Subject: Re: IDL 8.0 compile_opt changes
Posted by Paul Van Delst[1] on Thu, 07 Jan 2010 16:35:08 GMT
View Forum Message <> Reply to Message

Chris Torrance wrote
> The primary change is the use of the dot "." for object method calls.
> The use of the "." for method calls is now industry standard, for
> example in languages such as Java, Python, etc.

So? You open Pandora's box simply because newer languages do it a particular way?

And there are languages that don't use "." for method calls or structure dereference: e.g. Fortran90/95/2003/2008

In Fortran, "%" does that. (2003+ for the object method invocation)

I remember when Fortran90 was still being adopted and there was much wailing, gnashing of teeth, and wringing of hands about the fact that the Fortran standard used "%" rather than "." (the latter being used in some earlier DEC extentions) for structure component dereferencing.

Well, you know, we all got over it. :o)

BTW, the other change with Fortran that caused a lot of consternation (*.f == fixed format, *.f90 == free format) still causes a fair amount of head scratching so I don't recommend the file extension approach.

My PO is if people get annoyed when they see
  child = p->Get(index)
because they think it should look like
  child = p.Get(index)
they need to gain perspective about what is *really* important.

Is there a less nebulous reason for ITTVIS looking to replace "->" with "."? Otherwise, JD's points below highlight the sanity that should be adopted.

And, I would prefer ITTVIS's time be spent doing useful stuff like providing additional functionality rather than generating make-work for themselves and their users. E.g.:

- Improved interpolation functions for 1,2,3,..,N-dimensional data would be great.
- More options for integrating tabulated data too (e.g. being able to select the interpolation method, or integration technique (simspon's, boole's, etc.).
- A unit testing capability (maybe from Mike Galloy via his mgunit?)
- object widgets (maybe from David Fanning via Catalyst?)
- easy PS output from iTools like we have for Direct Graphics.
- metaprogramming capabilities (if you want IDL to be more like Python or ruby, that's where you should be spending your time) so I don't have to waste time writing boilerplate get_property and set_property methods for objects.

- etc..

Anyway...

cheers,

paulv

JD Smith wrote:
> While I long ago converted to [] for array subscripting for the
> reasons most succinctly expressed by Wayne, I have mixed feelings
> about converting the method invocation operator from '->' to '.'
> purely for cosmetic reasons. Problems I see with this:
>
> 1. Overloading structure dereference and method invocation breaks the
> ability to semantically parse a.b.
>
> With this overloading, it is impossible to determine if 'a.b' is a
> method procedure call, or a structure field dereference, *except at
> runtime in the IDL interpreter*! For example, in IDLWAVE you can a->b
> [M-Tab] and have all "b..." procedure methods completed, or c=a->b[M-
> Tab] for function methods. Though I'm not sure, I suspect this would
> have a similar impact on the Workbench: loss of edit-time or shell-
> interaction-time differentiation among structure fields/object methods/
> etc through direct inspection of the source.
>
> 2. It will *not* be immediately obvious, or *ever* obvious, to a human
> observer that code which includes statements like a.b(c) must be
> compiled with IDL8.0 to run correctly. Consider a simple function
> found deeply buried on disk:
>
> function do_something, input
> return, input.something_else(1)
> end
>
> This small function would compile equally in IDL 8 and IDL <8, but
> have a totally different meaning depending on what INPUT was passed in
> which version. You can make compile_opt idl2 the default in IDL 8,
> but this does little to relieve this issue, since older versions of
> IDL will compile this fine (and choke horribly if passed an object).
>
> 3. IDL is not Python. IDL enforces strict encapsulation of object
> data, i.e. all object data must be accessed through a method (except
> within the object's methods themselves). Python has no object data
> encapsulation. In Python it is natural to mix method invocation with
> data access. In IDL this only occurs only in an object's own
> methods. Which is clearer?

>
> self->limit, self.limit
>
> self.limit, self.limit
>
>
> Just my $1D-2. (BTW, I think negative indexing sounds great!).
>
> JD
