
Subject: Re: Is a dynamically sized pointer array object component possible?

Posted by [Michael Galloy](#) on Thu, 21 May 2009 22:21:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

Paul van Delst wrote:

```
> Hello,
>
> I'm trying to create an object with a component that is a pointer
> array... but I don't know the size of the array ahead of time. In
> structure-speak, I'm doing this:
>
> IDL> x={blah,y:ptr_new()}
> IDL> x.y=ptr_new(ptrarr(4))
> IDL> (*x.y)[0] = ptr_new(dindgen(20))
> IDL> help, *(*x.y)[0]
> <PtrHeapVar1>  DOUBLE   = Array[20]
>
> Now, if I knew what size of array I needed in advance (in this example,
> 4), I could do the following:
>
> IDL> x={blah,y:ptrarr(4)}
> IDL> x.y[0] = ptr_new(dindgen(20))
> IDL> help, *x.y[0]
> <PtrHeapVar1>  DOUBLE   = Array[20]
>
> The latter example is preferable since
> a) it more closely reflect the data, and
> b) the dereferencing is clearer.
>
> I tried to do this:
>
> IDL> x={blah,y:ptr_new()}
> IDL> x.y = ptrarr(4)
> % Expression must be a scalar in this context: <POINTER  Array[4]>.
> % Execution halted at: $MAIN$
>
> I (mostly) knew it wouldn't work, but is there a way to do this? Having
> a pointer to a pointer array I find..... disconcerting.
>
> In the final application I would have the following procedure,
>
> PRO blah__define
>   void = { blah, y:some_fancy_definition?.... }
> END
>
> and then do something like,
>
> PRO blah::allocate, n
```

```

> self.y = PTRARR(N_ELEMENTS(n)) ; This causes the heartache.
> FOR i = 0, N_ELEMENTS(n)-1 DO BEGIN
>   self.y[i] = PTR_NEW(DBLARR(n[i]))
> ENDFOR
> END
>
> to be called thusly:
>
> x = obj_new('blah')
> x->allocate([2,5,9,25])
>
> Is it doable? Am I missing another simple fix (ala the
> FORMAT_AXIS_VALUES function from a previous thread :o) I would like to
> avoid the double dereferencing if possible.
>
> Hopefully I've explained myself. Thanks for making it this far.
>
> cheers,
>
> paulv

```

How about creating a pointer to a pointer array?

```

pro blah::allocate, n
  compile_opt strictarr

  *self.y = ptrarr(n_elements(n))
  for i = 0, n_elements(n) - 1L do begin
    (*self.y)[i] = ptr_new(dblarr(n[i]))
  endfor
end

```

```

function blah::get, m, n
  compile_opt strictarr

  return, ((*self.y)[m])[n]
end

```

```

pro blah::set, m, n, value
  compile_opt strictarr

  ((*self.y)[m])[n] = value
end

```

```

function blah::init
  compile_opt strictarr

```

```
self.y = ptr_new(/allocate_heap)
return, 1
end
```

```
pro blah__define
void = { blah, y: ptr_new() }
end
```

This could be used like:

```
IDL> blah = obj_new('blah')
IDL> blah->allocate, [2, 5, 9, 25]
IDL> blah->set, 1, 0, findgen(5)
IDL> print, blah->get(1, 2)
      4.0000000
```

Mike

--

www.michaelgalloy.com
Associate Research Scientist
Tech-X Corporation

Subject: Re: Is a dynamically sized pointer array object component possible?
Posted by [Paul Van Delst\[1\]](#) on Fri, 22 May 2009 13:30:53 GMT
[View Forum Message](#) <> [Reply to Message](#)

mgalloy wrote:

```
> Paul van Delst wrote:
>> Hello,
>>
>> I'm trying to create an object with a component that is a pointer
>> array... but I don't know the size of the array ahead of time. In
>> structure-speak, I'm doing this:
>>
>> IDL> x={blah,y:ptr_new()}
>> IDL> x.y=ptr_new(ptrarr(4))
>> IDL> (*x.y)[0] = ptr_new(dindgen(20))
>> IDL> help, *(*x.y)[0]
>> <PtrHeapVar1>  DOUBLE   = Array[20]
>>
>> Now, if I knew what size of array I needed in advance (in this
>> example, 4), I could do the following:
>>
>> IDL> x={blah,y:ptrarr(4)}
>> IDL> x.y[0] = ptr_new(dindgen(20))
>> IDL> help, *x.y[0]
>> <PtrHeapVar1>  DOUBLE   = Array[20]
```

```

>>
>> The latter example is preferable since
>> a) it more closely reflect the data, and
>> b) the dereferencing is clearer.
>>
>> I tried to do this:
>>
>> IDL> x={blah,y:ptr_new()}
>> IDL> x.y = ptrarr(4)
>> % Expression must be a scalar in this context: <POINTER  Array[4]>.
>> % Execution halted at: $MAIN$
>>
>> I (mostly) knew it wouldn't work, but is there a way to do this?
>> Having a pointer to a pointer array I find..... disconcerting.
>>
>> In the final application I would have the following procedure,
>>
>> PRO blah__define
>>   void = { blah, y:some_fancy_definition?.... }
>> END
>>
>> and then do something like,
>>
>> PRO blah::allocate, n
>>   self.y = PTRARR(N_ELEMENTS(n)) ; This causes the heartache.
>>   FOR i = 0, N_ELEMENTS(n)-1 DO BEGIN
>>     self.y[i] = PTR_NEW(DBLARR(n[i]))
>>   ENDFOR
>> END
>>
>> to be called thusly:
>>
>> x = obj_new('blah')
>> x->allocate([2,5,9,25])
>>
>> Is it doable? Am I missing another simple fix (ala the
>> FORMAT_AXIS_VALUES function from a previous thread :o) I would like
>> to avoid the double dereferencing if possible.
>>
>> Hopefully I've explained myself. Thanks for making it this far.
>>
>> cheers,
>>
>> paulv
>
> How about creating a pointer to a pointer array?

```

Hi Mike,

That's what my first example above does (it was the only way I could make it work). I was trying to avoid that if possible to avoid the double dereferencing that would require in the object methods - as in your "get" and "set" method:

```
> function blah::get, m, n
>   compile_opt strictarr
>
>   return, (*(*self.y)[m])[n]
> end
>
> pro blah::set, m, n, value
>   compile_opt strictarr
>
>   (*(*self.y)[m])[n] = value
> end
```

And that is what I am doing now in my code. For example, my "set" method does

```
*(self.Frequency)[_Band] = Frequency
*(self.Response)[_Band] = Response
```

(where Frequency and Response are vectors.[*])

I just wanted to avoid the `*(self.y)` double dereference (DD) if possible. It has zero impact on the user, of course - I want to avoid the DDing for my own benefit (insert sheepish grin here)

Thanks for taking the time to write the code. It's a nice teaching example.

cheers,

paulv

[*] BTW, note also my use of "_Band". I have now adopted your methodology for things like,

```
; Check band keyword argument
IF ( N_ELEMENTS(Band) GT 0 ) THEN _Band = LONG(Band[0])-1 ELSE _Band = 0L
```

based on your post a few days ago. I've noticed that these type of small, incremental changes to create more robust code (like the snippet above) eventually leads to shifts in other people's perceptions about writing clean code (e.g. no side effects). Nothing earth shattering in this little post scriptum, of course, but still neat.

Subject: Re: Is a dynamically sized pointer array object component possible?

Paul van Delst wrote:

```
> That's what my first example above does (it was the only way I could
> make it work). I was trying to avoid that if possible to avoid the
> double dereferencing that would require in the object methods - as in
> your "get" and "set" method:
>
>> function blah::get, m, n
>>   compile_opt strictarr
>>
>>   return, ((*self.y)[m])[n]
>> end
>>
>> pro blah::set, m, n, value
>>   compile_opt strictarr
>>
>>   ((*self.y)[m])[n] = value
>> end
>
> And that is what I am doing now in my code. For example, my "set" method
> does
>
>   ((*self.Frequency)[_Band] = Frequency
>   ((*self.Response)[_Band] = Response
>
> (where Frequency and Response are vectors.[*])
>
> I just wanted to avoid the (*.self.y) double dereference (DD) if
> possible. It has zero impact on the user, of course - I want to avoid
> the DDing for my own benefit (insert sheepish grin here)
>
> Thanks for taking the time to write the code. It's a nice teaching example.
```

Another way to do it would be for "::allocate, n" to just create a pointer to a single vector of size TOTAL(n) and also save the n array, then for "::get, i, j" to use TOTAL(n, /CUMULATIVE) to find the correct value(s). I'm not sure that would be simpler, but it would eliminate the double dereference.

```
> [*] BTW, note also my use of "_Band". I have now adopted your
> methodology for things like,
>
> ; Check band keyword argument
> IF ( N_ELEMENTS(Band) GT 0 ) THEN _Band = LONG(Band[0])-1 ELSE _Band = 0L
>
> based on your post a few days ago. I've noticed that these type of
> small, incremental changes to create more robust code (like the snippet
```

- > above) eventually leads to shifts in other people's perceptions about
- > writing clean code (e.g. no side effects). Nothing earth shattering in
- > this little post scriptum, of course, but still neato.

Cool!

Mike

--

www.michaelgalloy.com

Associate Research Scientist

Tech-X Corporation

Subject: Re: Is a dynamically sized pointer array object component possible?

Posted by [Paul Van Delst\[1\]](#) on Fri, 22 May 2009 17:05:40 GMT

[View Forum Message](#) <> [Reply to Message](#)

mgalloy wrote:

- >
- > Another way to do it would be for "::allocate, n" to just create a
- > pointer to a single vector of size TOTAL(n) and also save the n array,
- > then for "::get, i, j" to use TOTAL(n, /CUMULATIVE) to find the correct
- > value(s). I'm not sure that would be simpler, but it would eliminate the
- > double dereference.

Ah, yes. That is actually how I do it now and is the reason I am changing it. For some larger context, the problem is when we have satellite instrument channel responses that contain significant regions of zero response (e.g. like microwave instruments where the channel response can straddle atmospheric absorption lines). Lumping the response altogether exposes our integrator to numerical problems and, since we use the first moment of the response curves as the channel central frequencies, that can eventually bugger up our radiance calculations.

I'm going with the pointer to a pointer array approach and just deal with the double dereference. As someone paraphrased Voltaire in a meeting I was at recently: perfection is the enemy of good enough. :o)

cheers,

paulv
