Subject: Re: IDL 8.0 compile_opt changes
Posted by H. Evans on Sat, 19 Dec 2009 10:00:03 GMT
View Forum Message <> Reply to Message

As a user that migrated from PVWave in the mid '90s, I still have many routines that use the older parenthetical array syntax (in some instances so they work on both systems). Slowly I'm migrating them to the better, less ambiguous bracket syntax. This is the sort of unilateral impetus that I'd need to finally finish off the job. ;-)

So, I'd say to go for the square bracket as the default syntax and confine the parentheses to the dustbin of history. And if nothing else, make the default for any loop index variable a 32 bit variable.

I think of these things like the millennium bug work: you need a large initial and short duration investment to make the software less prone to bugs, less ambiguous, less confusing and generally better. After a couple of years/releases, this change will become the new status quo and the users will have adapted. The trick is to make their migration path quick, easy and smooth. Is there a source code analyser/profiler that can go through a routine identifying which elements in the routine are functions and which are arrays? This would make it easier for us to go through less well known code and bring it up to date. Or perhaps while the program is running, it could parse and update the source files automatically?

Since the introduction of the square bracket syntax, IDL has been on the cusp of a paradigm change in the language (much like the Fortran 90 standard was a major change to the DecFortran standard[*]). Now's probably a good time to finish it off. Didn't we go through a similar thing with the change in the CALL_EXTERNAL string definitions a few years back with the move to 64 bit?!

What is most important, if you do make significant changes to the basic language syntax, then why not at the same time update the language/filenaming system to allow for an easier upgrade path for future updates so as to preserve backward compatibility while at the same time allow for future extensions. I see this as an issue of meta-data surrounding the individual source files - the version of the language the procedure/function was written for is not included in the file by default and so it's required that the compiler/interpreter make some judgement call on incomplete information. Now is the time to remedy this underlying problem (and something that exists with other languages).

As for the comments concerning the needs for different versions of IDL installed, we currently are able to switch back and forth from IDL v4.7 (?) to v7.1 on the same system - we are obliged to maintain older versions for software that has been frozen.

On Dec 18, 10:51 pm, Chris Torrence <gorth...@gmail.com> wrote:
> Possible solutions:
>
> 1. Change the default to be "compile_opt idl2", add a new "compile_opt
> idl1" to restore the existing behavior, and require users to retrofit
> existing code.

This gets my vote. It's a one off and forces an update to the new/
better syntax. Combine it with #5 to solve future version ambiguities.

> 2. Only change the default behavior for arrays within structures. Add
> a new "compile_opt allow_parens_with_structure_fields" (obviously that
> would not be the name). Existing users would still need to retrofit
> code, but not as much as #1.

I don't like this option. It permits a mixing of syntax, rather than a
simple and clear and standard syntax. It just provides another
variation to have to code around, and adds complexity.

> 3. Do #1 or #2, but also add a global preference for the compile_opt.
> By default it would be "idl2", but users could change it (we would
> need an "idl1" to turn off the new behavior). This is bad if the user
> wants to use existing libraries, but also wants to use the new method
> calls.

Global configurable defaults would cause confusion, for the reason
you've mentioned. The existing libraries would still have to be
updated by adding compile_opts to them to ensure they are run with the
correct version assumptions.

>
> 4. Do nothing, require users to use "compile_opt idl2" if they want
> the new "dot" methods. This is bad for new users, as they will get
> strange syntax errors and will not understand why.

No, it's time to embrace the not too distant past. ;-)

>
> 5. Add a new ".prx" extension (name TBD). If you have an existing
> ".pro" file then the defaults remain unchanged. The new ".prx" would
> default to "compile_opt idl2". This solves the problem, but might
> cause a "split" in the code base and confusion for the users.

I do like this option. It would be particularly nice to split the '@'
run scripts from the '.run' scripts (or have I missed something in the
past years in the wilderness). Explaining to new users the difference

between the '.pro' that is run one way and the '.pro' file that is run
the other becomes tedious as it isn't always obvious when the file is
one or the other.

Choose the new file extension carefully, though. At some time in the
future another syntactic update will occur, if you get it right now,
then it makes it easier next time, e.g. '.pr2', allowing for '.pr3' at
some point in the future.

> 1. How much code do you have that would break? Are you willing to
> retrofit your code?

Retrofitting old code is something that's been on my ToDo list for
quite a while now. No matter the solution you pick, someone's code
somewhere will break, and some of it will probably be mine.

> 2. Do you use existing libraries (like Astrolib or JHUAPL's) that
> would break? Could ITTVIS retrofit these 2 libraries and give them
> back to the community?

I'm certain that would be appreciated by the community.

>
> 3. Are there potential issues with changing to "defint32" (32-bit
> integers), or is the only problem with parentheses for arrays?

Any place where the integer must be confined to 16bit should be
explicitly specified in the code, e.g. reading binary files/call
external/etc.. I'm sure there are some routines of mine that would be
problematic here, and I confess that any bugs due to this change
should be considered as arising from a poor coding standard, rather
than a change to the language. And such a change would be welcomed as
it forces me to make the code more robust.

Regards,
Hugh

[*] Although why the pinheads decided to go against the common usage
of '.' as the structure/field delimiter I don't know. It's broken much
of my older DEC Fortran code and looks hideous: it makes it much
harder to visually separate the structure name from the field name.