# Bug 24131 - trunc() may fail when processing a value from an array

**Status:** RESOLVED NOT_A_BUG
**Version:** 9.6.9 RC 1
**Reported:** 2023-03-03 21:23 EST by Mark Talluto
**Modified:** 2023-03-07 19:57 EST
**CC List:** 2 users (show)

**Depends on:**
**Blocks:**
Show dependency tree / graph

**See Also:**
**Engine:** IDE
**Desktop OS:** Mac OS
**OS X Version:** 12.0.x (Monterey)
**Engine type:** arm64

| Attachments | |
|---|---|
| Stack showing the issue (3.23 KB, application/octet-stream) 2023-03-03 21:23 EST, Mark Talluto | Details |
| Add an attachment (proposed patch, testcase, etc.) | |

**Mark Talluto 2023-03-03 21:23:23 EST** Description

Created attachment 11403 [details]
Stack showing the issue

Expected: trunc() results to be consisent
Observed: trunc() used in a function may fail with certain values in the pass array.

The test stack as a single button.

The following code has a breakpoint to verify the value of the array sDateA["calc"]["days"] before proceeding. The expected value is: 642.041667

```
on mouseUp
--SOME SET UP OF THE ARRAY

 //CALC MONTHS FROM FRACTIONAL PART OF YEARS
   --put getDateValue(sDateA["calc"]["years"],12) into tMonths  --PASS

     breakpoint --******

 //CALC HOURS FROM FRACTIONAL PART OF DAYS
put getDateValue(sDateA["calc"]["days"],24) into tHours --FAIL
end mouseUp


function getDateValue pValue, pMultiplier
     local tDate, tTestVar, tValue

     //VALUE FOR FAIL
     //pValue = 642.041667
     //pMultiplier = 24
     //Expected: trunc() will return 1
     //Observed: trunc() will return 0
     //If pass the values we will fail
     //If we round the value before trunc() it will pass
     //If we hard-code the value it will pass

     //THE TESTS BELOW USE MODIFICATIONS THAT WILL BREAK THE YEAR TEST
```

```
    //FIRST ROUTINE WILL SUPPORT THE YEAR TEST

    //FAIL
    put trunc(pValue) into tDate
    put (pValue - tDate) * pMultiplier into tValue
    put trunc(tValue) --******
    return trunc(tValue)
end getDateValue
```

We would like the first FAIL to pass. There are other tests to show how we can work around this.

We are expecting the trunc(tValue): 1
We are getting a value: 0

**Panos Merakos 2023-03-06 10:02:58 EST** **Comment 2**

Hello Mark,

Thank you for the detailed report.

I think that what you see is a result of rounding error in decimal arithmetic.

So, essentially what happens in the example is:

```
put trunc(55558800/86400) into tDate
put (55558800/86400 - tDate) * pMultiplier into tValue
put trunc(tValue)
```

At this point, the debugger reports that tValue is 1. However, the exact value of tValue is something like:

```
0.9999999999
```

You can verify this by doing something like:

```
put tValue is 1 --> returns false
put tValue - 1 is 0 --> returns false
put tValue - 1 < 0 --> returns true.
```

So trunc(0.999999999) correctly reports 0

I think the problem is caused because the original calculation 55558800/86400 equals 643.04166666666666... (so it cannot be represented exactly in decimal)

Hope this helps.

Setting this to EXPERT_REVIEW

Kind regards,
Panos
--

**Mark Waddingham 2023-03-06 11:31:09 EST** **Comment 3**

Hi Mark,

So as Panos states this is a rounding issue caused by the fact that the fractions you are calculating do not admit a finite decimal (nor binary fp) representation.

This means that there is a tiny error (compared to the ideal) value when you compute the fraction, which is then compounded when you multiply it up.

I'm not sure there is a simple solution here with the approach you are taking beyond using round() as there is no flaw in trunc() or any of the arithmetic operations.

Indeed, this problem isn't even a 'binary floating point' problem as the same would occur with these fractional values if the engine used decimal floating point arithmetic internally - any computed value which has no finite decimal nor binary representation will introduce error unless rounded appropriately.

Is there an issue with using round() in this instance?

Warmest Regards,

Mark.

**Mark Talluto 2023-03-06 15:57:24 EST** **Comment 4**

Hi Panos and Mark,

I reviewed your example Panos and agree that your provided numbers on some calculators will generate a number less than one.

My example generates a number that is greater than one.

What is most interesting is that the same numbers work as expected if they are not passed to the function but provided inside it. The sample stack has examples of this working.

If it fails with the same values, no matter how the values got to the function, then I would agree that it is a rounding error alone.

The workaround is to use round() to make it work for all the test cases I have thrown at it.

**Mark Waddingham 2023-03-07 00:47:56 EST** **Comment 5**

Hi Mark,

Your example stack is not testing the same numbers.

The value being passed to getDateValue (and in sDateA["calc"]["days"]) is 55558800/86400.

Mathematically this is 643.04166666666.... (with a recurring 6).

Computers are finite so the actual value calculated is something like 643.041666666666629.

The value hardcoded in the getDateValue function is 642.041667:

```
642.041667 != 643.041666666666629
```

And even assuming this is actually a typo and you meant 643.041667:

```
643.041667 != 643.041666666666629
```

LiveCode does not turn numbers into strings until the number is required to be a string - when it does so the numberFormat is used - this controls how many decimal places are shown - the default being 6.

If (while in the debugger) you do:

```
set the numberFormat to "#.################"
put sDateA["calc"]["days"]
```

You will see a more accurate value than the hardcoded value you have been using (which gives a different result because it is not the same number!).

Anyway, this is not a bug in LiveCode and using 'trunc()' is not a workaround - rounding is just something you have to do appropriately when attempting to manipulate fractions which cannot be represented exactly.

Warmest Regards,

Mark.

**Mark Talluto 2023-03-07 19:57:52 EST** [Comment 6]

Regarding:

The value hardcoded in the getDateValue function is 642.041667:
   642.041667 != 643.041666666666629
And even assuming this is actually a typo and you meant 643.041667:
   643.041667 != 643.041666666666629

I was experimenting with different values and forgot to reset them.

I now understand the rounding error and that there isn't a flaw in trunc() or round().

It would have been helpful to have the debugger indicate that we had a non-integer in tValue instead of rounding to 1 when I hovered over the variable to get its value. In all fairness, seeing a result of 1 in the debugger and getting a trunc() of 0 should have been a clue.

To get a more accurate representation of tValue throughout its journey (visually in the debugger), I experimented with converting tValue into a string.

The code below is not a solution to the problem but rather a way to see what is happening in the debugger.
The comments at the end of each line are from the debugger. It wasn't until I converted tValue into a string that I realized what trunc() was operating with.

```
L1     put trunc(643.041666666666629) into tDate
L2     put (643.041666666666629 - tDate) * 24 into tValue --tValue = 1 in debugger,
0.99999+ internally
L3     put the num of chars of tValue into tCharCount --tCharCount = 17
L4     put char 1 to tCharCount of tValue into tValue --converts tValue to a string
of 0.999999999999091
L5     put trunc(tValue) --result is 0 as expected
```

**Note**

**You need to log in before you can comment on or make changes to this bug.**