Subject: Re: IDL 8.0 compile_opt changes
Posted by penteado on Tue, 05 Jan 2010 23:54:32 GMT
View Forum Message <> Reply to Message

On Jan 5, 8:41 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
> 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.

That is a good point that I had not considered. I only noticed that
with the assumed idl2 option, there would be no ambiguity in complete,
correct lines. But as you point out, there would be in the middle of
writing a line. Though, as you indicate in (3), the ambiguity would
only occur between methods and fields of self. In other cases, the
interpreter should know whether you typed a structure or an object.

>
> 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).

This problem that would be solved with a new extension, which also
avoids the backwards incompatibility issues.

>
> 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

Though it is a bit easier to understand the first one, there is no ambiguity, and I still find it clear from the syntax the meaning of the second line.

But (3) and (1) are good points to consider. Is it better to keep the different operators, which is in itself a clearer way, or to use the more universally adopted overloading convention? I see no obvious answer.

Contrary to C++, Java and Python, the flat namespace and case-insensitivity already make it trickier to pick names in IDL, so it may be even more important to differentiate between methods and fields with the operator.
