Subject: Still missing features in IDL 8
Posted by penteado on Tue, 12 Oct 2010 21:35:02 GMT
View Forum Message <> Reply to Message

I was showing IDL's lists and hashes to a friend, who immediately
noticed two important missing points:

1) A new method (or a new keyword, like, /pointer, to the existing
toarray()), that would create from an arbitrary list an array of
pointers, where each element in the array is a pointer to the
corresponding element of the list. That is, something equivalent to

```
function topointerarray,list
ret=ptrarr(n_elements(list))
foreach el,list,i do ret[i]=ptr_new(el)
return,ret
end
```

And a new keyword to list() and list::add (such as /dereference) that
would do the opposite: given a pointer array, would add to the list
the variables pointed to by each element of the pointer array (or !
null in case the element is a null pointer or a pointer to an
undefined variable).

Also, in some cases it would be desirable for list::toarray() and
hash::tostruct() to have a no_copy keyword, that would empty the list/
hash when making the array/structure, instead of making a copy. This
is already present in list() and list::add, but is also missing in
hash().

Why is (1) so needed? Besides better handling those conversions
(particularly when interfacing with old code that returns/expects
pointer arrays), for getting around the lack of:

2) This is not a simple additional feature as (1), but rather a change
to the language: Make a way for a list/hash element not be an
expression. One example of the problem is this error:

```
IDL> a=list()
IDL> a.add,bindgen(3)
IDL> a.add,-dindgen(2)
IDL> print,a
   0   1   2
      -0.0000000      -1.0000000
IDL> print,(a[1])[0]
      -0.0000000
IDL> (a[1])[0]=1.0
% Expression must be named variable in this context: <DOUBLE
```

```
Array[2]>.
% Execution halted at: $MAIN$
```

This error happend because (a[1]) is an expression:

```
IDL> help,(a[1])
<Expression>    DOUBLE    = Array[2]
```

And I can see why it is, as I guess it is the result of a call to list::overloadbracketsrightside(). But this makes it very inconvenient to change pieces of list/hash elements, like in the line that caused the error above. It would be necessary to retrieve the element, assigning it to some variable, change that variable, then put it back in the list/hash. Very awkward, and unnecessary if a pointer array is used instead of the list:

```
IDL> a=ptrarr(2)
IDL> a[0]=ptr_new(bindgen(3))
IDL> a[1]=ptr_new(-dindgen(2))
IDL> print,(*a[1])[0]
      -0.0000000
IDL> (*a[1])[0]=1.0
IDL> print,(*a[1])[0]
       1.0000000
```

Which obviously works because (*a[1]) is a variable:

```
IDL> help,(*a[1])
<PtrHeapVar2>   DOUBLE    = Array[2]
```

I realize that this is a change to the language, and somewhat tricky to implement. But it is probably easier than was the addition of operator overloading.

Subject: Re: Still missing features in IDL 8
Posted by chris_torrence@NOSPAM on Mon, 01 Nov 2010 15:30:07 GMT

View Forum Message <> Reply to Message

On Oct 31, 6:00 pm, Paulo Penteado <pp.pente...@gmail.com> wrote:
> On Oct 13, 2:38 pm, Chris Torrence <gorth...@gmail.com> wrote:
>
>
>
>> Regarding #2, what if you could use additional indices to access array
>> elements within lists?
>
>> For example:

>
>> IDL> a = LIST(FINDGEN(10), BYTARR(5,3))
>> IDL> help, a[0]
>> <Expression>   FLOAT   = Array[10]
>> IDL> help, a[0,3]   ; currently throws an error in IDL8.0
>> <Expression>   FLOAT   =   3.00000
>> IDL> a[0,3] = !pi   ; currently throws an error in IDL8.0
>
>> IDL> help, a[1]
>> <Expression>   BYTE   = Array[5, 3]
>> IDL> help, a[1,4,2]   ; currently throws an error in IDL8.0
>> <Expression>   BYTE   =   0
>> IDL> a[1,4,2] = 255   ; currently throws an error in IDL8.0
>
>> So the first index would give the list element, and the remaining
>> indices would index into the array itself. Obviously you could only
>> have up to 7 dimensions in your contained array, but that probably
>> isn't a huge limitation.
>
> I was writing a class like that, inheriting from list, and that
> brought me a question: Should the extra dimension (of the list index)
> be on the left, as above, or on the right?
>
> The notation (already valid for retrieving values) (a[1])[0] suggests
> that the array index should come on the left. However, writing a[1,0]
> suggests array dimensions, in which case the list index would make
> more sense on the right, as the list dimension is the slowest-varying
> one.
>
> Tough it would be a bit incoherent with the array dimension order, it
> seems to me that it is better to have the list index on the left. That
> way,
>
> print,(a[1])[0] ;already valid
>
> would be the same as
>
> print,a[1,0]
>
> instead of the more confusing
>
> print,a[0,1]
>
> Any thoughts on that?

Yes, that is exactly what I was thinking.

Back to your original thread - if we added this way of subscripting,

does that eliminate the need to convert a list to/from a pointer
array? I'd rather not add more functionality if we don't have to.

-Chris
ITTVIS

Subject: Re: Still missing features in IDL 8
Posted by penteado on Mon, 01 Nov 2010 17:17:52 GMT
View Forum Message <> Reply to Message

On Nov 1, 1:30 pm, Chris Torrence <gorth...@gmail.com> wrote:
> Yes, that is exactly what I was thinking.
>
> Back to your original thread - if we added this way of subscripting,
> does that eliminate the need to convert a list to/from a pointer
> array? I'd rather not add more functionality if we don't have to.

It would decrease the need, but not eliminate it. One use that I can
remember now is to handle arrays to/from code that uses pointers,
which before 8.0 were needed in a lot more situations. I am already
adding those conversions to the class I am writing to provide the new
overloadbrackets methods. I will post it here when it gets ready.

Subject: Re: Still missing features in IDL 8
Posted by penteado on Wed, 03 Nov 2010 00:05:57 GMT
View Forum Message <> Reply to Message

On Nov 1, 3:17 pm, Paulo Penteado <pp.pente...@gmail.com> wrote:
> On Nov 1, 1:30 pm, Chris Torrence <gorth...@gmail.com> wrote:
>
>> Yes, that is exactly what I was thinking.
>
>> Back to your original thread - if we added this way of subscripting,
>> does that eliminate the need to convert a list to/from a pointer
>> array? I'd rather not add more functionality if we don't have to.
>
> It would decrease the need, but not eliminate it. One use that I can
> remember now is to handle arrays to/from code that uses pointers,
> which before 8.0 were needed in a lot more situations. I am already
> adding those conversions to the class I am writing to provide the new
> overloadbrackets methods. I will post it here when it gets ready.

There is also the independent need of a no_copy keyword for
hash::tostruct(), list::toarray(), and hash::init(). For hash::init()
one can get around the lack of no_copy with temporary(). But for

toarray and tostruct, there is no way to use them without making a
copy.

Subject: Re: Still missing features in IDL 8
Posted by Bob[4] on Thu, 04 Nov 2010 18:29:37 GMT
View Forum Message <> Reply to Message

On Nov 1, 9:30 am, Chris Torrence <gorth...@gmail.com> wrote:
```
>  On Oct 31, 6:00 pm, Paulo Penteado <pp.pente...@gmail.com> wrote:
>
>
>
>>  On Oct 13, 2:38 pm, Chris Torrence <gorth...@gmail.com> wrote:
>
>>>  Regarding #2, what if you could use additional indices to access array
>>>  elements within lists?
>
>>>  For example:
>
>>>  IDL> a = LIST(FINDGEN(10), BYTARR(5,3))
>>>  IDL> help, a[0]
>>>  <Expression>    FLOAT     = Array[10]
>>>  IDL> help, a[0,3]   ; currently throws an error in IDL8.0
>>>  <Expression>    FLOAT     =       3.00000
>>>  IDL> a[0,3] = !pi   ; currently throws an error in IDL8.0
>
>>>  IDL> help, a[1]
>>>  <Expression>    BYTE      = Array[5, 3]
>>>  IDL> help, a[1,4,2]   ; currently throws an error in IDL8.0
>>>  <Expression>    BYTE      =    0
>>>  IDL> a[1,4,2] = 255   ; currently throws an error in IDL8.0
>
>>>  So the first index would give the list element, and the remaining
>>>  indices would index into the array itself. Obviously you could only
>>>  have up to 7 dimensions in your contained array, but that probably
>>>  isn't a huge limitation.
>
>>  I was writing a class like that, inheriting from list, and that
>>  brought me a question: Should the extra dimension (of the list index)
>>  be on the left, as above, or on the right?
>
>>  The notation (already valid for retrieving values) (a[1])[0] suggests
>>  that the array index should come on the left. However, writing a[1,0]
>>  suggests array dimensions, in which case the list index would make
>>  more sense on the right, as the list dimension is the slowest-varying
>>  one.
>
```

>> Tough it would be a bit incoherent with the array dimension order, it
>> seems to me that it is better to have the list index on the left. That
>> way,
>
>> print,(a[1])[0] ;already valid
>
>> would be the same as
>
>> print,a[1,0]
>
>> instead of the more confusing
>
>> print,a[0,1]
>
>> Any thoughts on that?
>
> Yes, that is exactly what I was thinking.
>
> Back to your original thread - if we added this way of subscripting,
> does that eliminate the need to convert a list to/from a pointer
> array? I'd rather not add more functionality if we don't have to.
>
> -Chris
> ITTVIS

It seems to me that adding array subscripting to the list object is
wrong since lists can take many more types of objects than arrays.
This seems to be a product of IDL think that "everything is an array",
which does not make sense anymore in 8.0. I think the proper way to
implement the feature Paulo is asking for is to have an array_list
object, which would only take arrays and could have overloaded
brackets to get the extra subscripting. I think this is what Paulo is
implementing already and it would seem it could be completely
implemented with no changes to the language. Perhaps if you wanted to
implement it in the language, an ARRAY keyword could be added to the
list::init to force the list to only accept arrays and only if that is
set then the bracket overloading could be added. It still seems
cleaner to have a subclass object, however.

Bob

Subject: Re: Still missing features in IDL 8
Posted by penteado on Thu, 04 Nov 2010 22:28:49 GMT
View Forum Message <> Reply to Message

On Nov 4, 4:29 pm, Bob <bobnnamt...@gmail.com> wrote:
> It seems to me that adding array subscripting to the list object is

> wrong since lists can take many more types of objects than arrays.
> This seems to be a product of IDL think that "everything is an array",
> which does not make sense anymore in 8.0. I think the proper way to
> implement the feature Paulo is asking for is to have an array_list
> object, which would only take arrays and could have overloaded
> brackets to get the extra subscripting. I think this is what Paulo is
> implementing already and it would seem it could be completely
> implemented with no changes to the language. Perhaps if you wanted to
> implement it in the language, an ARRAY keyword could be added to the
> list::init to force the list to only accept arrays and only if that is
> set then the bracket overloading could be added. It still seems
> cleaner to have a subclass object, however.

I disagree. I find bracket overloading to be one of the best points in IDL 8, and that brackets are proper for anything that may make use of them: arrays, lists, hashes, and whatever other classes, like those used by Graphics.

For that reason one important consideration in the class I am writing is to make sure that the other subscripts (those that come after the subscript for the list) will be passed on to whatever is in the list make use of them. I was initially thinking about only arrays in lists because that was the first need I encountered. But I realized that other subscriptable things should make use of that too. When that list class gets done and tested, I will probably extend that to a hash class, using some intermediate class to be inherited, which could be used for any other class that wanted that kind of forwarding for extra indices.

It is up to the user to know in what way the things being accessed can be subscripted, in the same way that the user must know, for instance, not to use more dimensions than an array has when subscripting it.

Subject: Re: Still missing features in IDL 8
Posted by JDS on Thu, 04 Nov 2010 23:29:58 GMT
View Forum Message <> Reply to Message

Lists and hashes, while very welcome (!), are somewhat cumbersome to use with out some syntactic sugar along the lines of what has been discussed. In addition to:

```
 list[1,2,3]
```

for a list/array containing another list/array, containing another list/array, we need, e.g.:

```
 list[3,'foo',2:5]
```

for a list containing a hash or structure, containing an array or
list.   Currently you must use

 (list[3])['foo']

A better syntax, if it could be arranged, would be:

 list[1][2]

or

 list[3]['foo']

Even better if these could function correctly on the LHS of an
assignment.

I'm actually surprised the analogy of hashes with structures wasn't
expressed directly, ala:

 list[3].foo

... but that of course brings in the new (and in my opinion
unnecessary and in this case, deleterious) conflation of "." with "-
> ".

JD

Subject: Re: Still missing features in IDL 8
Posted by penteado on Thu, 04 Nov 2010 23:40:38 GMT
View Forum Message <> Reply to Message

On Nov 4, 9:29 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
> Lists and hashes, while very welcome (!), are somewhat cumbersome to
> use with out some syntactic sugar along the lines of what has been
> discussed. In addition to:
>
>   list[1,2,3]
>
> for a list/array containing another list/array, containing another
> list/array, we need, e.g.:
>
>   list[3,'foo',2:5]
>
> for a list containing a hash or structure, containing an array or
> list.

That is just what I was doing.

>   Currently you must use
>
> (list[3])['foo']
>
> A better syntax, if it could be arranged, would be:
>
> list[1][2]
>
> or
>
> list[3]['foo']
>
> Even better if these could function correctly on the LHS of an
> assignment.

I agree that way would be better, not just because it would not mess
with the 8D limits. But that would require changes to the language,
for the new syntax, and a very different way for the overloadbrackets
method to work.

Subject: Re: Still missing features in IDL 8
Posted by JDS on Sat, 06 Nov 2010 19:03:31 GMT
View Forum Message <> Reply to Message

On Nov 4, 5:40 pm, Paulo Penteado <pp.pente...@gmail.com> wrote:
> On Nov 4, 9:29 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
>
>> Lists and hashes, while very welcome (!), are somewhat cumbersome to
>> use with out some syntactic sugar along the lines of what has been
>> discussed. In addition to:
>
>> list[1,2,3]
>
>> for a list/array containing another list/array, containing another
>> list/array, we need, e.g.:
>
>> list[3,'foo',2:5]
>
>> for a list containing a hash or structure, containing an array or
>> list.
>
> That is just what I was doing.
>
>>   Currently you must use
>

>> (list[3])['foo']
>
>> A better syntax, if it could be arranged, would be:
>
>> list[1][2]
>
>> or
>
>> list[3]['foo']
>
>> Even better if these could function correctly on the LHS of an
>> assignment.
>
> I agree that way would be better, not just because it would not mess
> with the 8D limits. But that would require changes to the language,
> for the new syntax, and a very different way for the overloadbrackets
> method to work.

I don't see how 8D limits come in. The limits are the number of
possible arguments to the _OVERLOADBRACKETSLEFTSIDE method, which is
much larger than 8. Unfortunately, the proposed method is deeply
flawed, as there is a unavoidable ambiguity between array and list
indices. Consider:

```
IDL> la=objarr(2,2)
IDL> for i=1b,4 do la[i-1]=list([i,2*i^2],100b+[i,2*i^2])
IDL> for i=0,3 do print,la[i]
    1      2
  101    102
    2      8
  102    108
    3     18
  103    118
    4     32
  104    132
```

Now consider the proposed syntax: la[0,1,1]

Which indices apply to which entity? I.e., is this:

```
IDL> print,(la[0,1])[1]
  103    118
```

or

```
IDL> print,((la[0])[1])[1]
  102
```

One option would be to say that arrays of lists gobble up as many
indices as they have dimensions, but that is unnecessarily limiting
and unintuitive, since you might like to index them using a single
scalar as I have. You might also say each list gobbles only only a
single index.

This also point out how the other syntax is superior:

 la[0,1][1] vs. la[0][1][1]

would be different things. But I agree, this is not easy to
accommodate at this point.

A few more comments on LIST and HASH:

We need LISTARR and HASHARR to go with OBJARR.

You should be able to assign a single scalar variable to "hash slices"
to set each of the specified keys to that same value, just as you can
assign a single value to a subset of lists:

```
IDL> a=hash({one:1,two:2})
IDL> a[ ['one','two'] ] = 4 ;; this should work!
% Key and Value must have the same number of elements.
% Error occurred at: HASH::_OVERLOADBRACKETSLEFTSIDE
%                    $MAIN$
% Execution halted at: $MAIN$
```

JD

Subject: Re: Still missing features in IDL 8
Posted by penteado on Sat, 06 Nov 2010 21:51:23 GMT

View Forum Message <> Reply to Message

On Nov 6, 5:03 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
>> I agree that way would be better, not just because it would not mess
>> with the 8D limits. But that would require changes to the language,
>> for the new syntax, and a very different way for the overloadbrackets
>> method to work.
>
> I don't see how 8D limits come in. The limits are the number of
> possible arguments to the _OVERLOADBRACKETSLEFTSIDE method, which is
> much larger than 8.

The methods can take more arguments, but the parser currently only

allows 8 dimensions in the brackets. Which is why I said there are
currently two 8D limits: one for arrays, which is probably hard to
change, and one for the overloaded brackets, which can probably
increase easily (though at the cost of throwing errors if, say, 9D
were used on a class that had its methods written for only up to 8).

```
>  Unfortunately, the proposed method is deeply
> flawed, as there is a unavoidable ambiguity between array and list
> indices.  Consider:
>
> IDL> la=objarr(2,2)
> IDL> for i=1b,4 do la[i-1]=list([i,2*i^2],100b+[i,2*i^2])
> IDL> for i=0,3 do print,la[i]
>        1       2
>      101     102
>        2       8
>      102     108
>        3      18
>      103     118
>        4      32
>      104     132
>
> Now consider the proposed syntax: la[0,1,1]
>
> Which indices apply to which entity?  I.e., is this:
>
> IDL> print,(la[0,1])[1]
>      103     118
>
> or
>
> IDL> print,((la[0])[1])[1]
>      102
```

As IDL is now, la[0,1,1] cannot work. la is an array, not an object,
so it will not take 3D indices as it is 2D. An error will be thrown,
and no overload method will even be called. So parenthesis have to be
used to select an individual element of la, which would be an object,
and there is no problem.

```
>
> This also point out how the other syntax is superior:
>
>  la[0,1][1]  vs. la[0][1][1]
>
> would be different things.  But I agree, this is not easy to
> accommodate at this point.
```

Yes, that would be better, but cannot be implemented without changing the language, parsers and interpreters, and only ITTVIS could do that.

```
>
>  A few more comments on LIST and HASH:
>
>  We need LISTARR and HASHARR to go with OBJARR.
>
>  You should be able to assign a single scalar variable to "hash slices"
>  to set each of the specified keys to that same value, just as you can
>  assign a single value to a subset of lists:
>
>  IDL> a=hash({one:1,two:2})
>  IDL> a[ ['one','two'] ] = 4 ;; this should work!
>  % Key and Value must have the same number of elements.
>  % Error occurred at: HASH::_OVERLOADBRACKETSLEFTSIDE
>  %                    $MAIN$
>  % Execution halted at: $MAIN$
```

Yes, doing that should be possible (and I was surprised when I first noticed it is not), but no new data types (which is what a new array type would require) are needed. Only a simple change to the _overloadbracketsleftside methods.

Maybe I will add that to the methods I am writing. It would be implemented by either a foreach or a replicate. Probably a replicate, meaning a user would have to directly use a foreach instead when copying would be a problem, which seems far less likely to be the case: if the array of copies is big enough to be a problem, putting it into a list is probably a bigger, independent problem anyway, with the list overhead.

Subject: Re: Still missing features in IDL 8
Posted by penteado on Wed, 10 Nov 2010 01:41:59 GMT
View Forum Message <> Reply to Message

I have also been considering how to add a deep copy method to my class. In the lack of a standard name for a deepcopy method that could be recursively searched for, I was thinking of searching for _overloadbracketsrightside, and when finding one, calling it with [*], which would be expected to give the first level copy (as list and hash do).