Subject: Re: IDL 8.0 compile_opt changes
Posted by David Fanning on Fri, 18 Dec 2009 22:33:40 GMT
View Forum Message <> Reply to Message

Chris Torrence writes:

> I'm writing to you to ask your opinion on some potential changes in
> IDL 8.0. We have made some enhancements to the language to support the
> new graphics functions, and to make IDL simpler to learn, especially
> for new users.

Oh, dear! I would have put this out late Friday
afternoon, the week before Christmas, too. ;-)

I haven't thought through all of the ramifications
of this, but I'm glad you are. Personally, I would
be *extremely* grateful to have idl2 be the default
compiler option. It will break most of the IDL code
of my immediate supervisors (except, of course, for
those who are better IDL programmers than I am),
but that might actually be a Good Thing. Lord
knows there are a *lot* of reasons to get away
from 1970-style code. Is it possible to outlaw
the rainbow color table while you are at it?

I wouldn't worry too much about the established
libraries. I think they would adapt quickly. (Although
it might help to put a nice T-shirt in the box when
you send the maintainers their 8.0 upgrade.)

I'll give it some more thought while I'm shuffling
back and forth to the refrigerator for more eggnog
this next week.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: IDL 8.0 compile_opt changes
Posted by wlandsman on Sat, 19 Dec 2009 02:26:17 GMT

View Forum Message <> Reply to Message

On Dec 18, 4:51 pm, Chris Torrence <gorth...@gmail.com> wrote:

>
> 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.
>
I strongly support this option.

ITTVIS (then RSI) introduced square brackets with V5.0 in 1997. At
the time, I wrote a procedure

http://idlastro.gsfc.nasa.gov/ftp/contrib/landsman/v5/idlv4_ to_v5.pro

to automatically convert procedures using round parentheses for
indexing to use square brackets instead. idlv4_toV5 isn't perfect
-- it can get confused by parentheses within strings or matching
parentheses not on the same line -- but it took care of 95% of the
work, and I was able to convert the entire Astrolib.

From that time, whenever I've modified an astrolib procedure, I also
added a compile_opt idl2 at the beginning. The idea was that while I
was a testing a procedure for a new modification, I could also test if
making the default integer type long introduced any problems. And
there was one case where making the change did introduce some subtle
problems. This was for some homemade low-level database software
in which one parsed a byte stream. What happened was that "a = 0 &
readu,1,a" was now reading 4 bytes instead of 2. But this was the
only case of a problem, and I haven't had another problem in the past
eight years. I would think that 12 years of recommending compile_opt
idl2 is enough for people to adjust.

I should also add as an old (in more than one sense of the word)
programmer, I was never really comfortable with IDL objects until I
started learning some Python. Seeing the dot notation made me
realize that an object is just an IDL structure with functions
included ;-) So I agree that adding the dot notation to IDL should
certainly help new users. --Wayne

Subject: Re: IDL 8.0 compile_opt changes
Posted by penteado on Sat, 19 Dec 2009 02:39:08 GMT
View Forum Message <> Reply to Message

It is nice that you are asking us, instead of just dropping the
changes.

On Dec 18, 7:51 pm, Chris Torrence <gorth...@gmail.com> 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. For example, in IDL
> 8.0, the following code will create a plot, and retrieve the first
> child object:

I agree with the change, but the big problem I see with it is that
code
written in the new syntax would not work on older versions, which
would be a problem when we write code to be shared with other users.
In the academic
environment (the only I am familiar with), I still see a lot of people
using
IDL 6 (and many only using command line, editing source files with
vi).
Many times it is not their choice, because they are stuck with what is
installed (or the subset that works) in the department computer they
use.
Often I have found that such system administrators excel at making
users
suffer, from bad choices, and from not making things work properly. In
this
particular case, I am talking about administrators not updating IDL,
or not
bothering to make idlde work.

So since that change (and possibly others) would make the new syntax
backwards-incompatible, I would favour the use of a new extension to
identify
files with the new syntax, regardless of using or not option #5. And
if all
syntax changes are that simple, it would be nice to have a translator,
so that
we could write in the new syntax, but still have usable code for older
versions. Otherwise, I (and I guess others) would have to refrain from
using
the new syntax for a while.

The worst problem I see with the new extension is that it would allow
for the same routine to exist in a .pro and .prx file. The default
should be to pick the file with the new extension when there are both,
but it does create a new source of confusion. However, considering how
easily things can already get confusing due to files with the same
names existing in different places in the path, I do not find this
problem very significant.

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

I have a lot of old stuff written when I did not know any better.
However, any time I use that stuff for a new application, or to send
to
somebody else, it already takes a substantial overhaul anyway, to
replace all
the ugly stuff I used back then. So, the code I would have to retrofit
is code I already have to fix regardless of that change.

>
> 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?

Occasionally I use them. I would find it very nice if ITTVIS did the
necessary work on them.

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

I do not see a problem with that, since the default is being promoted.
If any code needs a particular data type, it should be picking the
type explicitly anyway, not relying on defaults.

Generally, I welcome the changes, to get rid of awkwardness or
limitations
inherited from ancient times, as David indicated (I also agree with a
rainbow
ban). Because of that, in principle, I would prefer for myself option
#1.

However, considering the effect on other users, I find option #5 to be
better.
If option #1 is used, I can already see a bunch of users not upgrading
to IDL
8 because it will break the code they use. Even worse, I see
department system
administrators not replacing old IDL versions because it would upset
some old
professor, whose code written in 1972 would stop working, and who
cannot be
bothered to think about why.

This is related to the general situation I saw in many universities,

where new
generations of students keep suffering, stuck with 1970s programming
because
that is the way they get "taught" by the old professors who keep doing
things
the same way they did 30 years ago. Using IDL is already a big step
from the
still frequent practice of doing everything in F77, writing text
files, and
then plotting them on some other, awkward software. Also, a lot of
people keep
writing F77-style IDL.

So while I agree with David that making the default break old code has
the positive effect of inducing change, I think that the side effect
of inhibiting upgrades may be even worse. The license files being
version-specific and the difficult installation in Linux are already
two big obstacles, that keep people using old versions. With option
#5, the new extension will make it more visible to old users that
there is something new and better, without being so aggressive as to
just make their code stop working.

Though option #3 is more flexible, it has the bad side effect of
making the execution environments less uniform, since the default
would not be known when writing the code, while option #5 gives an
unchanging rule. More users would be confused when their code worked
on one computer, and not in another (creating a new way for the sky to
fall).

I find options #4 and #2 to be undesirable. In case of #4, it would
keep the tolerance to the old style, and keep the current additional
complication to enforce the new style. I always saw the idl2 option as
a form to ease the transition, which should eventually become default.
And even without the new syntax, it is already confusing to deal with
the errors that arise when idl2 is not used, so it is good to have it
as default. As for option #2, it is a compromise between #1 and #4, it
has the problems of both options, and creates additional complexity.

Regardless of which option gets chosen, it is important that the
changes are well advertised and explained on the website, without
having to download the new version to read about it on the help. Even
though I look forward for all the new features at each new version,
sometimes I only notice one of them when I read this newsgroup or
somebody else's website (one recent example being the iTool changes
which Michael Galloy wrote about). Also, it is good to explain the
reasons for changes, as I often see people frustrated by what they
perceive as arbitrary and needless changes in software.

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.

Subject: Re: IDL 8.0 compile_opt changes
Posted by H. Evans on Sat, 19 Dec 2009 10:19:57 GMT
View Forum Message <> Reply to Message

Another thought, will the new syntax contain more information about
the code itself (whether it is an array

or a method, etc.) it should be easier to downgrade new code to the
older syntax, shouldn't it? To me, the best formed language doesn't
overload its syntax.

In the new syntax the '.' would be overloaded, so some clever parsing
would be required to determine if the '.' was a method call (changed
to '->') or a structure field (leave it as '.').

Hmmmmm...is this really an improvement? Would we not be muddying the
waters with ambiguity again? It took long enough to move from the
overloading of the '()' where it was for both array subscripting and
function parameter delimiting.

H.

Subject: Re: IDL 8.0 compile_opt changes
Posted by M. Katz on Sat, 19 Dec 2009 16:13:37 GMT
View Forum Message <> Reply to Message

I have 400,000 lines of code in my libraries. Maybe 100,000 is
"active." I would avoid upgrading to avoid having to convert every old
() to [] in code I wrote years ago but still rely on. I favor the .pro
and .prx split as an elegant solution for seamless recognition of
compile options, that preserves the function of old code with the
least amount of work. If there are .pro and .prx routines with the
same name, the compiler could choose the .prx by default, or have a
preference setting that allows an override.

Subject: Re: IDL 8.0 compile_opt changes
Posted by penteado on Sat, 19 Dec 2009 16:16:33 GMT
View Forum Message <> Reply to Message

On Dec 19, 8:19 am, "H. Evans" <bloggs...@googlemail.com> wrote:
> Another thought, will the new syntax contain more information about
> the code itself (whether it is an array
> or a method, etc.) it should be easier to downgrade new code to the
> older syntax, shouldn't it? To me, the best formed language doesn't
> overload its syntax.
>
> In the new syntax the '.' would be overloaded, so some clever parsing
> would be required to determine if the '.' was a method call (changed
> to '->') or a structure field (leave it as '.').
>
> Hmmmmm...is this really an improvement? Would we not be muddying the
> waters with ambiguity again? It took long enough to move from the

> overloading of the '()' where it was for both array subscripting and
> function parameter delimiting.
>
> H.

It would only be ambiguous when the idl2 option is not used
(explicitly or by default). With it, a.b is only a structure field of
there are [], or it is being used as a variable. A function would
require the (), and a routine could not appear in an expression.

Which is why idl2 should be the new default, either always (option
#1), or based on the file extension (option #5). That way, it would
also be possible to have a translator that would generate code
compatible with older versions, since it would find no ambiguities.

Subject: Re: IDL 8.0 compile_opt changes
Posted by H. Evans on Sat, 19 Dec 2009 17:45:02 GMT
View Forum Message <> Reply to Message

On Dec 19, 5:16 pm, pp <pp.pente...@gmail.com> wrote:
> On Dec 19, 8:19 am, "H. Evans" <bloggs...@googlemail.com> wrote:
>
>
>
>> Another thought, will the new syntax contain more information about
>> the code itself (whether it is an array
>> or a method, etc.) it should be easier to downgrade new code to the
>> older syntax, shouldn't it? To me, the best formed language doesn't
>> overload its syntax.
>
>> In the new syntax the '.' would be overloaded, so some clever parsing
>> would be required to determine if the '.' was a method call (changed
>> to '->') or a structure field (leave it as '.').
>
>> Hmmmmm...is this really an improvement? Would we not be muddying the
>> waters with ambiguity again? It took long enough to move from the
>> overloading of the '()' where it was for both array subscripting and
>> function parameter delimiting.
>
>> H.
>
> It would only be ambiguous when the idl2 option is not used
> (explicitly or by default). With it, a.b is only a structure field of
> there are [], or it is being used as a variable. A function would
> require the (), and a routine could not appear in an expression.
>
> Which is why idl2 should be the new default, either always (option

> #1), or based on the file extension (option #5). That way, it would
> also be possible to have a translator that would generate code
> compatible with older versions, since it would find no ambiguities.

My point was that it would be a trivial parse to downgrade, as with the conversion of '[' to '('. Requiring a check for the idl2 option in the code first requires that the 'downgrade' software be much more clever. The change from '->' to '.' syntax requires a fairly simple substitution script. My main point is that the '.' syntax would become overloaded and thus context sensitive. The conversion must, therefore, also include context sensitivity.

H.

Subject: Re: IDL 8.0 compile_opt changes
Posted by wlandsman on Sat, 19 Dec 2009 17:55:49 GMT

View Forum Message <> Reply to Message

On Dec 19, 11:13 am, "M. Katz" <MKatz...@yahoo.com> wrote:
> I have 400,000 lines of code in my libraries. Maybe 100,000 is
> "active." I would avoid upgrading to avoid having to convert every old
> () to [] in code I wrote years ago but still rely on.

What if 99,900 lines of code could be automatically converted using a single command?   As I mentioned, my very old code idlv4_to_5 (http:// idlastro.gsfc.nasa.gov/ftp/contrib/landsman/v5/idlv4_to_v5.p ro) wasn't perfect, but I think most of its omissions are fixable without too much trouble.   I suspect the only case where an automatic conversion couldn't work is for lines within an EXECUTE() statement.

My problem with a new extension name is that I would like to write code that would work on both post- and pre-8.0 versions.   For the next few years I would envision writing code snippets like this

```
   if !VERSION.RELEASE GE '8.0' then begin
;............Use the dot for object method calls
   endif else begin
;............Use the traditional object method syntax
    endelse
```

so that the code could still run in versions prior to V8.0.   But these earlier IDL versions won't have any recognition of a .prx extension.

Hmmm... I could see writing code like this getting very unwieldy.
Maybe instead we would need a new program IDLV7_to_V8 to automatically convert code to the new method syntax...

--Wayne

Subject: Re: IDL 8.0 compile_opt changes
Posted by chris_torrence@NOSPAM on Sat, 19 Dec 2009 19:11:27 GMT
View Forum Message <> Reply to Message

On Dec 19, 10:55 am, wlandsman <wlands...@gmail.com> wrote:
>
> so that the code could still run in versions prior to V8.0.   But
> these earlier IDL versions won't have any recognition of a .prx
> extension.
>
> Hmmm... I could see writing code like this getting very unwieldy.
> Maybe instead we would need a new program IDLV7_to_V8 to automatically
> convert code to the new method syntax...
>
> --Wayne

I'm not sure it would be worth writing your code to include both the
new "dot" way, and the old method way. You would just be duplicating
the method calls for no good reason. If you knew you were going to
share the code with older versions, you would probably be better off
just writing the code the old way.

Another point of information: We are adding a lot of new language
features to IDL 8.0. Things like a new "foreach" operator, negative
array subscripting a[0:-1], plus a lot more. You would need to avoid
using *any* new language features if you wanted to run on an older
version. This was always true, like when we added && || ++, etc.

To be honest, I don't really like the idea of a new extension. It
seems like it just masks the problem - essentially it is like
"compile_opt", but even harder to explain. People who found a .prx on
the web would have to remember that they need to run the code in IDL
8.0+. Plus it splits the community's code base into two pieces: the
"old" and the "new". If you found a .pro on the web, is it named .pro
because it is old and crufty, or simply because it doesn't contain any
new syntax? At least in my mind, IDL == .PRO, and .PRO == IDL.

More thoughts? More eggnog?

Subject: Re: IDL 8.0 compile_opt changes
Posted by penteado on Sat, 19 Dec 2009 20:32:06 GMT
View Forum Message <> Reply to Message

On Dec 19, 5:11 pm, Chris Torrence <gorth...@gmail.com> wrote:
> Another point of information: We are adding a lot of new language
> features to IDL 8.0. Things like a new "foreach" operator, negative
> array subscripting a[0:-1], plus a lot more. You would need to avoid
> using *any* new language features if you wanted to run on an older
> version. This was always true, like when we added && || ++, etc.

I was wondering about that. Which is why I said that code could be translated, *if* all the syntax changes were as simple as the "." operator. If there are a lot of improvements (will lists, maps, and other containers finally arrive?), then code backwards compatible will be so limited (in the new perspective) that people will have to adapt around the change. In that case, it is good if the change is big, because it makes it easier to convince other people to upgrade.

But that is a separate issue, unrelated to making idl2 the default option. The big problem I see with just making it default is that some old code will stop working, and a lot of people will resist upgrading because of that. They already resist upgrading to IDL 7, which does not break much (I noticed a few changes in the iTools objects that did). The main difficulties to upgrades now are that it is hard to install and make it all work (in Linux anyway; I have not noticed difficulties in Windows, and I do not know the situation in Macs), and the license file may be limited to older versions. Remember that a lot of people use IDL on their own computer, which they can upgrade, but with the license through a department server, which they can not do anything about. It is very necessary to make upgrades easier, and breaking compatibility would only make them harder. And while a lot of people do not upgrade, it is hard for those who did to write code using the new features, and to teach them to new users, thus slowing progression to the new ways, which is already dreadfully slow.

Which is why I see the new extension as a better solution. Yes, just putting a compile_opt idl1 would immediately make the old routines work, but then the same file would not work on an older IDL (because of the unknown idl1), and a lot of people use the same programs on multiple computers, some of which they might not be able to update. So if they cannot retrofit old code (either because they cannot immediately stop their work to do it, or because they did not write the code and do not understand it), they will need to keep two sets of files, one with compile_opt idl1, to run in IDL 8, and another without it, to run on older versions.

With a new extension, old stuff keeps working, new users can learn in the new ways, and retrofitting can be gradual, as time permits.

I do not see problem in associating IDL with .prx (or whatever, just make sure it is not in use by some other common program). I think it

makes it easier to explain and interpret. With it, you immediately
know the program takes IDL 8 to run; without it, if the writer did not
inform you, there would be only a bunch of syntax errors to hint at an
incompatibility.

Even though a new extension is like compile_opt, it is better, because
it is more visible: everybody would see there is something new, even
without opening the file, while a compile_opt may easily go unnoticed
inside it. I myself only learned about [] and compile_opts after years
of using IDL without knowing they were there. Yes, it is the result of
faulty education I got, but it is the typical kind of faulty computer
programming education that scientist keep getting. I would probably
have wondered about why there is another extension much sooner.

Also, today when we encounter a .pro file we already wonder if it is
old and crufty anyway, so we have to look inside it.

Subject: Re: IDL 8.0 compile_opt changes
Posted by Robbie on Sun, 20 Dec 2009 11:15:19 GMT
View Forum Message <> Reply to Message

Another suggestion, although this feature might already exist in IDL
7.1 or 7.2.

Would it be possible to get the workbench to automatically put
compile_opt idl2 at the top of every new .pro file? This would ensure
that new users experience the new language features. I personally
don't use compile_opt as often as I should.

Thanks

Robbie

Subject: Re: IDL 8.0 compile_opt changes
Posted by wallabadah on Mon, 21 Dec 2009 01:10:56 GMT
View Forum Message <> Reply to Message

On Dec 20, 10:15 pm, Robbie <ret...@iinet.net.au> wrote:
> Another suggestion, although this feature might already exist in IDL
> 7.1 or 7.2.
>
> Would it be possible to get the workbench to automatically put
> compile_opt idl2 at the top of every new .pro file? This would ensure
> that new users experience the new language features. I personally
> don't use compile_opt as often as I should.

>
> Thanks
>
> Robbie

This is easy to implement - I wrote a routine to generate a function/
procedure skeleton with a few default options already written
(including compile_opt idl2), and a comment file header that
encourages me to comment my code properly (it works!). I use it on Mac
OS X at the IDL prompt in the Terminal like:
IDL> idl_codefile, "my_great_function", /is_function
and the skeleton file is constructed, named and opened for editing in
BBEdit/TextWrangler. There's also a version for creating new object
code files, where you supply an array of function names and an array
of procedure names, and the skeleton is generated so you can fill in
the appropriate bits of code. Can save a hundred or so lines of coding
- and I'm sure something similar could be integrated with the
workbench.

I think moving to a new file extension is a mistake, except if it were
used to distinguish between scripts/@includes and files of functions/
procedures to be compiled and run.
I think compile_opt idl2 should be the default, and also that there
should be a better mechanism for checking IDL version at runtime and
compile time, and people *should use it* in new code. How about if the
preprocessor/compiler issued warnings when code requiring a certain
IDL version is compiled... similar to the !warn structure variable.
For example, turning on the minimum_idl_version_required flag would
cause IDL to print that "IDL Version 6.X is required to run this
code". One could use it to check code before sending to an elderly
colleague still running IDL 5.x (or to someone running IDL 7.X, for
that matter). This would be very useful. It could even add it to the
source code itself (eg. top line, as a comment, including the line
number and context).
If we could then write conditional code that correctly allowed for
different IDL version (ie. don't parse the next bit of code if the IDL
version is such that IDL won't understand it anyway - read
http://www.dfanning.com/tips/backwards.html for a refresher), life
would be a fair bit simpler. No need for separate files for each IDL
version, just code that takes advantage of new features if the IDL
version allows, and uses older methods/workarounds/defaults to
something else if the IDL version is too low. This doesn't help with
legacy code, but I think it's a good feature to implement now for the
next time this sort of thing happens, and a standard practice for code
going into "publicly-accessible" libraries.

Writing a couple of routines to check existing code libraries and
update them should be a relatively simple exercise (for the IDL

developers!), as they already have the code required to parse IDL
routines written into IDL itself. As a previous poster has said, this
sort of thing should take care of the majority of existing code, the
remainder would probably require some human interaction. Some of this
old code probably needs a serious review anyway.

If people expect IDL to be updated and become a 'modern' language,
there need to be some serious changes, which are likely to affect
existing code. If you'd like to stay on IDL 6.X for the next decade,
go for it, but you can't have your cake and eat it too.

Will

Subject: Re: IDL 8.0 compile_opt changes
Posted by ddegraff on Mon, 21 Dec 2009 05:33:52 GMT

View Forum Message <> Reply to Message

On Dec 19, 2:11 pm, Chris Torrence <gorth...@gmail.com> wrote:

>
> Another point of information: We are adding a lot of new language
> features to IDL 8.0. Things like a new "foreach" operator, negative
> array subscripting a[0:-1],
>
> More thoughts? More eggnog?

No! negative index is a Bad Idea!

```
z= where(bad ne 0)
 if z[0] ne -1 then begin
    lots of stuff
 endif
```

I have a lot of code like this, which would no longer be valid if -1
were a legal array index. I could go thought and include a count in
the where function but that's a lot more work than fixing all the ().
Actually I was surprised that you could still use () for arrays. I
tell my students to always use [].

david

Subject: Re: IDL 8.0 compile_opt changes
Posted by penteado on Mon, 21 Dec 2009 06:16:48 GMT

View Forum Message <> Reply to Message

On Dec 21, 3:33 am, ddegr...@stny.rr.com wrote:
> No! negative index is a Bad Idea!
>
> z= where(bad ne 0)
> if z[0] ne -1 then begin
>   lots of stuff
> endif
>
> I have a lot of code like this, which would no longer be valid if -1
> were a legal array index. I could go thought and include a count in
> the where function but that's a lot more work than fixing all the ().
> Actually I was surprised that you could still use () for arrays. I
> tell my students to always use [].
>
> david

I use that a lot, too. I do not know what the negative indexes will
mean in IDL (the same as in Python?), but that does not by itself mean
that where() would no longer return -1 for nothing found.

Also, I think that replacing those occurrences by checking where's
count is easier than replacing () by [], since they can be found by
text searches. In the case of (), simple text searches would find a
lot of uses of () for precedence in expressions, and in function
calls. And because of the function call ambiguity, it can be tricky
when inspecting the code to determine if something is a function or a
variable. In the case of -1 returned by where, there is no ambiguity,
and if where's results are always test close to it (as is probably the
case most often), it is not so difficult to find and replace all
occurrences.

Still, if there would be no bad side effects, it would be better if
where() will only return -1 meaning something other than not found if
it some new keyword is set. That way, old constructs like the one you
mentioned would be unaffected.

Somehow, this reminds me what it would also be really nice to have a
syntax for accessing substrings with indexes, as in Python, Fortran or
C/C++. Having to use strmid() all the time is a bit tiring, specially
for writing to a piece of a string, when the pieces have to be put
together with +, strjoin or strput.

Subject: Re: IDL 8.0 compile_opt changes
Posted by Wout De Nolf on Mon, 21 Dec 2009 11:00:29 GMT
View Forum Message <> Reply to Message

On Fri, 18 Dec 2009 18:26:17 -0800 (PST), wlandsman

<wlandsman@gmail.com> wrote:

> ITTVIS (then RSI) introduced square brackets with V5.0 in 1997.   At
> the time, I wrote a procedure
>
> http://idlastro.gsfc.nasa.gov/ftp/contrib/landsman/v5/idlv4_ to_v5.pro

I was wondering whether ITTVIS couldn't include something in the
workbench (as it is parsing code anyway) that would locate:
- () uses instead of []   (STRICTARR issues)
- a = 0 instead of a = 0s (DEFINT32 issues)
- ...all other things involving COMPILE_OPT and code breaking issues

---

Subject: Re: IDL 8.0 compile_opt changes
Posted by H. Evans on Mon, 21 Dec 2009 13:15:37 GMT
View Forum Message <> Reply to Message

On Dec 21, 6:33 am, ddegr...@stny.rr.com wrote:
> On Dec 19, 2:11 pm, Chris Torrence <gorth...@gmail.com> wrote:
>
>
>
>> Another point of information: We are adding a lot of new language
>> features to IDL 8.0. Things like a new "foreach" operator, negative
>> array subscripting a[0:-1],
>
>> More thoughts? More eggnog?
>
> No! negative index is a Bad Idea!
>
> z= where(bad ne 0)
>   if z[0] ne -1 then begin
>   lots of stuff
>   endif
>
> I have a lot of code like this, which would no longer be valid if -1
> were a legal array index. I could go thought and include a count in
> the where function but that's a lot more work than fixing all the ().
> Actually I was surprised that you could still use () for arrays. I
> tell my students to always use [].
>
> david

I also have large amounts of code that use exactly this type of range
limit checking. The completely correct solution is to get the number
of elements found in the where command and check on that. This,

though, doesn't solve the basic problem of having to envelop all
subsequent code in an IF/THEN/ELSE construct, which I find becomes
unwieldy, almost as unwieldy as x[N_ELEMENTS(x)-1], which looks much
nicer as x[-1].

I've often thought the WHERE function should be provided so it can be
used as follows:

```
IF ( WHERE( z eq bad_data, i) ) THEN z[i] = !values.f_nan
```

I find it more succinct, less clumsy and more obvious than:

```
I = WHERE( z EQ bad_data, count)
IF ( count GT 0) z[i] = !values.f_nan
```

Obviously it's simple enough to implement a basic rewrite:

```
FUNCTION IS_FOUND, condition, indices, COUNT=COUNT, _EXTRA=_EXTRA
  indices = WHERE( condition, count, _EXTRA=_EXTRA)
  RETURN, count GT 0
END
```

But the underlying problem is how to maintain backwards compatibility
and provide a new syntax for indexing from the end of the array. I'm
not sure that the new file extension solution would work here, as it's
possible that the newer code would output correct -ve indices that
would be used as input to older code that would cease to function.

In the example you give above, it should be a matter of just ensuring
that the WHERE function continues to work as it currently does, but
the user must be aware that the "-1" return value is not to be used in
indexing. This is likely to break older code that doesn't check that
the WHERE found any valid indices, i.e.:

```
i = where( x eq bad_data)
x[i] = !values.f_nan
```

But then, this sort of code is quite likely to be a source of problems
in the first place and would be improved by the new coding
restrictions.

Alternatively, could a new data type be created for indexing, e.g.
ARRAY_INDEX? An index is more than just an ordinal. Typecasting could
be used transparently to convert much of the existing code to the
newer index type. The type could have an equivalent undefined/NaN
value for no indices found, and while -ve numbers could refer to
reverse indexing a more robust solution could be implemented. This
could also then be shoehorned into the old way of working with a
suitable pragma option, set either through the file name extension or

the compile_opt statements. For .pro files, the undefined value would
be returned as -1L, and for the .prx files it could be genuinely
undefined. This should stop old code from breaking and still allow for
the extension of the reverse indexing.

I believe the opportunity for addressing many of the existing
shortcomings in the language should be grasped if the option for a new
file extension is chosen.

H.

Subject: Re: IDL 8.0 compile_opt changes
Posted by Foldy Lajos on Mon, 21 Dec 2009 13:52:02 GMT

View Forum Message <> Reply to Message

On Mon, 21 Dec 2009, H. Evans wrote:

```
> ...   almost as unwieldy as x[N_ELEMENTS(x)-1], which looks much
> nicer as x[-1].
```

I have ### for 'last element' operator in FL:

```
FL> a=indgen(5) & print, a[###] ;  a[4]
        4
FL> b=indgen(3,4,5) & print, b[###/2, ###/2, ###/2] ; b[1,1,2]
       28
```

It would be useful to have something similar in IDL.

regards,
lajos

Subject: Re: IDL 8.0 compile_opt changes
Posted by H. Evans on Mon, 21 Dec 2009 14:43:34 GMT

View Forum Message <> Reply to Message

On Dec 21, 2:52 pm, FÖLDY Lajos <fo...@rmki.kfki.hu> wrote:

```
> On Mon, 21 Dec 2009, H. Evans wrote:
>> ...   almost as unwieldy as x[N_ELEMENTS(x)-1], which looks much
>> nicer as x[-1].
>
> I have ### for 'last element' operator in FL:
>
> FL> a=indgen(5) & print, a[###] ;  a[4]
>         4
>  FL> b=indgen(3,4,5) & print, b[###/2, ###/2, ###/2] ; b[1,1,2]
>        28
```

I could live with that notation type...or some similar operator ;-)

Subject: Re: IDL 8.0 compile_opt changes
Posted by chris_torrence@NOSPAM on Mon, 21 Dec 2009 17:23:12 GMT
View Forum Message <> Reply to Message

Just a quick note about the negative indices. They are *only* allowed
in ranges or as a scalar index. They are *not* allowed in arrays of
indices.

For example:

a[0:-1] In IDL80, returns all the elements

a[-1] In IDL80, returns the last element

a[[-1,0,1,2]] will either truncate the -1 to 0, or will throw an
error if the "strictarrsubs" compile option is set. This behavior will
remain the same in IDL80. We will not change it.

So, for example, David's example:

```
z= where(bad ne 0)
if z[0] ne -1 then begin
   lots of stuff
endif
```

This will still work *fine* in IDL80.

Now, in IDL80 the following might cause a problem:

```
  z= where(bad ne 0)
  do something with bad[z] (without checking for -1)
```

In this case, if "where" returns a scalar -1, then bad[z] will indeed
return the last element. However, it would be bad to use the where
result without checking for a -1...

So, in summary, in IDL 8.0:
1. Negative indices are only used for ranges or a scalar index
2. Negative indices within an index array will still behave the same
(either clip or throw an error)
3. The behavior of Where remains unchanged (and you still need to
check for a -1 result, as always)

Returning to the compile_opt discussion...

-Chris
ITTVIS

Subject: Re: IDL 8.0 compile_opt changes
Posted by markb77 on Thu, 24 Dec 2009 03:40:52 GMT
View Forum Message <> Reply to Message

Seems like a good idea to me just to change the default compilation
behavior to idl2, without adding another extension etc. I think the
libraries will be updated quickly enough by the user community.
Adding a new extension would just create unnecessary complexity which
is not what the IDL language needs. I have a fairly large code base
which is written using the [] standard, so I'm not worried about it
breaking too badly.

Mark

Subject: Re: IDL 8.0 compile_opt changes
Posted by Mark Piper on Tue, 29 Dec 2009 15:26:59 GMT
View Forum Message <> Reply to Message

On Dec 20, 4:15 am, Robbie <ret...@iinet.net.au> wrote:
> Another suggestion, although this feature might already exist in IDL
> 7.1 or 7.2.
>
> Would it be possible to get the workbench to automatically put
> compile_opt idl2 at the top of every new .pro file? This would ensure
> that new users experience the new language features. I personally
> don't use compile_opt as often as I should.
>
> Thanks
>
> Robbie

Hi Robbie,

You can add a COMPILE_OPT call to the existing code templates. There's
a short (2 min) video of this on the VIS website:

http://www.ittvis.com/portals/0/webinar/workbench-templates- 2/code-templates-modifying.html

mp
