This paper belongs to my MuseScore issue from 2020-08-22 with the title “automatic renaming of chord symbols when changing the fret number of a fretboard diagram.”

Here you have a detailed description of how the automatic renaming of chord symbol names could be processed in relation to the change of fret number in the fretboard diagrams inspector.

The following prerequisites are supposed:

1. Any fretboard diagram attached from the templates section to a note/chord on the stave brings along its chord symbol.
2. It is detectable if a chord symbol exists for the same note where the fretboard diagram is attached to (because chord symbols can be deleted separately).
3. Access on the (existing) chord symbol name is granted.
4. The string of the chord symbol name can be parsed und partly replaced by another string.
5. The initial fret number of the diagram is detectable (it is shown in the fret number counter box before the change is made).
6. The number of moved frets can be determined (it’s the difference between the initial fret number and the fret number after the change has been made.
7. A mouse click on one of the up/down arrow keys of the fret number counter box can trigger a bunch of commands. During its execution the fret number counter box does not react to any further mouse click or keyboard input. Only if the execution has finished the keys are functional again. As well any direct keyboard input into the fret number input field can trigger the same bunch of commands and inhibits further keyboard or mouse input for the duration of the execution of those commands.

The following facts are objectively given and are used for the method.

1. The first character of a chord symbol name is always the root letter of the chord. There are 7 possible root letters, C, D, E, F, G, A and B.
2. The second character of a chord symbol name *can be* an accidental. There are only 2 accidentals: b and #. If the second character is one of them, the root name of the chord is comprised of the first *and* the second character of the chord symbol name. If the second character is neither of both, the root of the chord is comprised of only the first character of the chord symbol name.
3. The remainder of the string of the chord symbol name from the third character onwards is irrelevant for our case and is kept unchanged in every way. This remainder consists of the digits for additional triads plus a possible accidental or some other characters. Only the digits 5, 7, 9, 11, 13 can occur at the same time, with an accidental or a sign of similar meaning. This amounts to a theoretical maximum of 12 characters following the root name. In practice, the remainder will hardly come up to this number.
4. The relation between the initial fret of a diagram from the template section and the root letter of the chord symbol that comes with the diagram is fixed and relative. Once a diagram is initially bound to a certain fret and a chord symbol is bound to the diagram, the root letter of the chord moves relatively to the move of the diagram on the fretboard.
5. Moving the diagram one fret upwards or downwards is equal to changing the root letter of the chord for one half-tone step. Moving for several frets means changing the root letter for several half-tone steps or whole-tone steps. The relation between the position on the fretboard and the root letter is biunique.
6. If the root letter of the initial chord has a sharp, sharps should be maintained when moving the diagram up and down the fretboard. When having a flat, flats should be maintained. If the root letter of the initial chord has no accidental at all, sharps should be used in an upward move and flats in a downward move. In this case moving around can produce different sorts of root letters.
7. There are two methods of determining the new chord symbol name.
I. You retrieve the root letter of the chord from the symbol and determine the number of frets and the direction, the diagram has been move on the fretboard. You consider a set of only 7 letters and identify the position of the retrieved root letter within this set. According to the number of moved frets you find the new root letter of for the chord symbol. With this method you must conclude logically whether an accidental has to be added or not. The set is comprised of a comma-separated list with the 7 elements
“C, D, E, F, G, A, B”.
II. As before you retrieve the root letter of the chord from the symbol and determine the number of frets and the direction, the diagram has been move on the fretboard. But now you consider a general set of 12 combinations of a letter plus an accidental. The set of possible chord names would then comprise a comma-separated list as follows:
“C, C#, D, D#, E, F, F#, G, G#, A, A#, B”. Again you identify the position of the retrieved root letter within the set and then perform the same number of steps into the same direction as with the diagram on the fretboard. This way you get the new root letter for the chord symbol.
Both methods can be understood as a circle, i. e. moving forward from the last element returns to the first element and moving backward from the first element returns to the last element.
8. Both methods are applied immediately when changing the fret number of a diagram – before the fret number counter box becomes active again for a possible next change of the fret number. Thus it is not necessary to recognize the moment when editing fret numbers has ended (which would prove difficult anyway). Considering mouse actions this is always a step of 1. With input via keyboard being possible too, it gets a bit more complicated, for keyboard input can produce steps greater than 1.
9. With the method I you parse the first and second character of the initial chord symbol name separately. Then you determine how many steps in one direction have been made. In case of *upward* moves, you decide whether you started from a chord name with a plain letter or a chord name with a letter plus accidental – and in the latter case whether the accidental has been a sharp one or a flat. For plain letters, a sharp accidental would be added when an odd number of steps had been done (because the move was *upwards*). If a sharp was already there, the accidental will be removed if an odd number of steps had been done. If a flat accidental was already there, the accidental is removed in case of an odd number of steps. The same proceedings concern downward moves, just vice versa. Furthermore, attention must be paid, when moving upwards across letter E or downwards across letter C, because accidentals might possibly have to be omitted in certain cases. Still, this method has a problem when moving upwards from a chord letter with a sharp accidental and then for some reason back down. This could result in a different chord name (with flat accidental), although it should be the same as the initial one. For this and some more reasons method II would prove much more suitable.
10. With the method II, however, you would need a second set comprised of the 12 elements
“C, Db, D, Eb, E, F, Gb, G, Ab, A, Bb, B”.
This method doesn’t need any of those sophisticated thoughts on whether to use an accidental or not and which one. You just again parse the initial chord name’s first two characters separately. If the second character is a flat or sharp, you combine it with the first character into a search string. If the second character is neither a sharp nor a flat, the search string is comprised only of the first character. Then you identify the search string within the set of possible chord names. As you have already parsed the second character, you right away know, which set is to be used, that with the sharps or that with the flats. If neither sharp nor flat exists in the initial name, the decision which set is to be used is based upon the direction of the move. The direction can easily be determined by subtraction of the initial fret number of the diagram from the final. Positive results are “upwards”, negative results are “downwards”. This method has also the advantage that resulting chord names stay consistent when moving the diagram to and fro on the fretboard. This behaviour only works if the use of one or the other set is decided before processing any fret number. I. e. this decision (and along with it all of the parsing of the initial chord name etc.) has to be made immediately when activating the inspector for a chord diagram before the fret number counter box becomes active. The values from that initial parsing should stay valid as long as the inspector relates to that very diagram. They are reset to default as soon, as any other object in the stave gets marked and the inspector changes into this object's properties.

I will try to put these ideas into sort of a pseudo code.

At the moment of marking a diagram object in the stave the following variables should be initiated from their default value false, 0 or “” (Null-string).

SETF = “C, Db, D, Eb, E, F, Gb, G, Ab, A, Bb, B” ### SETF (set of possible root names with flats)
SETS = “C, C#, D, D#, E, F, F#, G, G#, A, A#, B” ### SETS (of possible root names with sharps)
F = get\_fret\_number() ### F (initial fret number of diagram, before
 ### any change has been made)
If exist chord\_symbol ⇨ E = true end; ### E (Boolean marker if a chord symbol is existent.
 ### It concerns only the chord symbol that
 ### is attached to the same note/cord as the
 ### marked diagram!
If not E = ⇨ end\_method() ### when no chord symbol exists
else I = get\_chord\_symbol\_name(), ### I (initial chord symbol name as it was
 ### before the inspector had been activated)
 if I[2] = “b” ⇨ FLAT = true; ### save permanently the information on the
 SHARP = false; ### chord names set that is to be used
 R = I[3;12] ### R (remainder of chord name) cannot get longer
 ### than 12 characters
 else if I[2] = “#” ⇨ SHARP = true; ### toggle the set to be used depending
 FLAT = false; ### on the accidental in the initial chord name
 R = I[3;12] ### R (remainder of chord name)
 else SHARP = false; ### if no accidental existed do not prefer any
 FLAT = false; ### at the moment
 R = I[2;12] ### R (remainder of chord name)
 end ### now the mode of the root letter and the
 ### remainder of the initial chord have been
 end; ### saved permanently
 :start\_processing; ### jump point for repeated steps
 IF = I[1]; IS = I[2]; ### IF (first character), IS (second character)
 if FLAT or SHARP ⇨ S = I[1;2] ### S (search string, letter plus accidental.
 else S = I[1] ### if root name contains neither flat nor
 ### sharp, it’s just a plain letter
 end; ### now the actual chord name has been
 ### parsed, even for several steps
 if fret\_number\_counter\_mouse\_action() ⇨ ### if fret number has been changed with
 ### mouse. Attention, the change via keyboard
 ### has to be detected here, too!
 M = get\_new\_fret\_number(); ### retrieve the new fret number.
 OFFS = (M – F); ### OFFS (number of moved frets)
 if OFFS > 0 ⇨ ### upward move
 if (not FLAT) and (not SHARP) ⇨ ### when root has been a plain letter
 POINT = pos(S;SETS); ### POINT\* (pointer to the element in the
 ### *sharpened* set of possible chord
 ### names – because of upward move –
 ### in case of a plain root letter)
 if (POINT + OFFS) < 12 ⇨ POINT = (POINT + OFFS) ### offset stays within the boundaries of
 ### the set of possible chord symbol names
 else POINT = (POINT + OFFS – 12) end; ### (POINT + OFFS) > 11 and offset exceeds
 ### beyond the upper boundary of the set,
 ### hence must be wrapped
 end; ### POINT now points to the new root letter
 ### and has been wrapped around if necessary
 N = SETS[POINT]; ### N (new name for chord symbol comprised
 concat(N, R); ### of the offset-element from the chosen set
 ### plus unchanged remainder of the initial
 ### chord name which can’t get longer than
 ### 12 characters)
 end; ### end of processing plain root letter
 if FLAT ⇨ ### when root has been a flattened letter
 POINT = pos(S;SETF); ### POINT\* (pointer to the element in the
 ### *flattened* set of possible chord
 ### names – despite of upward move –
 ### because of the flattened root letter)
 if (POINT + OFFS) < 12 ⇨ POINT = (POINT + OFFS) ### offset stays within the boundaries of
 ### the set of possible chord symbol names
 else POINT = (POINT + OFFS – 12) ### (POINT + OFFS) > 11 and offset exceeds
 ### beyond the upper boundary of the set,
 ### hence must be wrapped
 end; ### POINT now points to the new root letter
 N = SETF[POINT]; ### N (new name for chord symbol comprised
 concat(N, R); ### of the offset-element from the chosen set
 ### plus unchanged remainder of the initial
 ### chord name
 end; ### end of processing flattened root letter
 if SHARP ⇨ ###
 POINT = pos(S;SETS); ### POINT\* (pointer to the element in the
 ### *sharpened* set of possible chord
 ### names because of the sharpened
 ### root letter)
 if (POINT + OFFS) < 12 ⇨ POINT = (POINT + OFFS) ### offset stays within the boundaries of
 ### the set of possible chord symbol names
 else POINT = (POINT + OFFS – 12) end; ### (POINT + OFFS) > 11 and offset exceeds
 ### beyond the upper boundary of the set,
 ### hence must be wrapped
 end; ### POINT now points to the new root letter
 N = SETS[POINT]; ### N (new name for chord symbol comprised
 concat(N, R); ### of the offset-element from the chosen set
 ### plus unchanged remainder of the initial
 ### chord name
 end; ### end of processing sharpened root letter
 else ### downward move (M – F < 0)
 ### the situation M-F = 0 can never occur,
 ### because then M equals F and no change
 ### had been made, which means neither a
 ### mouse action nor a keyboard input had
 ### been performed (or the same fret number
 ### as before had been input and that should
 ### not be considered a keyboard input).
 if (not FLAT) and (not SHARP) ⇨ ### when root has been a plain letter
 POINT = pos(S;SETF); ### POINT\* (pointer to the element in the
 ### *flattened* set of possible chord
 ### names – because of downward move –
 ### in case of a plain root letter)
 if (POINT + OFFS) > 0 ⇨ POINT = (POINT + OFFS) ### offset stays within the boundaries of the
 ### set off possible chord names
 else POINT = (12 + POINT + OFFS) ### (POINT + OFFS) < 1 and offset exceeds
 ### beyond the lower boundary of the set,
 ### hence must be wrapped
 end; ### POINT now points to the new root letter
 ### and has been wrapped around if necessary
 N = SETS[POINT]; ### N (new name for chord symbol comprised
 concat(N, R); ### of the offset-element from the chosen set
 ### plus unchanged remainder of the initial
 ### chord name)
 end; ### end of processing plain root letter
 if FLAT ⇨ ### when root has been a flattened letter
 POINT = pos(S;SETF); ### POINT\* (pointer to the element in the
 ### *flattened* set of possible chord
 ### names – despite of upward move –
 ### because of the flattened root letter)
 if (POINT + OFFS) > 0 ⇨ POINT = (POINT + OFFS) ### offset stays within the boundaries of the
 ### set off possible chord names
 else POINT = (12 + POINT + OFFS) ### (POINT + OFFS) < 1 and offset exceeds
 ### beyond the lower boundary of the set,
 ### hence must be wrapped
 end; ### POINT now points to the new root letter
 N = SETF[POINT]; ### N (new name for chord symbol comprised
 concat(N, R); ### of the offset-element from the chosen set
 ### plus unchanged remainder of the initial
 ### chord name
 end; ### end of processing flattened root letter
 if SHARP ⇨ ###
 POINT = pos(S;SETS); ### POINT\* (pointer to the element in the
 ### *sharpened* set of possible chord
 ### names – despite the downward move –
 ### because of the sharpened
 ### root letter)
 if (POINT + OFFS) > 0 ⇨ POINT = (POINT + OFFS) ### offset stays within the boundaries of the
 ### set off possible chord names
 else POINT = (12 + POINT + OFFS) end; ### (POINT + OFFS) < 1 and offset exceeds
 ### beyond the lower boundary of the set,
 ### hence must be wrapped
 end; ### POINT now points to the new root letter
 N = SETS[POINT]; ### N (new name for chord symbol comprised
 concat(N, R); ### of the offset-element from the chosen set
 ### plus unchanged remainder of the initial
 ### chord name
 end; ### end of processing sharpened root letter
 ### the new chord symbol name now has
 ### been created for all sorts of root letters
 I = N ### the new chord symbol name is written back
 ### into to the chord symbol itself and at the same
 ### time is assigned to the initial chord symbol
 ### name variable, in case a further change should
 ### take place
 end; ### reactivate fret\_number\_counter for mouse
 ### action or keyboard input
 goto start\_processing ### check if another mouse action / keyboard
 ### input has occurred on fret\_number\_counter
end; ### wait until inspector is closed or activated for
 ### another object.

\* SETF and SETS are arrays of 11 elements each, with the first element having index 1. Retrieving the element n with pos() means to get the complete Element, even if its 2 Bytes long (letter plus accidental). Neither this nor starting indexes with 1 instead of 0 is proper syntax, I know. It is just to illustrate the method.
E, FLAT and SHARP are Boolean indicators. F, M, POINT and OFFS are integers. I, IF, IS, N, R and S are strings. Supposedly the code can be shortened a lot. It’s only a draft. And perhaps you find a completely differen and more elegant solution.

The described method has certain limitations which I consider acceptable. To take precautions for each and every condition would render the method much too complicated und would stand in no relation to the rare frequency of occurrence of those situations. As far as it comes to my mind these limitations are:

1. The initial chord symbol name, which the diagram brings along, could already be a wrong one. If so, it must have been stored in the templates already with a wrong symbol name. It is of course the liability of the user to provide for correct chord symbol names in the templates!
2. In rare situations a user might prefer a sharpened root name, after executing several steps with the fret number, although the initial root had been a flattened one (or vice versa). In this case the root name indeed has to be edited manually. Providing possibilities for such an unsystematic approach would presume another element with which the preferred set of chord names – flattened or sharpened – explicitly could be chosen (and the element had to be incorporated in the code and handled accordingly). The whole procedure would get too complex and confusing, I guess. A workaround might be to move the diagram one fret upwards to a chord symbol with no accidentals, then unmark the diagram (in order to reset the Booleans SHARP and FLAT), mark it again and move it back to the former fret. For flats to be turned into sharps you vice versa move the diagram down one fret, unmark and mark it and move the diagram one fret up again. But this workaround is too cumbersome and editing the chord symbol directly is a faster and more convenient way.

Thank you for your patience and for your interest in this issue.

Michael Gimmel