") and/or 2) detect these conflicts at compile-time or first object-instantiation time, and at least tell the user something useful before arbitrarily picking one interpretation. But neither of those solutions puts the cat back in the bag.

JD