Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by m_schellens on Mon, 29 Jul 2013 08:12:46 GMT
View Forum Message <> Reply to Message

Am Sonntag, 28. Juli 2013 06:13:18 UTC+2 schrieb bobnn...@gmail.com:
> On Friday, July 26, 2013 11:24:01 AM UTC-6, mschellens wrote:
>
>> Am Freitag, 23. März 2012 00:24:08 UTC+1 schrieb pp.pe...@gmail.com:
>
>>
>
>>> It is not an intrinsic limitation of their objectyness. It is all about how the overloadbracketsleftside method is defined. Once someone pointed out that problem to me (in the case of arrays), I had several discussions with the developers (some in this newsgroup), and eventually the methods were changed (in 8.1, I think) so that through multiple indices it can be done with arrays:
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> IDL> h=hash('a',[0,3,9])
>
>>
>
>>>
>
>>
>
>>> IDL> print,h['a']
>
>>
>
>>>
>
>>
>
>>> 0 3 9

>
>>
>
>>>
>
>>
>
>>> IDL> print,h['a',1]
>
>>
>
>>>
>
>>
>
>>>   3
>
>>
>
>>>
>
>>
>
>>> IDL> h['a',1]=-1
>
>>
>
>>>
>
>>
>
>>> IDL> print,h['a',1]
>
>>
>
>>>
>
>>
>
>>>   -1
>
>>
>
>>>
>
>>
>
>>>

>
>>
>
>>>
>
>>
>
>>> The problems shows up in structures and array indices if they are used in the way written above:
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> IDL> print,(h['a'])[1]
>
>>
>
>>>
>
>>
>
>>>   -1
>
>>
>
>>>
>
>>
>
>>> IDL> (h['a'])[1]=99
>
>>
>
>>>
>
>>
>

>>> % Expression must be named variable in this context: <INT   Array[3]>.
>
>>
>
>>>
>
>>
>
>>> % Execution halted at: $MAIN$
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> And that is all fault of the parser's weird use of values instead of references on qualified names. Which, for that reason, and for breaking the least surprise principle, should be changed in a future version (a compile_opt, or a new file extension, to keep compatibility).
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> But I think that the multiple indices for arrays approach could easily be applied to structures, just making a small change to the overloadbracketsleftside method. I may write a small derived class for that purpose. In that way, it would be possible to do
>
>>
>

>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> h['a','b']=2
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> On Thursday, March 22, 2012 6:58:19 PM UTC-3, JDS wrote:
>
>>
>
>>>
>
>>
>
>>>> HASH's are nice in that they can contain any variable type, and make iterating over deeply nested data structure more humane. On the other hand, their limitations as separate standalone objects, and not as a deeper feature of the language, are very apparent, for example, when you'd like to alter something inside of them:

>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>> IDL> h=hash('a',{b:1,c:2})
>
>>
>
>>>
>
>>
>
>>>> IDL> print,h['a'].b
>
>>
>
>>>
>
>>
>
>>>>   1
>
>>
>
>>>
>
>>
>
>>>> IDL> h['a'].b=2
>
>>
>
>>>
>
>>
>
>>>> % Attempt to store into an expression: Structure reference.

>
>>
>
>>>
>
>>
>
>>>> % Execution halted at: $MAIN$ 84
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>> The same happens for arrays stored in hashes. You're required to copy the entire array or structure out, modify it, then copy it back into the hash (remind anyone of the old days and widget state?). If it were possible for HASH dereferencing to return true IDL variable references, they would be far more useful.
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>> Any workarounds to this "write-only" HASH behavior (other than, say, "use a pointer")?
>
>>
>
>>>
>

>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>> JD
>
>>
>
>>
>
>>
>
>>
>
>>
>
>> I revive this thread now, because it came to my interest only recently (guess why).
>
>>
>
>>
>
>>
>
>> I think I found a possibility to store in-place in IDL objects, which is fully compatible to to-date code. And it looks and feels like it should be:
>
>>
>
>>
>
>>
>
>> e. g. (not working now)
>
>>
>
>>
>
>>
>
>> IDL> h=hash('a',{b:intarr(5),c:2})

>
>>
>
>> IDL> h['a'].b[1:3] = [2,2,7]
>
>>
>
>>
>
>>
>
>> Just signal to the OBJECT::_OVERLOADBRACKETSLEFTSIDE the struct access by setting OBJREF to a variable with a value of !NULL (must be done automatically by the IDL environment. Note: OBJREF is usually set to an instance of the SELF object.)
>
>>
>
>> OBJECT::_OVERLOADBRACKETSLEFTSIDE then sets OBJREF to a PTR to the element accessed. The IDL environment must then left-struct-access the heap variable.
>
>>
>
>> This only works for
>
>>
>
>> a) heap variables,
>
>>
>
>> which is fine as all HASH and LIST elements are heap variables.
>
>>
>
>> b) Single elements (it could be extended easily by returning a PTR array).
>
>>
>
> Maybe you could post some code, as I cannot follow your description.
>
>
>
> I will say that the inability of accessing directly into hashes and lists (by reference, not copy) is so disappointing that I am moving away from IDL. So if you found a way around this it would be very nice.
>
>
>

>>
>
>>
>
>> This can be done even with any (IDL_OBJECT derived) object as no internal tricks are needed in _OVERLOADBRACKETSLEFTSIDE.
>
>>
>
>>
>
>>
>
>> (Of course this is also the way it is done in GDL :-)
>
>>

I should have been more expilcit:
Currently, you cannot use my suggestion, as it requires some changes within IDL.

I wanted to point out, that it is possible to extend IDL in a 'natural' (and backwards-compatible) way.
EXELIS should incorporate this in the next IDL version (or ASAP). Because currently this is a clear lack of IDL.

And in GDL the feasibility is demonstrated.

Marc
