Subject: IDL 8.0 compile_opt changes
Posted by chris_torrence@NOSPAM on Fri, 18 Dec 2009 21:51:39 GMT
View Forum Message <> Reply to Message

Hi all,

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.

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:

```
p = PLOT(x,y)
child = p.Get(index)
```

Disregarding the actual syntax, the problem occurs when IDL tries to compile the second line of code. By default it will think that "Get" is just an array field inside of a structure, and that the parentheses are just indexing into the array.

Now, you could use "compile_opt idl2", or "compile_opt strictarr" to remove the ambiguity. In this case, for IDL 8.0, it will then know that this is a function method call.

That's fine for existing users, like us, who are happy to sprinkle "compile_opt idl2" all through our code. However, for new users this is strange, and it would be hard to explain why they are getting a syntax error.

For IDL 8.0, we want the new language features like the "dot" to work "out of the box". So, ideally, we would change the default compile_opt for IDL to be "idl2" - this includes both "defint32" and "strictarr". All integer scalar constants would be 32-bit by default, and parentheses would not be usable for array indexing. However, this creates a backwards compatibility problem. IDL .save files would be fine (the code is already compiled), but large libraries (like JHUAPL) would not be usable without changes to the code.

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.

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.

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.

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.

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.

We would really like to hear your opinion on these potential solutions. Questions to think about:

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

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?

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

Thanks for reading all of this, and we look forward to your replies.
Happy Holidays!

-Chris

Dr. Christopher Torrence
IDL Software Development Manager
ITTVIS

Subject: Re: IDL 8.0 compile_opt changes
Posted by proffmw@gmail.com on Thu, 24 Dec 2009 04:08:21 GMT
View Forum Message <> Reply to Message

Hi all,

As an old fogey who still has working code from 1982 (version 2, when
we used VMS and our disk quota was 5 kiloblocks), I much prefer
the .pro/.prx solution. If I have to dig into that old code to change
parentheses to brackets, specify that 16 bit associated variables are
16 bits, remove all VMS-specific keywords, and everything else that is
now obsolete, I'll probably feel compelled to rewrite 28 years worth
of code to make it look and run better, and my productivity will drop
to or below zero... You think I can improve code without breaking it?

Write a V8 complier that will look for the .prx file and, not finding
that, will look for the .pro file. In that way a user can have 2
versions if needed, or a single compatible version.

(I currently use versions 6.2, 6.3, 6.4, and 7 on 4 different
machines. I upgrade only
when machines die or are retired. Often upgrades to a higher version
than I really need/want are dictated by forces beyond my control.)

Cheers,
Fred

Subject: Re: IDL 8.0 compile_opt changes
Posted by David Fanning on Thu, 24 Dec 2009 04:48:03 GMT
View Forum Message <> Reply to Message

proffmw@gmail.com writes:

> As an old fogey who still has working code from 1982 (version 2, when
> we used VMS and our disk quota was 5 kiloblocks), I much prefer
> the .pro/.prx solution. If I have to dig into that old code to change
> parentheses to brackets, specify that 16 bit associated variables are
> 16 bits, remove all VMS-specific keywords, and everything else that is
> now obsolete, I'll probably feel compelled to rewrite 28 years worth
> of code to make it look and run better, and my productivity will drop
> to or below zero... You think I can improve code without breaking it?

I think most people could probably put a "compile_opt idl1" into their
code without breaking it.

> (I currently use versions 6.2, 6.3, 6.4, and 7 on 4 different
> machines. I upgrade only

> when machines die or are retired. Often upgrades to a higher version
> than I really need/want are dictated by forces beyond my control.)

Then, nothing will change for you. :-)

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 penteado on Thu, 24 Dec 2009 06:13:25 GMT
View Forum Message <> Reply to Message

On Dec 24, 2:48 am, David Fanning <n...@dfanning.com> wrote:
> prof...@gmail.com writes:
>> As an old fogey who still has working code from 1982 (version 2, when
>> we used VMS and our disk quota was 5 kiloblocks), I much prefer
>> the .pro/.prx solution. If I have to dig into that old code to change
>> parentheses to brackets, specify that 16 bit associated variables are
>> 16 bits, remove all VMS-specific keywords, and everything else that is
>> now obsolete, I'll probably feel compelled to rewrite 28 years worth
>> of code to make it look and run better, and my productivity will drop
>> to or below zero... You think I can improve code without breaking it?
>
> I think most people could probably put a "compile_opt idl1" into their
> code without breaking it.

If he does that, he will break the code for use with older versions,
since the option idl1 does not exist in them. As probably the upgrade
will not happen at the same time on all 4 machines he uses, he would
need to keep two sets of files, one to use with IDL 8 with the idl1
option, and the other, unchanged, to use with IDL <8. In my
experience, it is common for people to use the same code on many
different computers (often even all from the same filesystem), and
they do not have control over when upgrades are installed in some, or
all of those computers.

This is one example of why I was saying that a new extension should be
used. It is the best way to keep old code working while making the
needed changes. Even those who want to retrofit their code will not

want to keep two copies of each file (with and without the idl1 option), and may not have the time to immediately stop everything to find and replace all the () in the old files.

Even in Fortran, when people wanted to get rid of the old horrible fixed source format the choice was made to use a new extension. So the new compilers would know to treat the .f files with the old horrible standard, and the .f90 files with the new one. Fortran is a language with a strong lack of good choices in its design, but in that particular decision was a good one. Sure, it made a lot of people not even noticing that there was something new, even to this day (if you are not familiar with Fortran, a lot of people still use the 1977 standard as if it is the only one). But a lot of people would have kept using old compilers if the new ones did not work with unaltered old code.

As I said before, IDL currently has issues that already make its upgrade too difficult. It should not have a new obstacle. Keep in mind that the most frequent posters in this newsgroup are likely to be better informed of IDL's changes and better ways to write code, and more willing to make changes, than most IDL users. If idl2 becomes default, that majority of users may simply not understand why their code stopped working, and so keep using the older versions. And most of the well informed users know how inconvenient it is to keep limiting what they use in their new code because some if its users are still with an old version of IDL. And even if those people do not even know it, we can send them our files in the new standard and their relatively new compilers will understand it.

It reminds me of the (true) case of the man who, after living well for most of life while blind, recovered his vision in his fifties through surgery. His family then thought that he should immediately live as a normal person, without ever using his cane or any of the other aids he used when blind. Since he did not know how to make sense of the images, he could no longer function, got stressed, depressed, sick and died shortly afterwards. This story is told in Oliver Sacks' book "An Anthropologist on Mars", and was somewhat altered into the story of the movie "At First Sight", with Val Kilmer and Mira Sorvino. My point is that even if change is for the better, forced instant change with no adaptation can have severe bad side effects.

In short, breaking compatibility with old code would be much worse than a new extension, both for those who agree that the new way if doing things is necessary, and for those who do not even now there is a new way. I say this from experience of how people in the academic sector write their code, and of trying to get them to stop suffering from the old ways.

Subject: Re: IDL 8.0 compile_opt changes
Posted by David Fanning on Thu, 24 Dec 2009 11:44:21 GMT
View Forum Message <> Reply to Message

pp writes:

> As I said before, IDL currently has issues that already make its
> upgrade too difficult. It should not have a new obstacle. Keep in mind
> that the most frequent posters in this newsgroup are likely to be
> better informed of IDL's changes and better ways to write code, and
> more willing to make changes, than most IDL users. If idl2 becomes
> default, that majority of users may simply not understand why their
> code stopped working, and so keep using the older versions. And most
> of the well informed users know how inconvenient it is to keep
> limiting what they use in their new code because some if its users are
> still with an old version of IDL. And even if those people do not even
> know it, we can send them our files in the new standard and their
> relatively new compilers will understand it.

You are probably right. But even though it is Christmas
week, and I have suspended my usual cynicism for a few
days, recent events (say since at least IDL 6.1) have not
convinced me that ITTVIS does much of anything out of
concern for their plodding-behind, direct graphics brethren.
If they are willing to make a big break like this, I suspect
it has more to do with making sure those people *can't* keep
up than concern for their future. Changing the file extension
will simply explicitly spell out the doom that awaits those of us
who wants to stick with a *.pro extension.

At least with a compile option there is the illusion of hope.

Peace be unto the World!

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 Thu, 24 Dec 2009 13:07:35 GMT
View Forum Message <> Reply to Message

On Dec 24, 1:13 am, pp <pp.pente...@gmail.com> wrote:

>> I think most people could probably put a "compile_opt idl1" into their
>> code without breaking it.
>
> If he does that, he will break the code for use with older versions,
> since the option idl1 does not exist in them.

Maybe it would be better then to have a flag (or separate icon) when starting up IDL, e.g.

idl -classic

that would recognize the parentheses indexing (and thus *not* recognize the new method syntax). This would ensure that users could run their old code, but also encourage them to take the 20 minutes to automatically convert their code to use square bracket indices. (Note that IDL currently knows when () is used as an index and when it is used for a function in a pre-8.0 procedure -- since it knows the names of all variables -- so the conversion can certainly be automated.) --Wayne

---

Subject: Re: IDL 8.0 compile_opt changes
Posted by penteado on Thu, 24 Dec 2009 16:49:26 GMT
View Forum Message <> Reply to Message

On Dec 24, 11:07 am, wlandsman <wlands...@gmail.com> wrote:
> Maybe it would be better then to have a flag (or separate icon) when
> starting up IDL, e.g.
>
> idl -classic
>
> that would recognize the parentheses indexing (and thus *not*
> recognize the new method syntax). This would ensure that users
> could run their old code, but also encourage them to take the 20
> minutes to automatically convert their code to use square bracket
> indices. (Note that IDL currently knows when () is used as an index
> and when it is used for a function in a pre-8.0 procedure -- since it
> knows the names of all variables -- so the conversion can certainly be
> automated.) --Wayne

I would not say that the conversion can be automated. The current criteria to decide whether something is a function or variable depend not only on the defined variables, but also on the known functions, at

the time the reference is encountered. The trouble happens if there are both a function and a variable with the same name. If the reference is found before the function is compiled, it is taken to be a variable. If the function happens to have been compiled, the reference is taken to be a function. So the conversion of a file does not depend only on its contents, but also on the environment when the file is read. And as many other topics here indicate, routine name resolution is fraught with difficulties.

What could be automated is to add the compile opt to, say, all routines found in a directory. The trouble is that those files would no longer work on older versions because of the idl1, so it is not a good solution either. Having a switch or a preference making the interpreter work the old way is better than making idl2 the default all the time. But it has the problem that one would not be able to run both old and new code during the same session, so it would still pose an obstacle to change, though a smaller one. Which is why I still see a new extension as a better choice.

Subject: Re: IDL 8.0 compile_opt changes
Posted by alt on Fri, 25 Dec 2009 10:30:12 GMT
View Forum Message <> Reply to Message

First of all thank you for asking.
I personally approve the default behaviour for “idl2” and do not agree with “dot”. IMHO “dot” will make the code less readable.
I want to propose another possible solution for compiler configuration. Put some config (xml?) file in the library directory. This file could describe the desired compiler options for all pro-s in directory (and subdirs). So the problem remains only for mixed directories which it seems a much rarely case. Subconfigs could override upper dir config and so on (e.g. as for Apache .htaccess files). These configs could manage all possible properties that concerns underlying pro-s. Their role can be extended even to manage procedures and common block namespaces.
Best Regards,
Dmitriy Altyntsev

Subject: Re: IDL 8.0 compile_opt changes
Posted by H. Evans on Sat, 26 Dec 2009 21:21:27 GMT
View Forum Message <> Reply to Message

So, what's the problem with using a "new" syntax that involves the comment character? This is what happens with numerous Unix shell scripts. For example:

;!compile_opt idl1

which would not break old code, but would inform the new version that the old syntax should be used when interpreting the .pro file. This could/should be the first line of the file. This would make converting older files a doddle with the flexibility of setting the default compile_opt to idl2 while at the same time not breaking the file for older versions of IDL. Ideally a flexible syntax should be specified (could even be XML format if we really wanted to be absurd) that can be encapsulated in the commented header of a file.

I'm still partial to a new file extension, though. As mentioned before, it worked quite well with the FORTRAN community (and let's face it, there hasn't been a compiled computer language to rival Fortran developed in the last 20 years.). And let's not forget that minor, but much needed upgrade to c: c++.

H.

Subject: Re: IDL 8.0 compile_opt changes
Posted by penteado on Mon, 28 Dec 2009 03:44:51 GMT
View Forum Message <> Reply to Message

On Dec 26, 7:21 pm, "H. Evans" <bloggs...@googlemail.com> wrote:
> So, what's the problem with using a "new" syntax that involves the
> comment character? This is what happens with numerous Unix shell
> scripts. For example:
>
> ;!compile_opt idl1
>
> which would not break old code, but would inform the new version that
> the old syntax should be used when interpreting the .pro file. This
> could/should be the first line of the file. This would make converting
> older files a doddle with the flexibility of setting the default
> compile_opt to idl2 while at the same time not breaking the file for
> older versions of IDL. Ideally a flexible syntax should be specified
> (could even be XML format if we really wanted to be absurd) that can
> be encapsulated in the commented header of a file.
>
> I'm still partial to a new file extension, though. As mentioned
> before, it worked quite well with the FORTRAN community (and let's
> face it, there hasn't been a compiled computer language to rival
> Fortran developed in the last 20 years.). And let's not forget that
> minor, but much needed upgrade to c: c++.

I used Fortran as an example because C++ is so far from C that it

seems more like a new language rather than an upgrade. But same as Fortran, they also generally use the same compilers, which decide what to do based on the file extension.

An example of the trouble of a backward-incompatible update keeping the same extension is the current situation with Python. Python 3 has been out for about one year now, and still some major Linux distributions ship with Python 2, because of all the old software that would not work unchanged. Which slows down the transition, forcing the use of the older version in new software being developed now.

Subject: Re: IDL 8.0 compile_opt changes
Posted by wlandsman on Mon, 28 Dec 2009 12:36:57 GMT
View Forum Message <> Reply to Message

On Dec 24, 11:49 am, pp <pp.pente...@gmail.com> wrote:
> The trouble happens if there
> are both a function and a variable with the same name. If the
> reference is found before the function is compiled, it is taken to be
> a variable. If the function happens to have been compiled, the
> reference is taken to be a function. So the conversion of a file does
> not depend only on its contents, but also on the environment when the
> file is read. And as many other topics here indicate, routine name
> resolution is fraught with difficulties.

That ambiguity provides a reminder of why users should use square brackets for indexing, irrespective of any IDL 8.0 changes. The test program

```
pro test
rot = indgen(3)
print,rot(1)
return
end
```

will treat rot(1) as the ITTVIS rot.pro function, rather than a variable, but only if it has been previously compiled. The consistent use of square brackets for indexing removes all ambiguity.

While a program to automatically convert to square brackets won't know whether a user really wants rot(1) to be a function call or an indexed variable, it can certainly flag all such ambiguous cases, so that the user can manually edit them if they don't want the function call. Such cases should be relatively rare, and one could argue that the program was broken to begin with.

Incidentally, Bill Thompson had written a converter to square brackets that was better than mine. (It handled most EXECUTE() strings and main level programs.) It is available at http://tinyurl.com/yasbr7m

--Wayne

Subject: Re: IDL 8.0 compile_opt changes
Posted by Jeremy Bailin on Wed, 30 Dec 2009 05:31:57 GMT
View Forum Message <> Reply to Message

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

Am I the only person who sees this as the bigger issue? I've never used () for array subscription (came into IDL just after []) but I'm guilty of reading in binary files assuming 16-bit integers.

My two cents: I hate the idea of a new file extension thanks to some bad experiences with Fortran. I make extensive use of existing libraries. I probably lean towards making idl2 the default, but much as I've always thought it was an ugly method, I think the idea of hiding an idl1 preference behind a comment character as H.E. suggests is the best solution.

-Jeremy.

Subject: Re: IDL 8.0 compile_opt changes
Posted by R.Bauer on Mon, 04 Jan 2010 10:30:39 GMT
View Forum Message <> Reply to Message

Chris Torrence schrieb:
> Hi all,
>
> 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.
>

Hi Chris

Happy new year to you and ittvis. Thanks for asking us.

> 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:
>
> p = PLOT(x,y)
> child = p.Get(index)

I support this change. I felt always the attempt of doing OO in idl is extremly uncomfortable and ugly compared to python.
I think this change is important for new users or old users using OO in different languages.

>
> Disregarding the actual syntax, the problem occurs when IDL tries to
> compile the second line of code. By default it will think that "Get"
> is just an array field inside of a structure, and that the parentheses
> are just indexing into the array.
>
> Now, you could use "compile_opt idl2", or "compile_opt strictarr" to
> remove the ambiguity. In this case, for IDL 8.0, it will then know
> that this is a function method call.
>
> That's fine for existing users, like us, who are happy to sprinkle
> "compile_opt idl2" all through our code. However, for new users this
> is strange, and it would be hard to explain why they are getting a
> syntax error.

In my eyes the change should have been done one release after the parenthesis change. Now it is quite late. Some of my coleagues still use the old style '()' and they have produced lots of stuff. I often become told "Why should I change this - it works". There would have been less brokeness if they had done it right at the right time.

Because we have lots of users using `()` I want to get a refactoring tool which converts `()` into `[]`.

Also it is a good idea to refactor some of that old code or remove the crap.

I think it is urgent to skip this old compile opt feature for the old style as soon as possible. Yes I don't want it to be kept!

>
> For IDL 8.0, we want the new language features like the "dot" to work
> "out of the box". So, ideally, we would change the default compile_opt
> for IDL to be "idl2" - this includes both "defint32" and "strictarr".

> All integer scalar constants would be 32-bit by default, and
> parentheses would not be usable for array indexing. However, this
> creates a backwards compatibility problem. IDL .save files would be
> fine (the code is already compiled), but large libraries (like JHUAPL)
> would not be usable without changes to the code.
>
>
> 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.

No

see above

>
> 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.

No

see above
>
> 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.
>

No

see above

> 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

see above

> 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.
>
>

hmm No

At first glance it sounds like it can be the output of the refactored
old sources. And the old pro files were interpreted by a builtin idl7.

But that shifts the problem only into a different extension. You have by
this a compile opt by a file extension.

So I disagree - This is also no solution

> We would really like to hear your opinion on these potential
> solutions. Questions to think about:

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

Can't estimate

The code I am responsible for could be refactored by me. I like to have
a tool which helps to do it and which I don't have to write on my own.

>
> 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?

We use these libraries. One problem is that most of the code has only
doctests as examples. If you convert these libs how do you test that you
did it correct?

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

For our sources it is "only problem with parentheses for arrays".

>

> Thanks for reading all of this, and we look forward to your replies.
> Happy Holidays!

Please also add to idl8 features we urgent need. netCDF4, svg output.
Otherwise we don't need to do this effort.

best regards

Reimar

Subject: Re: IDL 8.0 compile_opt changes
Posted by chris_torrence@NOSPAM on Mon, 04 Jan 2010 19:41:49 GMT
View Forum Message <> Reply to Message

>
> Please also add to idl8 features we urgent need. netCDF4, svg output.
> Otherwise we don't need to do this effort.
>
> best regards
>
> Reimar

Hi Reimar,

NetCDF 4 is available as a patch to IDL 7.1, on the ittvis.com ->
Downloads -> Product Downloads -> IDL 7.1. The patch also includes an
update to HDF5 1.8.3 and CDF 3.3.

Cheers,

Chris
ITTVIS

Subject: Re: IDL 8.0 compile_opt changes
Posted by David Fanning on Mon, 04 Jan 2010 19:58:52 GMT
View Forum Message <> Reply to Message

Chris Torrence wites:

> NetCDF 4 is available as a patch to IDL 7.1, on the ittvis.com ->
> Downloads -> Product Downloads -> IDL 7.1. The patch also includes an
> update to HDF5 1.8.3 and CDF 3.3.

Chris,

I am confused about the updates. I see my IDL Windows version is
IDL 7.1.2, but all of the Windows updates listed on that page are
7.1.1. Where did I get my Windows 7.1.2 upgrade? And does this
include the NetCDF 4 stuff, or do I need to download that, too?

Confused,

David

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

Subject: Re: IDL 8.0 compile_opt changes
Posted by R.Bauer on Tue, 05 Jan 2010 14:09:37 GMT
View Forum Message <> Reply to Message

Reimar Bauer schrieb:
> Chris Torrence schrieb:
>> Hi all,
>>
>> 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.
>>
>
> Hi Chris
>
> Happy new year to you and ittvis. Thanks for asking us.
>
>> 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:
>>
>>   p = PLOT(x,y)
>>   child = p.Get(index)
>
> I support this change. I felt always the attempt of doing OO in idl is
> extremly uncomfortable and ugly compared to python.
> I think this change is important for new users or old users using OO in

> different languages.
>

if I see this right I can't have a var plot or do we have now case sensitives vars too?

That is only a minor issue in comparison that I am quite sure the icg idl plotlib becomes broken by removing the old direct graphics methods anyway.

e.g.
http://www.fz-juelich.de/icg/icg-1/idl_icglib/idl_source/idl _work/fr_lib/creaso_examples/example7.pro
http://www.fz-juelich.de/icg/icg-1/idl_icglib/idl_source/idl _html/gif/icon_example7.pro.png

or
http://www.fz-juelich.de/icg/icg-1/idl_icglib/idl_source/idl _work/rb_lib/examples/plot/plotxy/plotxy_d_bsp2_p.pro
http://www.fz-juelich.de/icg/icg-1/idl_icglib/idl_source/idl _html/gif/icon_plotxy_d_bsp2_p.pro.gif

Almost all of our publications are based on that library.

Reimar

Subject: Re: IDL 8.0 compile_opt changes
Posted by pgrigis on Tue, 05 Jan 2010 14:57:22 GMT
View Forum Message <> Reply to Message

On Jan 5, 9:09 am, Reimar Bauer <R.Ba...@fz-juelich.de> wrote:
> Reimar Bauer schrieb:
>
>
>
>> Chris Torrence schrieb:
>>> Hi all,
>
>>> 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.
>
>> Hi Chris
>
>> Happy new year to you and ittvis. Thanks for asking us.
>
>>> 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:
>
>>>   p = PLOT(x,y)
>>>   child = p.Get(index)
>
>> I support this change. I felt always the attempt of doing OO in idl is
>> extremly uncomfortable and ugly compared to python.
>> I think this change is important for new users or old users using OO in
>> different languages.
>
> if I see this right I can't have a var plot or do we have now case
> sensitives vars too?
>
> That is only a minor issue in comparison that I am quite sure the icg
> idl plotlib becomes broken by removing the old direct graphics methods
> anyway.
>
>   e.g.http://www.fz-juelich.de/icg/icg-1/idl_icglib/idl_source
/idl_work/fr_...http://www.fz-juelich.de/icg/icg-1/idl_icgli b/idl_source/idl_html/gif...
>
>   orhttp://www.fz-juelich.de/icg/icg-1/idl_icglib/idl_source/i
dl_work/rb_...http://www.fz-juelich.de/icg/icg-1/idl_icglib/ idl_source/idl_html/gif...
>
> Almost all of our publications are based on that library.
>
> Reimar

Hi Reimar - as of now I believe there's no conflict
between structures and functions.

You can have a structure called

sqrt={a:1,b:2}

and the function sqrt() will still work,
for instance the following slightly
obfuscated command works well:

IDL> print,sqrt(sqrt.b)
      1.41421

So why do you expect a conflict in the new version
between your structure "plot" and a function plot()
that returns an object?

Ciao,
Paolo

Subject: Re: IDL 8.0 compile_opt changes
Posted by chris_torrence@NOSPAM on Tue, 05 Jan 2010 18:57:54 GMT
View Forum Message <> Reply to Message

On Jan 4, 12:58 pm, David Fanning <n...@dfanning.com> wrote:
> Chris Torrence wites:
>
>> NetCDF 4 is available as a patch to IDL 7.1, on the ittvis.com ->
>> Downloads -> Product Downloads -> IDL 7.1. The patch also includes an
>> update to HDF5 1.8.3 and CDF 3.3.
>
> Chris,
>
> I am confused about the updates. I see my IDL Windows version is
> IDL 7.1.2, but all of the Windows updates listed on that page are
> 7.1.1. Where did I get my Windows 7.1.2 upgrade? And does this
> include the NetCDF 4 stuff, or do I need to download that, too?
>
> Confused,
>
> David
>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming:http://www.dfanning.com/
> Sepore ma de ni thue. ("Perhaps thos speakest truth.")

I think you must have gotten IDL 7.1.2 with ENVI 4.7?

Anyway, you can check if you have the NetCDF patch by doing:
  help,'ncdf',/dlm
If it says version 3.x then you do not have the patch yet.

-Chris

Subject: Re: IDL 8.0 compile_opt changes
Posted by chris_torrence@NOSPAM on Tue, 05 Jan 2010 18:58:43 GMT
View Forum Message <> Reply to Message

On Jan 5, 7:09 am, Reimar Bauer <R.Ba...@fz-juelich.de> wrote:

> Reimar Bauer schrieb:
>
>
>
>> Chris Torrence schrieb:
>>> Hi all,
>
>>> 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.
>
>> Hi Chris
>
>> Happy new year to you and ittvis. Thanks for asking us.
>
>>> 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:
>
>>> p = PLOT(x,y)
>>> child = p.Get(index)
>
>> I support this change. I felt always the attempt of doing OO in idl is
>> extremly uncomfortable and ugly compared to python.
>> I think this change is important for new users or old users using OO in
>> different languages.
>
> if I see this right I can't have a var plot or do we have now case
> sensitives vars too?
>
> That is only a minor issue in comparison that I am quite sure the icg
> idl plotlib becomes broken by removing the old direct graphics methods
> anyway.
>
> e.g.http://www.fz-juelich.de/icg/icg-1/idl_icglib/idl_source
/idl_work/fr_...http://www.fz-juelich.de/icg/icg-1/idl_icgli b/idl_source/idl_html/gif...
>
> orhttp://www.fz-juelich.de/icg/icg-1/idl_icglib/idl_source/i
dl_work/rb_...http://www.fz-juelich.de/icg/icg-1/idl_icglib/ idl_source/idl_html/gif...
>
> Almost all of our publications are based on that library.
>
> Reimar

Hi Reimar,

We are not removing anything from IDL 8.0. All of the direct graphics routines are still there, and will work as they did before.
Cheers,
Chris

Subject: Re: IDL 8.0 compile_opt changes
Posted by David Fanning on Tue, 05 Jan 2010 19:17:42 GMT
View Forum Message <> Reply to Message

Chris Torrence wites:

> I think you must have gotten IDL 7.1.2 with ENVI 4.7?

No, I asked ITTVIS technical support for it when Wayne Landsman reported that Scope_Var_Fetch bug. They gave me this link:

http://www.ittvis.com/download/download_splash.asp?wdiid=157 2&si=225907

As it happens, the NetCDF patches were NOT in that release, I don't think. But I did download the IDL 7.1sdf patch and that just installed new netCDF and HDF DLLs in my IDL bin directory.

> Anyway, you can check if you have the NetCDF patch by doing:
> help,'ncdf',/dlm
> If it says version 3.x then you do not have the patch yet.

OK, good. I have the new version installed. Now what do I do? Is there a bit of documentation around somewhere? My impression is that netCDF 4 is *very* different from netCDF 3. Or, are these new files smart enough to figure everything out on their own. :-)

Cheers,

David

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

Subject: Re: IDL 8.0 compile_opt changes
Posted by monkman on Tue, 05 Jan 2010 21:51:39 GMT
View Forum Message <> Reply to Message

At LASP we have an IDL code base of at least several 100K
lines of code. Much of this code is used in operational
satellite environments, much was written by people whose
main focus isn't programming, and some is very old.
We don't have the resources to look through all of it,
fix what needs to be fixed, test and re-release it all.
Backwards incompatible changes would prevent us from
upgrading to IDL 8.0.

Changing the default integer size from 2 bytes to 4 bytes
would break a lot of our code which deals with a binary
nterface: writing Sybase bcp files, reading CCSDS packets
from a socket, certain bit manipulation code, call_external
etc.

Adding a compile_opt idl1 at the top of every .pro file
wouldn't compile under any IDL version less than than
IDL 8.0, and we run the same code with several versions
of IDL. So this isn't a solution.

Has ITTVIS looked into this possible solution?

What about a new IDL 8.0 command-line option (or a new
built-in system variable) to specify backwards compatibility?
Say if this option were set (or the system variable has
a certain value set by the users IDL startup file), then
the compiler would behave in a backwards compatible way:
parentheses OK for arrays, default ints are 2-bytes.
A given .pro file could then override this by specifying
compile_opt strictarr, and be able to compile new language
features.
If on the other hand, the backwards compatibility option
were *not* set, then the IDL 8.0 compiler would assume
compile_opt idl2 as desired. (But I still don't understand
why changing the default to defint32 is necessary, when the
planned language changes only require a change to strictarr)

If this solution works, the only necessary changes would be
to modify an IDL startup file, or shell scripts which
start IDL. These would provide the new command-line option
or system variable setting to specify backwards compatibility.
This would be much more feasible than changing every file in
the entire code base. And new users of IDL wouldn't have to
set the option, and thus would have the new language features
by default.

-Steve Monk
LASP

Subject: Re: IDL 8.0 compile_opt changes
Posted by JDS on Tue, 05 Jan 2010 22:41:09 GMT
View Forum Message <> Reply to Message

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

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.

Subject: Re: IDL 8.0 compile_opt changes
Posted by Maarten[1] on Wed, 06 Jan 2010 14:01:27 GMT
View Forum Message <> Reply to Message

A lot has been said about the subject, but I want to add a few thoughts as well. This seems to be an appropriate time.

On Dec 18 2009, 10:51 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:
>
> p = PLOT(x,y)

>   child = p.Get(index)
>
> Disregarding the actual syntax, the problem occurs when IDL tries to
> compile the second line of code. By default it will think that "Get"
> is just an array field inside of a structure, and that the parentheses
> are just indexing into the array.
>
> Now, you could use "compile_opt idl2", or "compile_opt strictarr" to
> remove the ambiguity. In this case, for IDL 8.0, it will then know
> that this is a function method call.

There is a lot in IDL that is decidedly _not_ industry standard
(calling a function and discarding the result, the way procedures are
called (as opposed to functions), character handling, named
parameters, manual line breaking (when a smarter parser could figure
out that a $ is still needed, ...). Are there better arguments to
require this change?

Don't get me wrong: _finally_ dusting off some parts of IDL is good in
my opinion. But one of the important reasons people use IDL is the
existing code, not its particularly pretty syntax.

> For IDL 8.0, we want the new language features like the "dot" to work
> "out of the box". So, ideally, we would change the default compile_opt
> for IDL to be "idl2" - this includes both "defint32" and "strictarr".
> All integer scalar constants would be 32-bit by default, and
> parentheses would not be usable for array indexing. However, this
> creates a backwards compatibility problem. IDL .save files would be
> fine (the code is already compiled), but large libraries (like JHUAPL)
> would not be usable without changes to the code.

Others have commented on this in depth. Some changes can be automated,
at least in part. For some of the really old code you have to wonder
if leaving it to run on old IDL would be a bad thing.

While you're at it: I'd like to have an option to use double precision
by default, only changing back to single precision when I explicitly
ask for it.

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

I think that these days (especially for new users) there are more
issues caused by the default short integers and single precision
values than the reverse. This excludes of course direct binary
reading, but for reading binary files you should have been explicit
from day one.

I do love the idea of negative indices, although I'd like them to mean the same as in Python. The samples Ive seen so far are off by one.

Best,

Maarten

Subject: Re: IDL 8.0 compile_opt changes
Posted by Kenneth P. Bowman on Wed, 06 Jan 2010 14:52:25 GMT
View Forum Message <> Reply to Message

In article
<5e4a2c1a-4de9-4a4e-8399-f5ec724e150c@j5g2000yqm.googlegroups.com>,
JD Smith <jdtsmith.nospam@yahoo.com> wrote:

> 1. Overloading structure dereference and method invocation breaks the
> ability to semantically parse a.

I have been following this discussion and trying to understand the implications. I don't claim any expertise in compiler construction or parsing methods. I think I agree with JD on this one, though.

It seems to me that the fundamental problem with parentheses in IDL is that they can mean two different things (array subscripting or function referencing). This is partially remedied by allowing (but not absolutely requiring) square brackets for subscripting.

Now the idea is to create a similar ambiguity with the dot operator? And the main justification is to make IDL read more like Python? As someone who uses structures a *lot*, but not objects, I can see the potential for serious confusion, by the programmer, if not by the parser. I can also hardly wait to try teaching this to beginning programmers.

> "The dot operator is used to indicate a structure field, except
> it can also mean to use an object method. What's an object
> method? Oh, we haven't gotten to that yet, but when you read
> someone else's programs, make sure you know the difference."

I love JD's example

> self.limit, self.limit

A compiler option to force double precision floating point constants and variables would be very useful to me (DEFFLT64?).

If this change is radical enough to require a new file suffix,
shouldn't the opportunity be taken to change so many other
"minor" issues, like default 2-byte integers?

Cheers, Ken

Subject: Re: IDL 8.0 compile_opt changes
Posted by Yngvar Larsen on Wed, 06 Jan 2010 14:56:08 GMT
View Forum Message <> Reply to Message

My 2 cents:

*) About negative indexing:

> For example:
>
> a[0:-1] In IDL80, returns all the elements
> a[-1] In IDL80, returns the last element

Is the only use for negative indexing a way to indicate the last
element? Then this is probably the most idiotic choice possible:
semantic overload of the integer -1 with no gain whatsoever (except
possibly in ITTVis' parsing code that might be simplified by some
clever tricks?). Why not use some other syntactic element like the
string 'end' (like matlab) or 'last' or '%' or whatever?

Or is there some other clever use? Reverse/step indexing like matlab's
"array(first:step:last)" with negative steps allowed would be
fantastic, but this doesn't require negative indices at all. I
propose:
IDL8> array = indgen(5)
IDL8> print, array[%:1:-2] ; array[first:last:step] like the FOR loop
indexing
4 2
IDL8> print, array[%]
4
IDL8> print, array[%/2]
2

etc

*) about '.' instead of '->':

NO,NO,NO!

IDL is not Python, as some people have pointed out. Any syntax that
requires access to the dynamic runtime to be resolved is a Bad Idea,

which is why the transition from () to [] was a Good Idea. I agree that '.' looks better than '->', but unless access to the internal 'self' structure is allowed from outside of objects, like in Python, I see absolutely no point in looking superficially like Python. That would in fact be _more_ confusing since it is not the same. Semantics is more important than syntax at the end of the day.

*) about legacy libraries:

> 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?

That would be the best solution to avoid scaring the many(?) users of these libraries from upgrading. And how about a tool in the IDE to convert legacy code from old to new style, i.e. () to [] and -> to '.' if this is chosen, optionally interactively. Also maybe including indicating common pitfalls like formatted IO for manual inspection. Should be easy to get 99% right as many already commented on.

*) about default 'idl2' breaking code:

In my opinion, code that is broken by this (mostly formatted IO for binary files of a specific format without specifying explicit datatypes) is in fact already broken by design. Or at least badly written.

I vote for default 'idl2'. 13 years of transition should be sufficient...

*) about a new extension for >=8.0 code: I hate the idea, though it solves the problem. I like better the idea of using the comment character e.g. ;%compile_opt idl1. It is butt ugly, but so is the code that needs it :)

*) about other issues for 8.0 (according to http://michaelgalloy.com/2009/12/28/agu-idl-users-group-meet ing.html#more-2092):

- operator overloading: nice!!
- null elements for arrays and structures: nice!! ('[]' and '{}'?)
- FOREACH loop: very useful. Even better if new data types like maps/lists/hashtables are planned.

Wishlist:
- More general datatypes like map/list/hashtable/tuple etc.
- I wish a scalar and an array with one element could be treated as different datatypes. Mathematically, they are different, so sometimes confusing bugs are caused by conflating them like IDL currently does.

- object based widgets anyone? And when will the new widgets used for
the Workbench be available for users? Motif looks so last millenium...
- some kind of package system or namespaces to avoid having to call my
own library functions/procedures/objects something like
" reallylongstringtominimizeprobabilityfornameclashes_myfuncti on "

--
Yngvar

Subject: Re: IDL 8.0 compile_opt changes
Posted by Yngvar Larsen on Wed, 06 Jan 2010 15:12:47 GMT
View Forum Message <> Reply to Message

On Jan 6, 3:56 pm, Yngvar Larsen <larsen.yng...@gmail.com> wrote:
> My 2 cents:
>
> *) About negative indexing:
>
>> For example:
>
>> a[0:-1] In IDL80, returns all the elements
>> a[-1] In IDL80, returns the last element
>
> Is the only use for negative indexing a way to indicate the last
> element? Then this is probably the most idiotic choice possible:
> semantic overload of the integer -1 with no gain whatsoever (except
> possibly in ITTVis' parsing code that might be simplified by some
> clever tricks?).

Hm. Quick followup to myself: I forgot about how Python uses negative
integers (seq[-n] == seq[length(seq)-n]), which is probably the
intended syntax here? Then at least it makes kind of sense.

--
Yngvar

Subject: Re: IDL 8.0 compile_opt changes
Posted by David Fanning on Wed, 06 Jan 2010 15:13:34 GMT
View Forum Message <> Reply to Message

Yngvar Larsen writes:

> - object based widgets anyone?

I'll donate the Catalyst Library. :-)

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 Yngvar Larsen on Wed, 06 Jan 2010 15:29:09 GMT
View Forum Message <> Reply to Message

On Jan 6, 3:56 pm, Yngvar Larsen <larsen.yng...@gmail.com> wrote:

> - I wish a scalar and an array with one element could be treated as
> different datatypes. Mathematically, they are different, so sometimes
> confusing bugs are caused by conflating them like IDL currently does.

Hm. I should read through before posting. Another followup to myself:
What I wrote above is not correct.

```
IDL> a = indgen(5)
IDL> print, 3*a
      0       3       6       9      12      15
IDL> print, [3]*a
      0
```

What I was thinking about is that some IDL functions seem to convert
arrays to scalars if they have length one. Cannot find a good example
right now. Hm.

--
Yngvar

Subject: Re: IDL 8.0 compile_opt changes
Posted by Matt[2] on Wed, 06 Jan 2010 16:18:06 GMT
View Forum Message <> Reply to Message

Yngvar Larsen <larsen.yngvar@gmail.com> writes:

> *) about a new extension for >=8.0 code: I hate the idea, though it
> solves the problem. I like better the idea of using the comment
> character e.g. ;%compile_opt idl1. It is butt ugly, but so is the code
> that needs it :)

I agree here 100%.

I hadn't thought of JD's problem with overloading the '.' operator before. That seems like it could cause problems with idlwave and maybe the workbench?

But really, a decent converter tool is what needs to be provided to allow everyone to feel comfortable upgrading. (Although, I note that the lasp people say they'll run on multiple IDL version and that may cause problems.) Of course the tool could have a "backwards compatability" where it just adds the ;%compile_opt idl1 and a "break everything from before" where it changes the () to [] and replaces your -> with dots.

Just another opinion.

Matt
--
Matthew Savoie - Scientific Programmer
National Snow and Ice Data Center
(303) 735-0785 http://nsidc.org

Subject: Re: IDL 8.0 compile_opt changes
Posted by penteado on Wed, 06 Jan 2010 18:35:13 GMT
View Forum Message <> Reply to Message

I realised now that I had not thought this through:

On Jan 5, 9:54 pm, pp <pp.pente...@gmail.com> wrote:
>> 1. Overloading structure dereference and method invocation breaks the
>> ability to semantically parse a.b.
> 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.

This is not actually a problem. It would just be the same kind of

ambiguity that happens in the middle of typing a name, when there are more than one that begin with that same characters. At that point, using completion should give the user both options. In the case of the function/member ambiguity, the options would be "a.b(" and "a.b", as the first is the function, and the second is the structure member.

> 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.

I had not remembered that in C++ and Java it is not possible to have the same name associated to a function and a variable at the same time. In Pyhton, the last use for a name rebinds it ("overwrites" the previous use), so there is only one thing with the name. So there is no ambiguity in them. In IDL, assuming the idl2 option would remove the ambiguity through the syntax, so no problems either.

So, yes, it is a bit easier when reading a program to resolve the meaning through a different operator, as it is now. But I find it only slightly easier than looking at the syntax, so not much of an issue. Contrary to what some people have written in this thread, this is not the same as the ()/[] issue, as only looking at a line written using () is not enough to know if it is a function or an array. In the case of the dot operator, assuming the idl2 option, there would not be any ambiguities in complete lines of code.

The only situation with an ambiguity would be the one mentioned by JD, in the middle of writing a line, when using autocompletion. But in those moments there is the unavoidable ambiguity from having several names that begin with the same characters anyway.

It could be argued that the dot is more proper because both data and functions are conceptually the same, they are "parts" of an object, and perhaps this is the reason for its choice in C++, Java and Python. So I do not see much difference between using either operator. Either is fine, as long as the idl2 option is assumed if the dot operator is used.

However, in the separate question of how to assume the idl2 compile option, which is already needed, even without the dot operator change, there are very important implications. Which is why I previously argued that a new extension is the best choice. The new extension is better even than the commented idl1 option, since no change to old files is required, and it prevents old IDL versions trying to interpret code that they cannot understand properly (as in the issue #2 mentioned by JD).

Subject: Re: IDL 8.0 compile_opt changes
Posted by Michael Galloy on Wed, 06 Jan 2010 21:25:29 GMT
View Forum Message <> Reply to Message

On 1/6/10 7:01 AM, Maarten wrote:
> I do love the idea of negative indices, although I'd like them to mean
> the same as in Python. The samples Ive seen so far are off by one.

My understanding of the negative indices proposal in IDL was that they would be the same as in Python:

```
 >>> a = [1, 2, 3, 4]
 >>> a[-1]
4
 >>> a[-2]
3
```

By the way, this could break old code as well. It also seems like a better reason for breaking backward compatibility than the "." as a method invocation. And once backwards compatibility is broken, then we might as well make all the changes we need at once (as long as we have a good conversion tool that makes most of the changes automatically).

Mike
--
www.michaelgalloy.com
Research Mathematician
Tech-X Corporation

Subject: Re: IDL 8.0 compile_opt changes
Posted by Michael Galloy on Wed, 06 Jan 2010 21:52:44 GMT
View Forum Message <> Reply to Message

On 1/6/10 7:56 AM, Yngvar Larsen wrote:
> *) about other issues for 8.0 (according to
> http://michaelgalloy.com/2009/12/28/agu-idl-users-group-meet ing.html#more-2092):
>
> - operator overloading: nice!!
> - null elements for arrays and structures: nice!! ('[]' and '{}'?)

This is also another feature that would have implications for backwards compatibility because I normally use N_ELEMENTS to indicate if a variable is undefined, but n_elements([]) eq 0. So an empty array or structure would show up as undefined in a lot of my code. Not sure if that would really be a problem or not. I suppose I should use SIZE(/TYPE) eq 0 to determine if a variable is undefined? But maybe SIZE([], /TYPE) would actually be 0 too? Some of the details about how

these things are implemented are actually pretty important.

> - FOREACH loop: very useful. Even better if new data types like maps/
> lists/hashtables are planned.

This would be really nice. I would love to not have to bring in my own objects to do simple things like lists and dictionaries.

> Wishlist:
> - More general datatypes like map/list/hashtable/tuple etc.
> - I wish a scalar and an array with one element could be treated as
> different datatypes. Mathematically, they are different, so sometimes
> confusing bugs are caused by conflating them like IDL currently does.
> - object based widgets anyone? And when will the new widgets used for
> the Workbench be available for users? Motif looks so last millenium...

Yes, that would be really nice. That was on the roadmap last spring.
Maybe in 8.1?

> - some kind of package system or namespaces to avoid having to call my
> own library functions/procedures/objects something like
> " reallylongstringtominimizeprobabilityfornameclashes_myfuncti on "

Mike
--
www.michaelgalloy.com
Research Mathematician
Tech-X Corporation

Subject: Re: IDL 8.0 compile_opt changes
Posted by wallabadah on Thu, 07 Jan 2010 00:30:18 GMT
View Forum Message <> Reply to Message

I agree with JD:

> 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

As Ken has pointed out, it was necessary to change to square brackets for array indexing to avoid ambiguity. Introducing a new ambiguity by allowing the dot operator to access both structure elements and object

functions is definitely a Bad Idea. Apart from breaking code completion as pointed out by JD, it makes code much harder to read - whoever wrote "I still find it clear from the syntax the meaning of the second line" is having a joke with us - this is step 1 of code obfuscation. I also think the -> operator visually depicts what it does, making the intention of the author absolutely clear when reading code. I don't think changing the language's syntax so that it is more like another language is a valid reason. Assuming introduction of the dot operator in IDL8.0 doesn't break ->, we're in a situation where there are two operators to do the same thing. Will ITTVIS deprecate -> in IDL10.0, requiring yet another update to everyone's code libraries. Regardless, with operator overloading those die-hards that want '.' to mean '->' could implement it in their own objects themselves.

On the subject of updating parentheses in old libraries - if the IDL interpreter/compiler is capable of discerning the different intent of () when parsing idl code into byte code - shouldn't it be possible to use this information to update the libraries (with zero human intervention)? I don't have a great understanding of how byte code is generated, but if the code can be compiled and run correctly by IDL, then IDL 'knows' whether the parentheses in 'old' code should be changed to square brackets. If this assumption is correct, updating () to [] in old code should be built in to IDL 8.0.

Will.

Subject: Re: IDL 8.0 compile_opt changes
Posted by Maarten[1] on Thu, 07 Jan 2010 15:27:08 GMT
View Forum Message <> Reply to Message

On Jan 6, 10:25 pm, mgalloy <mgal...@gmail.com> wrote:
> On 1/6/10 7:01 AM, Maarten wrote:
>
>> I do love the idea of negative indices, although I'd like them to mean
>> the same as in Python. The samples Ive seen so far are off by one.
>
> My understanding of the negative indices proposal in IDL was that they
> would be the same as in Python:
>
>>>> a = [1, 2, 3, 4]
>>>> a[-1]
> 4
>>>> a[-2]
> 3

Yes, in this case it is the same. The (subtle) difference comes in for ranges.

Python:
```
>>>  a = [1,2,3,4]
>>>  a[-1]
4
>>>  a[-2]
3
>>>  a[1:-1]
[2, 3]
```

The last one is the one I'm concerned about, as python does not include the last index in the range.

```
IDL>  a = [1,2,3,4]
IDL> print, a[3]
          4
IDL> print, a[1:3]
          2          3          4
```

(index 3 is equivalent to index -1).

Now you could say that Python and IDL already disagree here, but the off-by-one is worth mentioning anyway.

> By the way, this could break old code as well. It also seems like a
> better reason for breaking backward compatibility than the "." as a
> method invocation.

Agreed.

> And once backwards compatibility is broken, then we
> might as well make all the changes we need at once (as long as we have a
> good conversion tool that makes most of the changes automatically).

A conversion tool may not have to go forward. It may be easier to go backward with a tool. Especially since this allows for cleaning up the syntax. It will also prompt users to write code for the newer system, and let a tool worry about backward compatibility (compare to Python 3).

Maarten

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

Subject: Re: IDL 8.0 compile_opt changes
Posted by Michael Galloy on Thu, 07 Jan 2010 17:56:42 GMT
View Forum Message <> Reply to Message

On 1/7/10 8:27 AM, Maarten wrote:
> On Jan 6, 10:25 pm, mgalloy<mgal...@gmail.com> wrote:
>> On 1/6/10 7:01 AM, Maarten wrote:
>>
>>> I do love the idea of negative indices, although I'd like them to mean
>>> the same as in Python. The samples Ive seen so far are off by one.
>>
>> My understanding of the negative indices proposal in IDL was that they
>> would be the same as in Python:
>>
>>>> > a = [1, 2, 3, 4]
>>>> > a[-1]
>> 4
>>>> > a[-2]
>> 3
>
> Yes, in this case it is the same. The (subtle) difference comes in for
> ranges.
>
> Python:
>>>> a = [1,2,3,4]
>>>> a[-1]
> 4
>>>> a[-2]
> 3
>>>> a[1:-1]
> [2, 3]
>
> The last one is the one I'm concerned about, as python does not
> include the last index in the range.
>
> IDL> a = [1,2,3,4]
> IDL> print, a[3]
> 4
> IDL> print, a[1:3]
> 2 3 4
>
> (index 3 is equivalent to index -1).

>
> Now you could say that Python and IDL already disagree here, but the
> off-by-one is worth mentioning anyway.

I think we are agreeing here, but just to be sure: Python and IDL would
be specifying the endpoints of the range in the same way, it's just that
Python always includes the start index and excludes the end index (even
if not using negative indices):

```
 >>> a = [1, 2, 3, 4]
 >>> a[1:3]
    [2, 3]
```

Mike
--
www.michaelgalloy.com
Research Mathematician
Tech-X Corporation

Subject: Re: IDL 8.0 compile_opt changes
Posted by H. Evans on Fri, 08 Jan 2010 09:29:51 GMT
View Forum Message <> Reply to Message

On Jan 7, 5:35 pm, Paul van Delst <paul.vande...@noaa.gov> wrote:
> 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)

Not all of us >:-(. I still consider it a bone-headed decision. The
"%" sign as a delimiter is atrocious graphically. An ideal delimiter
is simple, e.g. '.', '_', '-', ' ', allowing the eye to quickly

identify it and use it to separate the surrounding text. Then there is all the legacy code that's 'broken'. Which is easier to quickly parse by eye: This.that, This_that, This-that, This%that. I'd love to know what the compelling argument was over the previous glorious DEC Fortran standard of '.'. Back on topic: the IDL '->' is quite reasonable in this respect.

The file extension is something that's easier to understand and live with on a daily basis. It's a single 'configuration' of the source code file at a single instance in time: it's creation.

>
> 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.
>

Hallelujah! I agree completely. Is there a better reason for migration to '.' from '->' other than it might be 'easier for new users' as I consider that argument a bit specious.

If we are rewriting the syntax in a big way, how about replacing the '$' continuation line character for the industry standard' semi-colon for command delimiting, and then using the '//' characters for comment delimiting.

> 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.:
>
And
- Extensions to the histogram function to calculate histograms
of weighted values - not all data is created equal,
e.g. h = histogram(x, weights=w), where it is the "w" elements
that are tallied in the bins, and not the number of 'x's in the bin.

- Anti-aliasing built into the direct graphics functions (or am I the only person
left that finds the object graphics to be overly complex and tedious to
produce a simple plot, while the direct graphics quality not of publication
quality?) Alternatively, a complementary function for the old direct graphics
routines that use object graphics, e.g. dgplot as a direct replacement of
plot. Particularly one that can be used in batch mode (no attached X

or window
device - a problem with using the iTools for batch processing).

- definitely support localisation of namespaces for common blocks/ functions. Ok,
perhaps we should be moving over to using objects anyway to encapsulate, but
even then there is name space clashing potential for object names.

- an ability to run dynamic library (call_external, dlm, etc) routines in a multi-threaded
execution space. I'm currently using a bridge object, but it's a bit hit/miss. Shared
memory in these bridge objects for strings, etc?

- more functionality on accessing the contents of SAVE files (delete variables, overwrite,
extract just a single named one, rename contents, etc.)

H.

Subject: Re: IDL 8.0 compile_opt changes
Posted by Maarten[1] on Fri, 08 Jan 2010 12:19:45 GMT
View Forum Message <> Reply to Message

On Jan 7, 6:56 pm, mgalloy <mgal...@gmail.com> wrote:
> On 1/7/10 8:27 AM, Maarten wrote:
>
>
>
>> On Jan 6, 10:25 pm, mgalloy<mgal...@gmail.com> wrote:
>>> On 1/6/10 7:01 AM, Maarten wrote:
>
>>>> I do love the idea of negative indices, although I'd like them to mean
>>>> the same as in Python. The samples Ive seen so far are off by one.
>
>>> My understanding of the negative indices proposal in IDL was that they
>>> would be the same as in Python:
>
>>>> a = [1, 2, 3, 4]
>>>> a[-1]
>>> 4
>>>> a[-2]
>>> 3
>
>> Yes, in this case it is the same. The (subtle) difference comes in for
>> ranges.

>
>> Python:
>>>> > a = [1,2,3,4]
>>>> > a[-1]
>> 4
>>>> > a[-2]
>> 3
>>>> > a[1:-1]
>> [2, 3]
>
>> The last one is the one I'm concerned about, as python does not
>> include the last index in the range.
>
>> IDL> a = [1,2,3,4]
>> IDL> print, a[3]
>> 4
>> IDL> print, a[1:3]
>> 2 3 4
>
>> (index 3 is equivalent to index -1).
>
>> Now you could say that Python and IDL already disagree here, but the
>> off-by-one is worth mentioning anyway.
>
> I think we are agreeing here, but just to be sure: Python and IDL would
> be specifying the endpoints of the range in the same way, it's just that
> Python always includes the start index and excludes the end index (even
> if not using negative indices):
>
>>>> a = [1, 2, 3, 4]
>>>> a[1:3]
> [2, 3]

Yes. Although this is a fundamental difference that is the result of a choice both language developers made. Thinking about it a bit longer, I don't think the two can be made to act the same: IDL always includes the end index of the range, while Python always excludes it. Some emphasis on this in the documentation may be needed, as Python probably is the most widespread programming language that offers the facility of negative indices.

Maarten

Subject: Re: IDL 8.0 compile_opt changes
Posted by penteado on Fri, 08 Jan 2010 13:36:47 GMT

On Jan 8, 7:29 am, "H. Evans" <bloggs...@googlemail.com> wrote:
> - Anti-aliasing built into the direct graphics functions (or am I the
> only person
> left that finds the object graphics to be overly complex and tedious
> to
> produce a simple plot, while the direct graphics quality not of
> publication
> quality?) Alternatively, a complementary function for the old direct
> graphics
> routines that use object graphics, e.g. dgplot as a direct
> replacement of
> plot. Particularly one that can be used in batch mode (no attached X
> or window
> device - a problem with using the iTools for batch processing).

To me it seems that the iTools are designed to be near direct replacements of their direct graphics counterparts. It is possible to create iTools non-interactively and without an X server, just by using the keywords:

user_interface='none',/no_saveprompt,/disable_splash

Since IDL 7.1, there has been a much easier to use procedural interface to the iTools, which includes things like iopen, isave, and idelete. I always found that these new features should have been better advertised when they were introduced. See

http://michaelgalloy.com/2009/08/15/idl-7-1-itools-procedura l-api.html
http://www.dfanning.com/itool_tips/invisible.html

For some examples.

> - more functionality on accessing the contents of SAVE files (delete
> variables, overwrite,
> extract just a single named one, rename contents, etc.)

It is already possible to read select variables from save files, even select heap variables in them, which I have used to obtain only the targets of select elements in pointer arrays. This is done with the IDL_Savefile object. Though I would still like some more of the CMSVLIB functionality to be built-in.

Subject: Ruby range operators? Re: IDL 8.0 compile_opt changes
Posted by Paul Van Delst[1] on Fri, 08 Jan 2010 21:16:09 GMT

Maarten wrote:
> On Jan 7, 6:56 pm, mgalloy <mgal...@gmail.com> wrote:
>> I think we are agreeing here, but just to be sure: Python and IDL would
>> be specifying the endpoints of the range in the same way, it's just that
>> Python always includes the start index and excludes the end index (even
>> if not using negative indices):
>>
>>>> > a = [1, 2, 3, 4]
>>>> > a[1:3]
>> [2, 3]
>
> Yes. Although this is a fundamental difference that is the result of a
> choice both language developers made. Thinking about it a bit longer,
> I don't think the two can be made to act the same: IDL always includes
> the end index of the range, while Python always excludes it. Some
> emphasis on this in the documentation may be needed, as Python
> probably is the most widespread programming language that offers the
> facility of negative indices.

Well, since they're mucking about with operators in general, maybe ITTVIS could go the ruby route and introduce the ".." and "..." range operators. The former is an inclusive range (same functionality as ":") and the latter is a range that excludes the higher value. So,

```
$ irb
irb(main)> a = [1,2,3,4,5,6]
=> [1, 2, 3, 4, 5, 6]

irb(main)> a[1..3]
=> [2, 3, 4]

irb(main)> a[1...3]
=> [2, 3]

irb(main)> a[1..-1]
=> [2, 3, 4, 5, 6]

irb(main)> a[1...-1]
=> [2, 3, 4, 5]
```

BTW, if IDL 8.0 will allow operator overloading, will it also allow for operator definition? The overloading should allow for ".." having the same result as ":", but will we be able to define functions/procedures that can be overloaded with "..." ?

cheers,

paulv

Subject: Re: IDL 8.0 compile_opt changes
Posted by JDS on Fri, 08 Jan 2010 21:42:38 GMT
View Forum Message <> Reply to Message

On Jan 6, 1:35 pm, pp <pp.pente...@gmail.com> wrote:
> I realised now that I had not thought this through:
>
> On Jan 5, 9:54 pm, pp <pp.pente...@gmail.com> wrote:
>
>>> 1. Overloading structure dereference and method invocation breaks the
>>> ability to semantically parse a.b.
>> 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.
>
> This is not actually a problem. It would just be the same kind of
> ambiguity that happens in the middle of typing a name, when there are
> more than one that begin with that same characters. At that point,
> using completion should give the user both options. In the case of the
> function/member ambiguity, the options would be "a.b(" and "a.b", as
> the first is the function, and the second is the structure member.

If you are the IDL interpreter, that's fine, because you know `a' is
either an object or a structure and can act accordingly. If you can
only presume that `a' may be some unknown structure, or some object,
the space of completions just ballooned by a factor of 2 at minimum.
If IDL had adopted the overloaded "." operator initially, the saved
keystroke of this syntactic might be worth the added ambiguity. Given
that it didn't, the pain associated with switching horses mid-streams
makes it a questionable move, in my view.

JD