Commands and Private Handlers

by Sarah Reichelt

LiveCode 2.8.1 introduced two new concepts in naming and using handlers: "command" and "private". "Command" is used instead of "on" for handlers that are not responses to system messages. "Private" enables you to declare certain handlers (functions or commands) as being for use inside their own script only.

The differences between functions & commands:

Functions are handlers that must return a value. They are called by using the function name followed by a pair of brackets containing any parameters. The returning value must be dealt with in some way, by being stored into a variable or field or by being stored in the special variable "it".

Commands do not have to return a value, although they can if you want. They are called by just using the name of the command followed by any parameters with no brackets required.

The easiest way to think of the difference is the consider that a call to a function is asking for information, while a call to a command is telling the computer to do something. While there are exceptions to this simple rule, it should make it easier to work out when each one is needed.

As an example, supposing we have a field called "Elapsed time". We could have a **function** that returned it's value:

```
function currentElapsedTime
return field "Elapsed time"
end currentElapsedTime
```

and a **command** that sets it to a new value:

```
on setElapsedTime pNewValue
put pNewValue into field "Elapsed time"
end setElapsedTime
```

To use these handlers, we would have something like:

```
put currentElapsedTime() into tElapsed
```

...and call it with:

```
setElapsedTime 300
```

Note that the example calling the function took the returned value and stored it into the tElapsed variable. The call also included the brackets even thought there are no parameters. In the call to the command, there were no brackets and the single parameter was just added on.

(These examples are so simple that you wouldn't actually bother to make them in separate handlers, but they show my point).

Command:

In the example of a command, I used the old method of prefacing a command's handler name with the word "on". The new way is to indicate a command using the word "command", unless the command is actually a system message.

Imagine a button script:

```
on mouseUp
showNewData
end mouseUp

command showNewData
put random(1000) into field "Random"
end showNewData
```

"mouseUp" is a system message, so the keyword "on" is used. We want this handler to be triggered "on" a mouseUp event. The "showNewData" handler is a command. It is not triggered directly by a system message, so it gets the keyword "command".

At the moment, "on" and "command" are synonyms so can be used inter-changeably, but this may not always be the case, so I recommend that you convert to using "on" for system messages and "command" for all other command handlers.

To check if something is a system message go to the Help and choose "Dictionary". If you have the Dictionary arranged so that the list of keywords is in a column on the left, click the white equals symbol just to the right of the keywords column header. This will put you into the horizontal display mode where you can display multiple columns for each dictionary entry. The one we are interested in is the "Type" column. If it isn't showing, right-click (control-click) in any of the column headers and choose "Type" from the popup menu that appears. Now click the "Type" header to sort by type and scroll the list until you get to all the entries of type "message". These are the ones that should get the "on" keyword.

Private:

The "private" keyword is a way of marking a command or function so that it is only accessible from inside the script containing it. The main advantage to this is when writing libraries that you wish to share. If I write a library including a command called "showClock" and you already have a command in your scripts called "showClock", then if you have my library as a frontScript, my "showClock" will work instead. If not, then yours will happen instead of mine.

However if I had declared my "showClock" handler as a private command, then there could be no possible conflict. My scripts inside my library would use my handler and all your scripts that were not inside my library would continue to use yours as if mine did not exist.Â

Each library will require at least one public handler or it can never be used, but it will be a lot easier only having to think of one unique name, instead of trying to keep every handler name unique.

The "private" keyword can be applied to either commands or functions. It should not be applied to message handlers using the "on" keyword. You cannot "pass" a private handler as they are not part of the normal message path, but this makes them faster than they would be in they were not private.

One limitation of private handlers is that they cannot be used with "send". I understand why this is so from outside their own scripts, but I would have thought that using a "send" to call a private handler from within a script would have been OK. This limitation means that they cannot be used for timed events. It also means that they cannot be used for callbacks.

Example stack:

Download the example stack "Private.rev" and open it in LiveCode 2.8.1 (or higher). Make sure "Use private function" is checked and have a look at the stack script in your script editor. You will see the three public functions that calculate volumes and a private function that calculates the area of a circle. Try using the top three buttons to calculate volumes. if you look at their scripts, you will see that they all call one of the public functions, but only the public functions in the stack script call the private function.

Now try the "Call a private function" button. This will give you a script error as if you were trying to use a nonexistent function. The private function cannot be called from any script other than the one that contains it.

Now have a look at the "Library" button's script. You will see that it contains a different and public areaCircle function that gives a nonsensical answer. Check the button to put this script "In use" (this makes it a front script) and try using any of the 3 Volume buttons at the top. You will see that they all still work fine, so they are still using the private function, not the one in the Library.

The "Use private function" checkbox at the bottom switches the stack script between using a private or a public function. Turn it off and try the "Call a private function" button again. This time it will work as there is no problem calling a public function. Make sure the "In use" button is checked and try using one of the Volume buttons. They will all return a volume of zero because the function in the Library is being used instead of the one in the stack script. As the stack script function is no longer private, this is following the standard message hierarchy.

Warning:

If you use either "command" or "private" in your scripts, they will not work with any version of LiveCode earlier than 2.8.1! If you think people using older versions of LiveCode might be using your stacks, then do not use these keywords.