Comparing Programming Approaches in LiveCode: An Order of Magnitude Difference?

http://learninglivecode.blogspot.com/2015/01/comparing-programming-approaches-in.html

I would like to say that this will be my most exciting, riveting, thrilling, hold-on-to-your-seat blog post yet. But, truth be told -- especially to the casual reader with little or no experience with LiveCode -- it may be the most boring one I've written. Of course, if you are a math nerd, perhaps the title piqued your interest a little with the mention of "order of magnitude." And I promise I will use this concept as the cornerstone of this posting. But, if that is the only thing that interests you, feel free to skip to the very end for a wonderful video about the powers of ten to make your decision to click your mouse or tap your screen to come here worthwhile.

Still here? OK, some quick background... In my previous post I mentioned how I received some helpful advice from Richard Gaskin a few months back suggesting that I could improve the performance of one of my projects if I were to take a different programming approach. Let me make it very clear at the start how grateful I am to Richard for taking the time to make this comment. This was a very kind gesture on his part. He was obviously under no obligation to respond, but took significant time to read through what I was working on and posted a very thoughtful comment. And, as I have googled other LiveCode questions and problems over the past two years, it's quite remarkable how many times I find an answer or suggestion given by Richard. He is a exemplary LiveCode citizen who is very willing to share his extensive expertise and who also seems to be constantly scanning the LiveCode world to try to help people out. I had the good fortune to meet him at RunRevLive 2014 in San Diego and he also seemed like a regular guy who was sincerely interested in whatever you were working on.

To briefly recap, I had been using text fields to handle list or data processing tasks and Richard wrote to say that I could dramatically speed things up if I would use variables instead. Here again is what Richard's wrote:
"...by using the "repeat for each" loop method combined with moving data out of the complex field structures into variables for use within the loop, this should bring your processing time down by at least an order of magnitude."
So yes, I've been thinking about this "order of magnitude" thing. I know Richard

knows exactly what an "order of magnitude" means. In contrast, many people who throw this phrase around just kind of vaguely know it means "much more." I think a lot of other people think it means "twice as much." But, actually, an order of magnitude is used to compare things on a logarithmic scale, which most commonly uses base 10. The best example I can think is the Richter Scale, which is used to compare the size of earthquakes. An order of magnitude is a step along the scale that uses the powers of 10. For example, consider the difference between 10^2, 10^3, and 10^4. (I can't easily show exponents in this blog, so I'm using the character ^ to mean "raised to the power of.") If written out, these numbers are 100, 1000, and 10000, respectively. The difference between these three values is not equal, but instead increases exponentially. In fact, I think the best way to think of something measured on a logarithmic scale is to go in descending order, that is, from 10000 to 1000 to 10. At each step, you are going 90% the remaining distance! If you take a really huge number, such as 10^100 (known as a googol, by the way), you eliminate 90% when you go to 10^99. Going to 10^98 likewise removes 90% of that remaining amount, and so on.

So, for Richard to make the assertion that changing my programming approach would improve performance by at least an order of magnitude is really saying something - he's saying that the speed will increase tenfold. But the more I thought about it, I really wondered if this were true. So, I decided to test this hypothesis.

A LiveCode Programming Experiment

To test Richard's hypothesis, I quickly built a LiveCode app that performed a very simple operation: It takes a long list of words and transfers it from one "container" to another. In one version, the container is a text field. In a second version, the container is a variable. The idea is to see how long it takes for LiveCode to complete the process using both types of containers: 1) from one text field to the other text field; and 2) from one variable to the other variable.

And while that sounds simple enough, there are actually several other decisions we have to make. For example, should I take a word from the top of the first list and add it to the top of the second? Or, how about taking a word from the bottom of the first list and moving to the bottom of the second? And, there are quite a few more variations I could throw in. Let's start with what I think would be the most grueling for LiveCode to perform: taking a word from the top of the first list and moving it to the top of the second (thus deleting it from the first list). I'll call this experiment 1.

Here is the code for this loop:

```
put the number of lines in field "field1" into L
repeat with i = 1 to L
  put line 1 of field "field1"&return before field "field2"
  delete line 1 of field "field1"
  put the number of lines in field "field1" into field "field1 total"
  put the number of lines in field "field2" into field "field2 total"

end repeat
```

I titled the two fields "field1" and "field2," with the words being transferred from the first to the second. This loop repeats for as many words (one word per line in the field) as there are in the field. I always take the first word (on line 1) from field1, so you will see "line 1" listed twice above: first to "put" the word into field2, then to delete the word.

The following line above puts that word into the first line of field2:

```
  put line 1 of field "field1"&return before field "field2"
```

As you can see, I also take advantage of the "before" keyword. This keyword very conveniently lets you put something at the beginning of a list of items without having to worry about how to move every other item. (A similar keyword is "after," which I talk about below.) It's also important to concatenate a "return" so that the word solely occupies the first line in field2.

The reason why I call this the most "grueling" approach for LiveCode to accomplish is that the words shown in the two fields are constantly shiftly, as shown in the video below.

The final two lines (before the "end repeat") are used to show a running tally of the number of words in each field as the loop executes. Depending on the experiment being performed, this can be the only feedback given to us that something is actually "working." So, I hope you can see that there is not many lines of code here.

Now, let's take a look at the related code for moving the words from one variable to another.

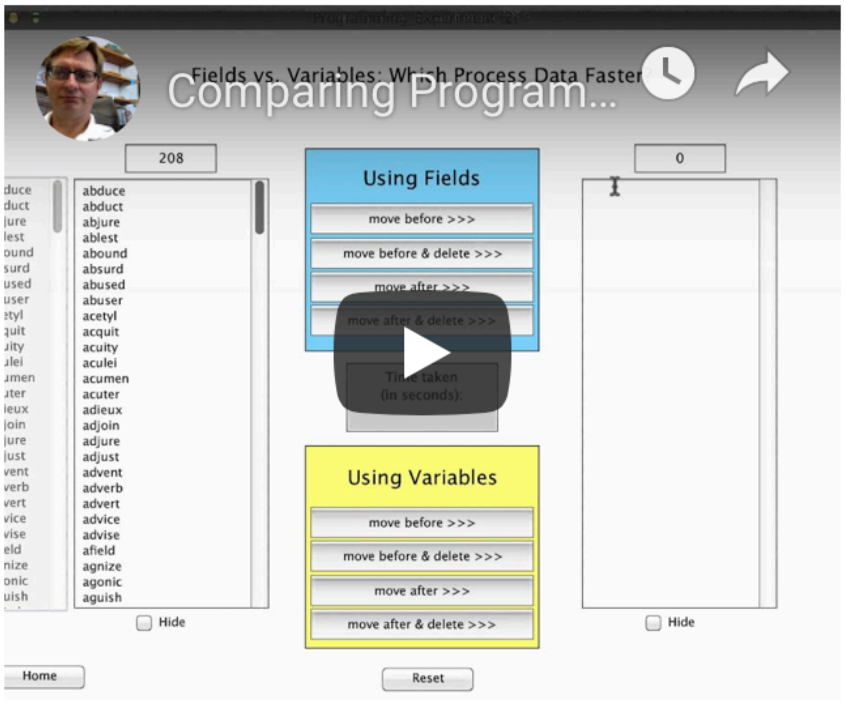
```
put the number of lines in varField1 into L
repeat with i = 1 to L
  put line 1 of varField1&return before varField2
  delete line 1 of varField1
  put the number of lines in varField1 into field "field1 total"
  put the number of lines in varField2 into field "field2 total"

end repeat
```

Your first reaction should be "Whoa, isn't this the same code?" Well, it's almost the same. Instead of referring to the two fields, I simply substituted the local variables "varField1" and "varField2". The important point is that this code processes the words exactly the same between the two examples, except for what containers are used to store the words.

If you are still confused as to what is going on here, watch a short movie I made demonstrating the app:

LiveCode file

[Get the free LiveCode Community version.]



https://youtu.be/lEWLFqD4ZGQ

OK, let's take a look at the results of this first experiment.

Experiment 1 Results

Here are the results from the first experiment (time given in seconds):

	Fields	Variables
move before and delete	141.72	54.59

As these data show, using variables instead of fields for this task is 2.6 times faster when the words are moved to the top of the second field the AND the words from the first field are deleted as they are moved. Yes, that is a big difference, but it is not an order of magnitude difference (i.e. 10 times faster). So, a preliminary conclusion is that Richard is correct in saying that the use of variables will be processed faster, but he overestimated the magnitude.

Experiment 2 Results

OK, let's do another experiment. What if we move the words from the first container to the second, but not delete them from the first. Accomplishing this difference is simply a matter of not including line 4 that begins with the word "delete." This does not seem to me to require as much processing power because you are doing less with the first field. But, who knows, perhaps some other dynamic is in play here that I'm not aware of. It's definitely worth exploring. Here are the results:

	Fields	Variables
move before	73.59	27.30

In this case, using variables instead of fields for this task is 2.7 times faster when the words are moved to the top of the second field, but not deleted from the first field. This is a tad faster, but I'll call it a tie. I am a little surprised that the difference is not lower.

Two More Experiments: 1) Move After and Delete; and 2) Move After, But Don't Delete

I performed two other experiments where I used the keyword "after" instead of "before." As the word implies, this means that each word was added to the second field at the end of the field instead of at the beginning. From a processing standpoint, after the words fill the field in the second field you don't actually see the words being added to the field as they are "below" the bottom edge of the field. So, this would imply that LiveCode can process this more quickly than putting them before. I'll share the data in a moment, but there were no big surprises here. But, I did yet another experiment which did surprise me.

One Last Experiment: What Happens When You Hide the Fields

Watching the program execute this list processing task in all of the previous experiments, particularly when the words are moved to the top of the second field and deleted from the first, it seems clear that moving the words from field to field takes quite a bit of processing power. You can just see the difference. (Actually, I find it quite mesmerizing.) I wondered, though, what would happen if the fields were hidden from view? So, one more experiment was in order.

This last experiment was pretty simple. It's the same as the first experiment, but with both fields hidden. The data from this second experiment were very surprising. There was virtually no difference in processing time between using fields and using variables!

Here is a table with all of the data from all of the experiments:

	Fields	Fields	Variables
	Visible	Hidden	
move before	73.59	27.29	27.30
move before and delete	141.72	54.60	54.59

move after	27.94	27.30	27.29
move after and delete	100.78	54.59	54.57

So, In Conclusion...

My main conclusion is that it doesn't seem to matter if you use fields or variables as the containers for list processing tasks such as the one used here, given the very important condition that the fields are not visible.

But on the Other Hand...

Of course, I might have completely misunderstood Richard's advice and therefore set up a perfectly useless experiment. That's OK, I've wasted time in much more uninteresting ways. But, I am open to any feedback that Richard or any other LiveCode expert may want to give me here. I'm even willing to revise my programming experiment to test other approaches, assuming I can figure out how to code it. Or, better yet, you could download the file yourself and revise it to test your own hypotheses - be sure to let me know what you find.

Many thanks again to Richard Haskin for making me think and for motivating me to explore this programming question. This is exactly the kind of feedback I was hoping for from experts in the LiveCode community when I began writing this blog.

Powers of 10 Video

I end this blog posting (most of which I'm sure many of you wisely skipped) by sharing what might be one of my most favorite science videos of all time. It's called simply "Powers of 10," and it is narrated Phillip Morrison, a famous physicist and science educator who taught at MIT and worked on the Manhattan project. This will truly give you an idea of what is meant by an "order of magnitude."

https://youtu.be/0fKBhvDjuy0

2 comments:

Richard Gaskin January 17, 2015 at 11:08 AM

As your tests show, moving data from rendered fields into variables can make an appreciable difference. Hidden fields save time over displayed fields, since the latter involves a great many more machine instructions to render the text on screen. Variables will still be faster than even hidden fields given the much smaller number of machine instructions in play, but less dramatically so than with visible fields.

A good start, but as I noted before we can probably boost it a bit more by using "repeat for each".

The example below takes both into account by removing all field accesses, such as those that display progress within the loop, and by running the task 100 times to amplify the performance difference to help account for the general overhead of the test itself.

Try this in a button added to your stack:

```
on mouseUp
put fld "field1" into tSrc
put 100 into tIterations
--
-- Test 1: repeat with
put the millisecs into t
repeat tIterations
repeat with i = 1 to the number of lines of tSrc
put line i of tSrc &cr after tDest1
end repeat
end repeat
put the millisecs - t into t1
--
-- Test 2: repeat for each
put the millisecs into t
repeat tIterations
repeat for each line tLine in tSrc
put tLine &cr after tDest2
end repeat
end repeat
put the millisecs - t into t2
--
-- Show results:
put "Results across "& the number of lines of tSrc &" lines "\
& tIterations &" times:"& cr& t1 &" ms "& t2 &" ms "&cr\
& (tDest1 = tDest2)
end mouseUp
```

Here I get:

```
Results across 1637 lines 100 times:
834 ms 25 ms
true
```

The last line of the output is just a sanity check to make sure both tests produce the same result.

The more interesting thing is the "repeat for each" itself - how does it work so quickly?

LiveCode's chunk expressions are wonderfully convenient, but that convenience often masks a fair amount of work the engine needs to do to satisfy our requests.

For example, if we write:

get line 20 of tSomeData

...LiveCode needs to evaluate every character as it traverses the string tSomeData, counting returns until it reaches 20 of them.

In Test 1 in the script above, every time the loop is run LiveCode has to start from the first character of tSrc and count returns until it reaches i. And since it starts at the first line and goes deeper with each iteration, each time through the loop it takes ever longer. Your list of about 1600 lines isn't bad, but if you use "repeat with" on lists of 100k lines or more it can become almost prohibitive.

In contrast, "repeat for each" in Test 2 works very differently. With this form the engine traverses tSrc only one time for all iterations. Each time through the loop it remembers where it left off, and only examines the characters until it finds the first return from where it last left off, after which it puts what it found up to that point into the variable declared in the repeat expression, tLine.

All that said, overall throughout of a routine will be affected by many factors. So while we can show a more than 30-fold speed boost with "repeat for each", that only happens in isolation. Still useful, but if the routine also needs to indicate progress those types of screen-rendering tasks will be slow enough to mitigate many of the performance gains of the newer repeat form.

Still, I feel it's often it's worth measuring such things for exactly that reason: we save clock cycles done in engine stuff so we can spend them informing the user.

If benchmarking becomes as much an obsession for you as it is for me, I put some notes together on that at LiveCode Journal:

Benchmarking Performance in LiveCode
http://livecodejournal.com/tutorials/benchmarking-revtalk.html

PS: I love that Eames video!
Reply
Replies

Lloyd Rieber January 18, 2015 at 2:43 PM
Thanks, Richard, this is very informative. Yes, I did miss your important point about using "repeat for each" form of the repeat loop in your original comment a few months ago. I will try that out.