Subject: Re: New Object Method Invocation Syntax Brokenness
Posted by penteado on Tue, 17 May 2011 17:37:45 GMT
View Forum Message <> Reply to Message

On May 17, 1:17 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
> In my version of IDL 8.0, your example follows, but my example does not. I.e. self->item([1,2,3,4]) still correctly calls the method in IDL 8.0, but no longer does it in 8.0.1. My take is that IDL 8 introduced the syntax ambiguity ("->" and "." interchangeable), then IDL 8.0.1 reversed the precedence in ambiguous cases, now favoring structure/class variable dereferencing over method calling.
>
> BTW, the documentation mentions this interchangeability in the context of method invocation:
>
> "Beginning with IDL 8.0, you can use the . and -> forms of the operator interchangeably; they are
> equivalent."
>
> In a sense, this bug cannot be fixed, since it is inherent in the choice to make "." mean two things. Unless idl2 is in force, you simply cannot know what I mean by:
>
> d=a.b(c)
>
> even (in the case of "b" being both a method name and a class variable), at runtime! ITT could certainly patch "->" to avoid breaking old code, but new code will always have this potential for silent brokenness (unless people shun "."). What's interesting is the main concern was putting off new users with meaningless syntax error messages. This example shows a much more problematic issue arises.
>
> One possible fix would be to make array subscripting usage following the "." operator implicitly require brackets "[]". This would break old code like c=a.b(4), but leave intact other uses of parentheses for array indexing. I'd call this a "partial idl2 requirement". It still leaves more than a year's worth of IDL versions in use silently breaking old code.

This gets confusing easily. If I may expand a bit on JD's example, to compare the dot with the arrow:

```
;++++++++++++++++++++++++
;  failure__define.pro
pro Failure::Explode_with_arrow
  prevent=[1b]  ;; Alway prevent the explosion, no matter what!
  if self->Prevent_Explosion(prevent) then $
     print, 'Explosion averted, go in peace.' $
  else print, 'Field used: !BOOM! Nuclear war initiated.'
end

pro Failure::Explode_with_dot
  prevent=[1b]  ;; Alway prevent the explosion, no matter what!
  if self.Prevent_Explosion(prevent) then $
```

```
    print, 'Explosion averted, go in peace.' $
  else print, 'Field used: !BOOM! Nuclear war initiated.'
end

function Failure::Prevent_Explosion, confirm
  print,'Got into the method, instead of using the field'
  return, keyword_set(confirm[0])
end

pro failure__define
  st={FAILURE,$
      prevent_explosion: 0b}
end
;++++++++++++++++++++++++
```

The results:

In IDL 7.1.1, which is how things should still be, to keep compatibility with old code:

```
IDL> print,!version
{ x86_64 linux unix linux 7.1.1 Aug 21 2009      64      64}
IDL> f=obj_new('failure')
IDL> f->explode_with_arrow
Got into the method, instead of using the field
Explosion averted, go in peace.
IDL> f->explode_with_dot
Field used: !BOOM! Nuclear war initiated.
```

In 8.0 it is broken one way:

```
IDL> print,!version
{ x86_64 linux unix linux 8.0 Jun 18 2010      64      64}
IDL> f=obj_new('failure')
% Compiled module: FAILURE__DEFINE.
IDL> f->explode_with_arrow
Got into the method, instead of using the field
Explosion averted, go in peace.
IDL> f->explode_with_dot
Got into the method, instead of using the field
Explosion averted, go in peace.
```

In 8.1 it is broken the other way:

```
IDL> print,!version
{ x86_64 linux unix linux 8.1 Mar  9 2011      64      64}
IDL> f=obj_new('failure')
IDL> f->explode_with_arrow
```

Field used: !BOOM! Nuclear war initiated.
IDL> f->explode_with_dot
Field used: !BOOM! Nuclear war initiated.

So I would say that what needs to be fixed now is only to make the arrow mean, always, method invocation. It should never be accepted for a structure field. The resolution of the ambiguity with the dot got right in 8.0.1: a field should take precedence over a method.
