MouseStillDown and Other Mysteries

by Richard Gaskin
11 July 2011

Every now and then you may come across something in how LiveCode works that'll leave you scratching your head. Most of the time it's not because the designers of the language were crazy, but merely considering aspects of the design that may not be obvious at first glance. The `mouseStillDown` message is one of these, especially when used in a behavior script.

You'll notice that if you have a `mouseStillDown` message in a button script it works fine, but unlike most other messages if you try to handle that in a card, stack, or behavior script it won't trigger at all.

In a way that's kinda good news, as `mouseStillDown` is an anomaly among mouse messages. I think you'll find the other mouse messages quite reliable in behavior scripts, but `mouseStillDown` is an oddity as noted in the Dictionary:

Usually, it is easier and more efficient to use the mouseMove message to track the movement of the mouse while the button is being held down.
Note: If there is no mouseStillDown handler in the target object's script, no mouseStillDown message is sent, even if there is a mouseStillDown handler in an object that's further along the message path.

Your behavior script should work if you add this to the target object:

```
on mouseStillDown
   pass mouseStillDown
end mouseStillDown
```

Why is `mouseStillDown` so different from other mouse messages? Because it's uniquely inefficient, and the `mouseMove` message was provided to provide an alternative that's far more flexible and takes fewer system resources for many similar needs.

Most OSes provide a message when the mouse first goes down, but not all of them provide a second message sent continuously while the mouse is being held down. So to provide `mouseStillDown` for us the engine needs to continually poll the OS for the state of the mouse button. It does this with a frequency defined in the idleRate, which is far less frequent than checking "if the mouse is down" in a repeat loop, but still not optimal.

If the message was always sent into the message path whether or not it's needed, anytime the user holds the mouse down it would trigger a lot of messaging overhead that's never used.
But what if you need it?
Like the idle message (HyperCard's message-clogging workaround for not having timers), all you need to do is include a mouseStillDown handler in any target you want it to be sent to, and the engine will then know to trigger that seldom-used message.
"Seldom-used?"
Yes:
Historically, mouseStillDown was most often used for dragging or other operations in which things need to be updated while the mouse is moving. In such cases, the developer probably doesn't need to update anything until the location of the mouse changes, but mouseStillDown is continually being sent anyway, requiring a lot of redundant processing for things which have no visible effect.
So the mouseMove message was added, providing a way to update things only when the mouse is moved.
MouseMove also works when the mouse is up, which can be useful for update mouse position indicators in a drawing program's rulers, for example.
This additional flexibility requires us to use a few other handlers to substitute for mouseStillDown, but it well worth the few seconds it takes to set up. You'll need a mouseDown to set a flag so the mouseMove can know that the mouse is down without having to poll the OS (you can also use this flag for other useful info, as shown below), and you'll need mouseUp and mouseRelease messages to clear the flag.
This example is for a splitter control that adjusts the groups on either side of it:

```
local sXOffset
on mouseDown
   -- Provide the info mouseMove will need later:
   put the mouseH - the left of me into sXOffset
end mouseDown
on mouseMove
   -- Is the flag still set?
   if sXOffset is not empty then
      -- If so, handle the splitter drag here:
```

```
      set the rect of grp "LeftGroup" to \
         0,0,the left of me, the height of this cd
      set the rect of grp "RightGroup" to \
      the right of me, 0, the width of this cd, the
height of this cd
   end if
end mouseMove
-- Clear the flag when the mouse is released over the
control:
on mouseUp
   put empty into sXOffset
end mouseUp
-- Clear the flag when the mouse is release when not
over the control:
on mouseRelease
   put empty into sXOffset
end mouseRelease
```

In addition to handling simple drags like a splitter, drag-and-drop operations can be handled using the messages provided for those (`dragStart`, `dragMove`, `dragDrop`, `dragEnd`) far more simply than emulating drag-and-drop behaviors with `mouseStillDown`.
So once we use `mouseMove` for movement-related things and the drag-and-drop messages for those types of actions, the remaining subset of cases where `mouseStillDown` can be useful are relatively few. And for those, you can still use it so long as you provide a handler for it in the target object.
Another LiveCode convention that can be tricky to wtap your head around at first is what `me` refers to in a behavior script. In most cases me refers to the object containing the script, but not so when used in a behavior script. When you first discover this, it may be a bit confusing.
More good news: the uncertainty you may feel is well addressed by the implementation. If you experiment a bit you'll find that "me" in a behavior script will always refer to the target object. This is very valuable in behavior scripts so the script can easily identify any object using it.
Once you've experimented to verify this, you'll become confident with their use and will want to use behavior scripts every day. And every night.

And on the weekends. Highly addictive. :) Behaviors are one of the most powerful additions to the language since arrays.

The key to learning nearly anything is experimentation. Dive in and play, see what works and what doesn't and learn why.

Having addressed the unique anomaly that is the `mouseStillDown` message, I hope you're inspired to explore behaviors and other powerful features LiveCode offers.

While your own confidence may not be as strong as the moment, I'm completely confident with with just an hour or two's experimentation to verify how behaviors work, you'll enjoy using them going forward and have simpler and more robust code in the process.