Subject: How to get _overloadSize to return N_Dimensions=0?
Posted by Matthew Argall on Tue, 27 May 2014 22:40:41 GMT
View Forum Message <> Reply to Message

I was wondering how it is possible to get the _overloadSize function to return N_DIMENSION=0 when there are no dimensions. Below is an example. The docs seems like it reports N_Elements(Size(input, /DIMENSIONS), which is not correct for an undefined variable...

```
IDL> print, size(!Null, /N_DIMENSIONS)
           0
IDL> .run test_olsize__define
Dimensions:   [0]
N_Elements:   0
N_Dimensions: 1


;---------------------------------------------
function test_olSize::_OverloadSize
   return, size(*self.value, /N_DIMENSIONS)
end

function test_olSize::Init
   compile_opt strictarr
   self.value = Ptr_New(/ALLOCATE_HEAP)
   return, 1
end

pro test_olSize__define
   class = {test_olSize, $
            inherits IDL_Object, $
            value: ptr_new()}
end


;Main level test program
myObj = Obj_New('Test_olSize')
print, 'Dimensions:   ', '[' + strjoin(strtrim(size(myObj, /DIMENSIONS), 2), ', ') + ']'
print, 'N_Elements:   ', strtrim(size(myObj, /N_ELEMENTS), 2)
print, 'N_Dimensions: ', strtrim(size(myObj, /N_DIMENSIONS), 2)
obj_destroy, myObj

end
```

Subject: Re: How to get _overloadSize to return N_Dimensions=0?
Posted by chris_torrence@NOSPAM on Wed, 28 May 2014 03:14:07 GMT
View Forum Message <> Reply to Message

Looks like a bug to me. One side note - you want to return SIZE with the DIMENSIONS keyword, not N_DIMENSIONS:

```
function test_olSize::_OverloadSize
  return, size(*self.value, /DIMENSIONS)
end
```

Regardless, it still doesn't work properly... I'll fix it for IDL 8.3.1.

-Chris
ExelisVIS

Subject: Re: How to get _overloadSize to return N_Dimensions=0?
Posted by Matthew Argall on Tue, 18 Apr 2017 13:56:43 GMT
View Forum Message <> Reply to Message

The problem with undefined variables was fixed, but there is another bug related to scalar values. I assume N_Elements is determined as Product(Size(input, /DIMENSIONS)). However, a scalar value has a dimension size of 0, so the result of 0 elements.

```
function test_olSize::_OverloadSize
   return, size(*self.value, /DIMENSIONS)
end

function test_olSize::Init, value
   compile_opt strictarr
   self.value = Ptr_New(value, /NO_COPY)
   return, 1
end

pro test_olSize__define
   class = {test_olSize, $
            inherits IDL_Object, $
            value: ptr_new()}
end


IDL> myObj = Obj_New('Test_olSize', 1)
IDL> Print, N_Elements(myObj)
          0
```

Subject: Re: How to get _overloadSize to return N_Dimensions=0?
Posted by Michael Galloy on Wed, 19 Apr 2017 21:19:40 GMT
View Forum Message <> Reply to Message

On 4/18/17 7:56 AM, Matthew Argall wrote:
> The problem with undefined variables was fixed, but there is another
> bug related to scalar values. I assume N_Elements is determined as
> Product(Size(input, /DIMENSIONS)). However, a scalar value has a
> dimension size of 0, so the result of 0 elements.
>
>
> function test_olSize::_OverloadSize
>   return, size(*self.value, /DIMENSIONS)
> end
>
> function test_olSize::Init, value
>   compile_opt strictarr
>   self.value = Ptr_New(value, /NO_COPY)
>   return, 1
> end
>
> pro test_olSize__define
>   class = {test_olSize, $
>   inherits IDL_Object, $
>   value: ptr_new()}
> end
>
>
> IDL> myObj = Obj_New('Test_olSize', 1)
> IDL> Print, N_Elements(myObj)
>   0
>

I would use something like the following for your _overloadSize method:

```
function test_olsize::_overloadSize
  return, size(*self.value, /n_dimensions) eq 0 $
          ? 1L $
          : size(*self.value, /dimensions)
end

IDL> myObj = Obj_New('Test_olSize', 1)
IDL> Print, N_Elements(myObj)
          1
```

Mike

--
Michael Galloy
www.michaelgalloy.com
Modern IDL: A Guide to IDL Programming (http://modernidl.idldev.com)

Subject: Re: How to get _overloadSize to return N_Dimensions=0?
Posted by Matthew Argall on Fri, 21 Apr 2017 16:48:55 GMT
View Forum Message <> Reply to Message

The fix has to be a bit more complicated. Using your suggestion,

```
IDL> myobj = Test_olSize(!Null)
IDL> Print, N_Elements(myobj)
           1


function test_olSize::_OverloadSize
    return, n_elements(*self.value)         eq 0 ? 0L : $
            size(*self.value, /n_dimensions) eq 0 ? 1L : $
            size(*self.value, /dimensions)
end


IDL> myobj = Test_olSize(!Null)
IDL> Print, N_Elements(myobj)
           0
```

Consequently, this fixes the original problem as well.