Subject: Re: Arrays in structures; workarounds?
Posted by John-David T. Smith on Thu, 04 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
>
> "J.D. Smith" <jdsmith@astro.cornell.edu> writes:
>
>> The problem here is reliance on *trailing* shallow dimensions. The
>> IDL manual quotes us:
>>
>> "As with other subscript operations, trailing degenerate dimensions
>> (those with a size of 1) are eliminated."
>>
>> While I can't agree with IDL's mixed notion of a variable's dimensionality
>> between help and direct structure member access, I think IDL has always been
>> clear about this point.
>
> Yes, this aspect of IDL's behavior is documented. I've complained
> about this before because the policy has the habit of biting you at
> very awkward moments. As a programmer, you rarely expect the shapes
> of your arrays to change without your asking!
>
> But my point here was that there is absolutely *no* legitimate way to
> find the true dimensions of a structure tag. Hopefully you will see
> why I want this soon enough, when I have time to finish my little
> project. It's pretty cool.
>
>> The one place you are justified in complaining is the truncation of a single
>> element vector to a scalar: this is a bug (or at least an inconsistency), and
>> affects structure field access only:
>
> Bingo. IDL's behavior of trimming degnerate dimensions from variables
> is *usually* okay because it still leaves you with an array at the
> end. However, when single-element tags in structures are extracted
> they are (a) converted to scalars, and (b) there is no way to know
> that this happened. Try passing this as X or Y to a routine like PLOT
> and you get a crashola. Documented or not, this is ridiculous
> behavior.
>
> Multi-element arrays have a similar problem, but I posted a solution
> for that. I was hoping to close this final gap...

One man's ridiculous behavior is another's feature. I love being able to extract image planes and pass them on to display programs without explicitly chopping off unnecessary dimensions, which I'd bet is why they introduced it in the first place. Do I take it that you're trying to teleport the dimensions over the head of structure extraction and reinstate it post facto? Getting

pretty deperate for those dimensions! When I find it necessary to keep a record of the dimensionality of the object a given data subsection originally belonged to, I arrange my data cubes or hypercubes "on their sides". This obviously won't work for vectors.

We can agree that automatically converting a vector to a scalar is absolutely incorrect, especially since it only seems to happen on structure field extraction.

If you can stand it, use pointers in those structures:

```
IDL> a=[1]
IDL> b={a:ptr_new(a)}
IDL> print,size(*b.a)
           1           1           2           1
```

no dimension chopping there!

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: Arrays in structures; workarounds?
Posted by Craig Markwardt on Thu, 04 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"J.D. Smith" <jdsmith@astro.cornell.edu> writes:

> The problem here is reliance on *trailing* shallow dimensions. The
> IDL manual quotes us:
>
> "As with other subscript operations, trailing degenerate dimensions
> (those with a size of 1) are eliminated."
>
> While I can't agree with IDL's mixed notion of a variable's dimensionality
> between help and direct structure member access, I think IDL has always been
> clear about this point.

Yes, this aspect of IDL's behavior is documented. I've complained about this before because the policy has the habit of biting you at very awkward moments. As a programmer, you rarely expect the shapes of your arrays to change without your asking!

But my point here was that there is absolutely *no* legitimate way to find the true dimensions of a structure tag. Hopefully you will see why I want this soon enough, when I have time to finish my little project. It's pretty cool.

> The one place you are justified in complaining is the truncation of a single
> element vector to a scalar: this is a bug (or at least an inconsistency), and
> affects structure field access only:

Bingo. IDL's behavior of trimming degnerate dimensions from variables is *usually* okay because it still leaves you with an array at the end. However, when single-element tags in structures are extracted they are (a) converted to scalars, and (b) there is no way to know that this happened. Try passing this as X or Y to a routine like PLOT and you get a crashola. Documented or not, this is ridiculous behavior.

Multi-element arrays have a similar problem, but I posted a solution for that. I was hoping to close this final gap...

Oh well,
Craig

P.S. I should be writing a proposal now. Yikes. Back to work..
--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: Arrays in structures; workarounds?
Posted by John-David T. Smith on Thu, 04 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
>
> Ed Santiago <esm@lanl.gov> writes:
>
>>> Does anybody know a way to work around this? [...] The
>>> only thing I can come up with is to parse the result of HELP,
>>> OUTPUT=out, but that seems like the crappiest solution ever.
>>
>> Yep. Crappy indeed... but I couldn't find an alternative, either.
>>
>> Below is a copy of "esmsize", a function I wrote last year when I
>> *absolutely needed* to obtain true dimensions within a struct.
>>

>> Hope it helps,
> ...
>
> Yes, that's what I was afraid I might have to do. Unfortunately I'd
> still like it to work for IDL 4, for which HELP, OUTPUT doesn't work.
>
> Here was one trick I found to determine the size of a structure tag,
> if it has *at least* two elements. Try this:
>
> IDL> zz = {x:reform(dblarr(2,2,1),2,2,1)}
> IDL> help, zz([0,0]).x
> <Expression> DOUBLE = Array[2, 2, 1, 2]
>
> In this case, I index the structure with the [0,0] list while at the
> same time extracting the X tag. You get the correct dimensions, with
> an extra "2" tacked on the end, which you can then hack off. You need
> to handle the case of X being 8 dimensional (!), but it works.
> Unfortunately this *doesn't* work if X has only element. Arghh!
>
> A comment on your procedure. I believe that you are treating the
> output of HELP too simply. When tag names are long enough, help will
> wrap the type description to the next line. Consider this:
>
> IDL> zz = {sdlfkjsdlkfjsdklfjslkfjsldkfjsdlkfjsljfsldkfjsdf:1}
> IDL> help, /struct, zz
> ** Structure <40045788>, 1 tags, length=2, refs=1:
> SDLFKJSDLKFJSDKLFJSLKFJSLDKFJSDLKFJSLJFSLDKFJSDF
> INT = 1
>
> This will affect both the first tag, and any of the following ones.
> Maybe it's better to do a search for the tag you want.

The problem here is reliance on *trailing* shallow dimensions. The IDL manual quotes us:

"As with other subscript operations, trailing degenerate dimensions (those with a size of 1) are eliminated."

While I can't agree with IDL's mixed notion of a variable's dimensionality between help and direct structure member access, I think IDL has always been clear about this point.

e.g.

```
IDL> x=reform(dblarr(2,2,1),2,2,1)
IDL> print,size(x)
        3         2         2         1         5         4
```

```
IDL> y=x
IDL> print,size(y)
          2           2           2           5           4
```

Whenever it gets a chance to, IDL snips off those trailing dimensions, not just on structure access. You may disagree with the policy, which was designed to accomodate image subscripting without too much dimensional fussing, but it is documented. Living with this behavior when you'd like to preserve then number of dimensions usually involves ensuring that subscripting extraction happens on leading dimensions. Making use of this behavior usually involves ensuring it happens on the trailing dimensions.

The one place you are justified in complaining is the truncation of a single element vector to a scalar: this is a bug (or at least an inconsistency), and affects structure field access only:

```
IDL> a=[1]
IDL> print,size(a)
          1           1           2           1
IDL> b=a
IDL> print,size(b)
          1           1           2           1
IDL> b={a:a}
IDL> help,b,/st
** Structure <823abb4>, 1 tags, length=2, refs=1:
   A               INT       Array[1]
IDL> help,b.a
<Expression>    INT       =        1
```

The statement from the documentation quoted above is wrong. The is one time a final unit length dimension is not degenerate --- for a vector of length 1.

Good Luck,

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: Arrays in structures; workarounds?
Posted by Ed Santiago on Thu, 04 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

> Here was one trick I found to determine the size of a structure tag,

>if it has *at least* two elements.

Cool. Weird... but cool. Thanks; I've added it to my bag of
tricks (which already contains a heckuva lot of craigm code & ideas).

>A comment on your procedure. I believe that you are treating the
>output of HELP too simply. When tag names are long enough, help will
>wrap the type description to the next line. Consider this:

Guilty as charged. I briefly considered cleaning up the code so it
handles that case, but decided it wasn't worth the time investment.
The esmsize() function serves in a controlled environment where I
know the _name_ of the structure element I want ("DATA"), but not
the dimensions (incredibly complex -- but flexible! -- telemetry
mode sets for an instrument on Deep Space One).

So in the usual "write-code-when-I-need-it" manner, I left the
multi-line HELP handling as an exercise for the reader... and
never thought anyone else but me would even care.

Wishing I had the time to hack PDL to where it suits my needs,
^E
--
Eduardo Santiago Software Type esm@lanl.gov RKBA!

Subject: Re: Arrays in structures; workarounds?
Posted by Craig Markwardt on Thu, 04 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Ed Santiago <esm@lanl.gov> writes:

>> Does anybody know a way to work around this? [...] The
>> only thing I can come up with is to parse the result of HELP,
>> OUTPUT=out, but that seems like the crappiest solution ever.
>
> Yep. Crappy indeed... but I couldn't find an alternative, either.
>
> Below is a copy of "esmsize", a function I wrote last year when I
> *absolutely needed* to obtain true dimensions within a struct.
>
> Hope it helps,
...

Yes, that's what I was afraid I might have to do. Unfortunately I'd
still like it to work for IDL 4, for which HELP, OUTPUT doesn't work.

Here was one trick I found to determine the size of a structure tag,
if it has *at least* two elements. Try this:

```
IDL> zz = {x:reform(dblarr(2,2,1),2,2,1)}
IDL> help, zz([0,0]).x
<Expression>    DOUBLE    = Array[2, 2, 1, 2]
```

In this case, I index the structure with the [0,0] list while at the
same time extracting the X tag. You get the correct dimensions, with
an extra "2" tacked on the end, which you can then hack off. You need
to handle the case of X being 8 dimensional (!), but it works.
Unfortunately this *doesn't* work if X has only element. Arghh!

A comment on your procedure. I believe that you are treating the
output of HELP too simply. When tag names are long enough, help will
wrap the type description to the next line. Consider this:

```
IDL> zz = {sdlfkjsdlkfjsdklfjslkfjsldkfjsdlkfjsljfsldkfjsdf:1}
IDL> help, /struct, zz
** Structure <40045788>, 1 tags, length=2, refs=1:
   SDLFKJSDLKFJSDKLFJSLKFJSLDKFJSDLKFJSLJFSLDKFJSDF
               INT       =        1
```

This will affect both the first tag, and any of the following ones.
Maybe it's better to do a search for the tag you want.

Craig

--

 -- --------------
Craig B. Markwardt, Ph.D.   EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 -- --------------

Subject: Re: Arrays in structures; workarounds?
Posted by Ed Santiago on Thu, 04 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

> Does anybody know a way to work around this? [...] The
  >only thing I can come up with is to parse the result of HELP,
  >OUTPUT=out, but that seems like the crappiest solution ever.

Yep. Crappy indeed... but I couldn't find an alternative, either.

Below is a copy of "esmsize", a function I wrote last year when I
absolutely needed to obtain true dimensions within a struct.

Hope it helps,
^E

```
;+
; NAME:
;     ESMSIZE
;
; IDENT:
;     $Id: esmsize.pro,v 1.3 2000/01/31 14:38:23 esm Exp $
;
; PURPOSE:
;     Front end to SIZE, which will preserve unary dimensions
;
; AUTHOR:
;     Ed Santiago
;
; CALLING SEQUENCE:
;     xx = esmsize( struct, index )
;
; INPUTS:
;     struct      IDL structure
;     index       string or integer index into structure
;
; OUTPUTS:
;     IDL SIZE thingy
;
; REASON FOR THIS BULLSHIT:
;     IDL collapses unary dimensions wherever it can.  For example:
;
;       Pepe> trashme = { foo:reform(indgen(10),10,1) }
;       Pepe> help,trashme,/st
;       ** Structure <81cab34>, 1 tags, length=20, refs=1:
;          FOO             INT       Array[10, 1]
;       Pepe> print,size(trashme.foo)
;                 1          10           2          10
;
;     See?  There is simply no fscking way to get IDL to recognize that
;     last dimension, even though the HELP command sees it.  Therefore,
;     this code was written to parse the HELP output.  Barf city.
;
;       Pepe> print,esmsize(trashme,'foo')
;                 2          10           1           2          10
;-
FUNCTION esmsize, struct, index_orig

  On_Error, 2
```

```
  ; We'll never get here, since the StRegeg()'s will not compile
  IF !Version.Release LT 5.3 THEN MESSAGE, myname() + ' requires IDL 5.3'

  ; Check the input args.  First arg must be a structure, and second
  ; must be an index.  If it's a string, convert to integer.
  IF size(struct, /TName) NE 'STRUCT' THEN MESSAGE, 'Arg 1 must be struct'

  CASE size(index_orig, /TName) OF
    'STRING': index = (where(Tag_Names(struct) EQ StrUpCase(index_orig)))[0]
    ELSE:     index = index_orig
  ENDCASE


  ; Obtain IDL's interpretation of the size...
  ss = size(struct.(index))

  ; ...as well as the HELP command's version.  Find the corresponding line.
  Help, struct, /Struct, out=foo
  foo = foo[index+1]

  ; If this structure element is an array, obtain the dimensions
  array_string = 'Array['
  pos = StrPos(foo, array_string)
  IF pos NE -1 THEN BEGIN
    pos = pos + StrLen(array_string)

    undefine, esmdims

    ; Keep looking for digits, and add them to our own "esmdims".
    WHILE StRegex(StrMid(foo,pos), '[0-9]+', /Bool) NE 0 DO BEGIN
      num = StRegex(StrMid(foo,pos), '[0-9]+', Len=len)

      tmp = long(StrMid(foo,pos+num,len))
      IF N_Elements(esmdims) EQ 0 THEN esmdims=tmp ELSE esmdims=[esmdims,tmp]
      pos = pos + num + len
    ENDWHILE

    ; If the dimensions don't match, override with our own.
    ndims = N_Elements(esmdims)
    IF ss[0] NE ndims THEN ss = [ndims, esmdims, ss[ss[0]+1:*]]
  ENDIF

  RETURN, ss
END
```

--

Eduardo Santiago Software Type esm@lanl.gov RKBA!