Subject: Re: TOTAL(): was Declaration of variables in IDL
Posted by Craig Markwardt on Thu, 04 Oct 2001 01:08:13 GMT
View Forum Message <> Reply to Message

Wayne Landsman <landsman@mpb.gsfc.nasa.gov> writes:

> Craig Markwardt wrote:
>
>> The output of TOTAL is always a floating point type.
>
> Unless, of course, one supplies the /DOUBLE keyword to TOTAL()...

When I said "a floating point type" I meant a generic floating point type, either float, double, complex, or double complex, as opposed to an integer type. I think I am in hearty agreement with your wishes for an integer-aware version of TOTAL().

Craig

> I mention this because at least a couple of times (e.g. when doing
> checksums) I have wished that TOTAL() also had a /L64 keyword. For
> example, according to MACHAR() on my Solaris machine, one loses
> precision with TOTAL() when totaling an integer array that sums to
> more than 2^53, but with a L64 output, one could sum up to 2^64-1
> without losing any precision.

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

Subject: Re: TOTAL(): was Declaration of variables in IDL
Posted by John-David T. Smith on Thu, 04 Oct 2001 15:38:26 GMT
View Forum Message <> Reply to Message

Wayne Landsman wrote:
>
> Craig Markwardt wrote:
>
>> The output of TOTAL is always a floating point type.
>
> Unless, of course, one supplies the /DOUBLE keyword to TOTAL()...
>
> I mention this because at least a couple of times (e.g. when doing checksums) I
> have wished that TOTAL() also had a /L64 keyword. For example, according to
> MACHAR() on my Solaris machine, one loses precision with TOTAL() when totaling

> an integer array that sums to more than 2^53, but with a L64 output, one could
> sum up to 2^64-1 without losing any precision.

You might be interested in this abbreviated exchange with RSI from three
years ago (almost to the day). This predated L64, of course:

JDS

I've always thought total() should just total the data
in the type it was passed, with the option of
conversion.

RSI

This ability to select the summation type to be either integer
or byte is not enabled because in most cases this makes no
sense due to overflow conditions. The maximum allowed value
for an integer and a byte are fairly small and are easily
overflowed.

I had forwarded your suggestion to Development. I thought
that you would want to see their response...there is some
explanation for why TOTAL is limited :

I think the original answer still stands. Total is designed to
be as fast, robust and safe as possible. The ranges on bytes
and ints are so limited that with most array operations would
overflow the type range.

Having said that, I think I should point out that the new array_equal()
function in v5.4 goes a long way to alleviating this problem in
practical cases.

Here's how.

Suppose you have a long vector:

```
v=randomu(sd,10000)
```

and you'd like to compute whether any elements are above .995. Two
older approaches:

1)
```
 wh=where(v gt .995,cnt)
 if cnt ne 0 then print,'Bigger than .995'
```

2)
```
 if total(v gt .995) gt 1. then print,'Bigger than .995'
```

These work, but are unecessarily slow for large arrays, especially if the condition being tested is likely to be satisfied right away. The new function gives us another option:

3)

if array_equal(v le .995,1b) eq 0b then print,'Bigger than .995'

#3 is almost always faster, depending on the random chance of where your statement is first proven false (allowing array_equal() to stop searching). This example is somewhat skewed, since the "v le .995"-type calculations dominate the timing. If these are instead precomputed, I get per run timings of:

#1) 2.45e-04
#2) 7.37e-05
#3) 3.95e-06

If you haven't used this new function, take a look. It also has the salubrious side-effect of making your intentions somewhat clearer.

JD
