Conditional Structures - Part 1: if/then/else

Eric Chatonet

23 May, 2005

Preamble

This article is dedicated to beginners: I hope they will find here a clear introduction to conditional structures.
And, who knows, some more advanced users might find here some useful remindersÂ too :-)

Introduction

Factual programming with object oriented languages like LiveCode consists mainly in responding to the userâ€™s actions when he/she moves the mouse, clicks, press a key and so on.
By sending messages like openCard, mouseUp or closeField, the engine informs about all what happens. Then you intercept the messages that are relevant in your environment by writing a proper handler:

```
on preOpenStack
  set the loc of this stack to the screenLoc
end preOpenStack
```

Here it is very simple but more often you will have first to analyze the context before responding: this is the basis for proper interaction.
Good programming lies in being prepared for all eventualities (including errors), finding which one occurs and then providing the right answer.
As for testing what is happening, you can be confronted to simple options or need to manage very complex analysis.
In all cases, LiveCode provides control structures to make the job: if/then/else and switch.

In this article (part 1), we shall explore `if/then/else`.
Later in part 2, we shall explore `switch`.

1. What is a condition?

Not exactly as in the real life :-) any condition when programming has to strictly evaluate as true or false (a Boolean value):

```
the platform
-- MacOS, Win32, etc: cannot be used as a condition
the number of cards
-- an integer: cannot be used as a condition

the platform = "MacOS"
the number of cards = 1
there is a button "Quit"
-- are strictly true or false: can be used as
conditions

MyBooleanTest()
-- depends on the value returned by this function
```

Of course, to be true a condition is not necessarily positive:

```
the platform <> "MacOS"
not the hilite of button 1
there is no button "Quit"
```

Mathematic equalities (or inequalities), most LiveCode properties values are true or false and then can be used as conditions.

2. If/Then simple conditional structure

***If** there are some bananas **then** give one to Dan*

Translated to LiveCode this simplest form will be:

```
    if <condition> then <command>
```

You may write it in different ways that are equivalent:

```
if <condition> then
  <statements>
end if
```

or:

```
if <condition>
    then
    <statements>
end if
```

```
-- all structure words appear at the beginning of a
line
```
Choose the one that seems the best legible for you.
`If` indicates the beginning of the conditional structure.
`If` is followed by the condition statement.
`Then` precedes the statements that will be executed if the condition is verified.
`End if` indicates conventionally the end of the structure.
Notes:
Closing the control structure by `end if` is compulsory when using more than one line.
Indentation reflects the structure (in order to set the correct indentation, just place the insertion point anywhere into the structure and press the tab key).
When `if` and `end if` are not aligned this means that you made an error…
When your statements are a list of statements, you canâ€™t use the one line form.
In order to avoid errors, itâ€™s a good idea to be in the habit of writing immediately the control items before specifying the statements: You will understand the benefits when writing nested complex structuresÂ :-)
Example:
```
on openCard
  if the number of this card = 1 -- condition
  then
    disable button "Previous" -- action
    -- you could add other statements here
  end if -- end of the structure
end openCard
```
In this example, the openCard handler will disable a button if it is the first card that is displayed.
Tips
You can write:
```
if the hilite of btn "Check Box" is <trueOrFalse> then
```
But in fact, this formulation is redundant because such a condition is always true or false. So you can only write:
```
if the hilite of btn "Check Box" then
if not the hilite of btn "Check Box" then
```

It might help you when beginning (but makes reading easier for all) to enclose conditions within parentheses:

```
if (the hilite of btn "Check Box") then
```

3. If/Then/Else simple conditional structure

If there are some bananas then give one to Dan else give him an apple

```
    if <condition> then <command> else <another
command>
```

Here too, you may write it in different ways which are equivalent:

```
if <condition>
then <command>
else <other command >
-- not really legibleÂ : prefer indented forms
```

or:

```
if <condition> then
  <statements>
else
  <other statements>
end if
```

or:

```
if <condition>
then
 <statements>
else
 <other statements>
end if
```

`If` indicates the beginning of the conditional structure.
`If` is followed by the condition statement.
`Then` precedes the statements which will be executed if the condition is verified.
`Else` precedes the other statements which will be executed if the condition is not verified.
`End if` indicates conventionally the end of the structure.

Notes

Then and else statements are always exclusive.
This means that if the condition is verified, the engine executes the then statements and skips the else statements without evaluating them.

Conversely, if the condition is not verified, the engine skips the then statements without evaluating them and goes to the else statements immediately.
Example:

```
on openCard
  if (the number of this card = 1) then
    -- verified condition
    disable button "Previous" -- action
  else
    -- unverified condition
    enable button "Previous" -- action
  end if -- end of the structure
end openCard
```

In this example, the `openCard` handler will disable a button if it is the first card that is displayed. When another card will be displayed (else), the button will be enabled.
Break:
LiveCode is a very flexible language that accepts many kinds of formulations and here you might prefer a â€œwithout conditionâ€ one line handler as following:

```
on openCard
   set the enabled of btn "Previous" to \
    the number of this cd <> 1
end openCard
```

4. Multiple inclusive conditional structure with *and* & *or*

If there are no bananas or there are no apples then give Dan a pear

```
if <condition 1> or <condition 2> then
  <statement>
end if
```

If *there are bananas* ***and*** *there are apples* ***then*** *give Dan both*

```
if <condition 1> and <condition 2> then
  <statement>
end if
```

`And` & `or` operators are handy to specify more sophisticated conditions:

```
put the number of this cd into tNum
if (tNum <> 1) and (tNum <> the number of cds) then
-- intermediate cards
```

You can mix `and` & `or` operators. In such a case, take operators precedence into account (`And` level is higher than `or` level).
Using well placed parentheses the precedence level of which is the highest can then override all others:

```
if <condition1> or <condition2> and \
    <condition3> or <condition4> then
```

Without any parentheses the engine will evaluate this to C1 or (C2 and C3) or C4.

```
if (<condition1> or <condition2>) and \
    (<condition3> or <condition4>) then
```

With parentheses the engine will evaluate that to (C1 or C2) and (C3 or C4) and that is usually that you want :-)
Notes:
When using the and operator, Rev use a short-circuit evaluation: it evaluates the first found condition. If it is true, it goes on and evaluates the second condition but if the first one appears to be false, it skips the second one because it is unnecessary to evaluate it.
With the or operator, all conditions are of course evaluated.

5. Multiple exclusive conditional structure with else if

If there are some bananas then give one to Dan else if there are some apples give him an apple else (innuendo: if there are no bananas and no apples) give him a pear

```
if <condition 1> then
  <statements>
else if <condition 2> then
  <other statements>
else
  <another statements>
end if
```

Often, a single condition is not enough to make the job. Then you will use another form of writing conditional structures that allows introducing as many conditions as you need.
With that form any condition excludes all others.
Notes:
The last condition (`else` only in the example above) means all cases that do not match any previous condition.

You can add as many `else if… then` lines as you need. But if they are very numerous, you might prefer use the more flexible `switch` control structure (article to come).
Example:

```
the platform
-- MacOS, Win32, etc: cannot be used as a condition
the number of cards
-- an integer: cannot be used as a condition

the platform = "MacOS"
the number of cards = 1
there is a button "Quit"
-- are strictly true or false: can be used as
conditions

MyBooleanTest()
-- depends on the value returned by this function
```

4
Tips:
Sometimes it can appear useful to specify all planned conditions strictly and keep the last else for tracking errors.
On the other hand, when trapping system messages you will find handy to keep the last else to pass the message:

```
on commandKeyDown pKey
  if pKey = "s" then SaveStack â€“ custom handler
  else pass commandKeyDown
end commandKeyDown
```

6. Nested combinations
A verified condition often calls another condition that calls…
To handle more complex analysis, you can use nested conditional structures:

```
the platform
-- MacOS, Win32, etc: cannot be used as a condition
the number of cards
-- an integer: cannot be used as a condition

the platform = "MacOS"
the number of cards = 1
there is a button "Quit"
-- are strictly true or false: can be used as
conditions

MyBooleanTest()
-- depends on the value returned by this function
```
6
In this example, 2 conditional structures are used.
Indentation lets you locate them easily…
The main one is a simple `if/then/else` and the other one an `if/else` if: you can mix all types to suit your needs!
There are no limitations when interlocking conditional structures: only the common sense will stop you… showing that your programming architecture can be improved :-)

Conclusion
Conditional structures bring to you interaction abilities with the user's environment, what he/she does, etc. So, there are major blocks of the LiveCode language.
The many ways of using them, combining several structures, adding tests with logical operators, etc… makes them flexible and very near from every day language.
They require only a logical mind that forgets the almostâor nearlyâ ways of thinking: that might be their only default :-)

Related topics in Revolution documentation
To go further you might be interested to check the following entries in the Revolution documentation dictionary:

- If control structure
- Then keyword
- Else keyword
- End if control structure
- And operator
- Or operator
- How I pretty print a script? (Topics section)