Case Study: Rotating Pitches To the left rather than the right using Transformations plugins

Bob Zawalich 29 June 2021

When the Transformation plugins were designed in Sibelius 6, we chose the word Rotate to mean “apply the value each selected note to the following note, and apply the value of the last note to the first note”.

They only go one direction. I think of the notes being laid out as a circle, and moving clockwise. These are really tricky, especially when you move the rhythms, because you need to split over barlines and deal with tuplets and all sorts of craziness. I would say that there is probably no chance that they will ever be rewritten to go in reverse (though I am not remotely a Sibelius employee, just the guy that wrote the code in the first place).

Here are the descriptions of the rotate plugins in the Reference:

Rotate Pitches
Rewrites the selection so that the pitches of the notes are shifted to the right by one note (so the pitch of the last note becomes the pitch of the first, the pitch of the first note becomes the pitch of the second, and so on), without changing the rhythms of the notes.
Select a passage and choose Note Input > Transformations > More > Rotate Pitches. The passage is rewritten in place.

Rotate Rhythms
Rewrites the selection so that the durations of the notes are shifted to the right by one note (so the duration of the last note becomes the duration of the first, the duration of the first note becomes the duration of the second, and so on), without changing the pitches of the notes.
Select a passage and choose Note Input > Transformations > More > Rotate Rhythms. The passage is rewritten in place.

Rotate Rhythms and Pitches
Rewrites the selection so that both the durations and pitches of notes in the selection are shifted to the right by one note (so the last note of the selection becomes the first note, the first note becomes the second note, and so on).
Select a passage and choose Note Input > Transformations > More > Rotate Rhythms and Pitches. The passage is rewritten in place.

So if we had 6 notes, represented by the numbers 1 – 6, and applied the plugin until we came back to the start, we would see this pattern.

123456
612345
561234
456123
345612
234561
123456

With only 3 notes, we would have

123
312
231
123

So if you want to run the plugin multiple times to simulate going in reverse, you need to know how many notes you are dealing with, and run it that one fewer times than the number of notes.

If we had 3 notes and wanted to rotate left we would want to see

123
231
312
123

How do we get from 123 to 231? We run the right rotate plugin 2 times (the number of notes -1)

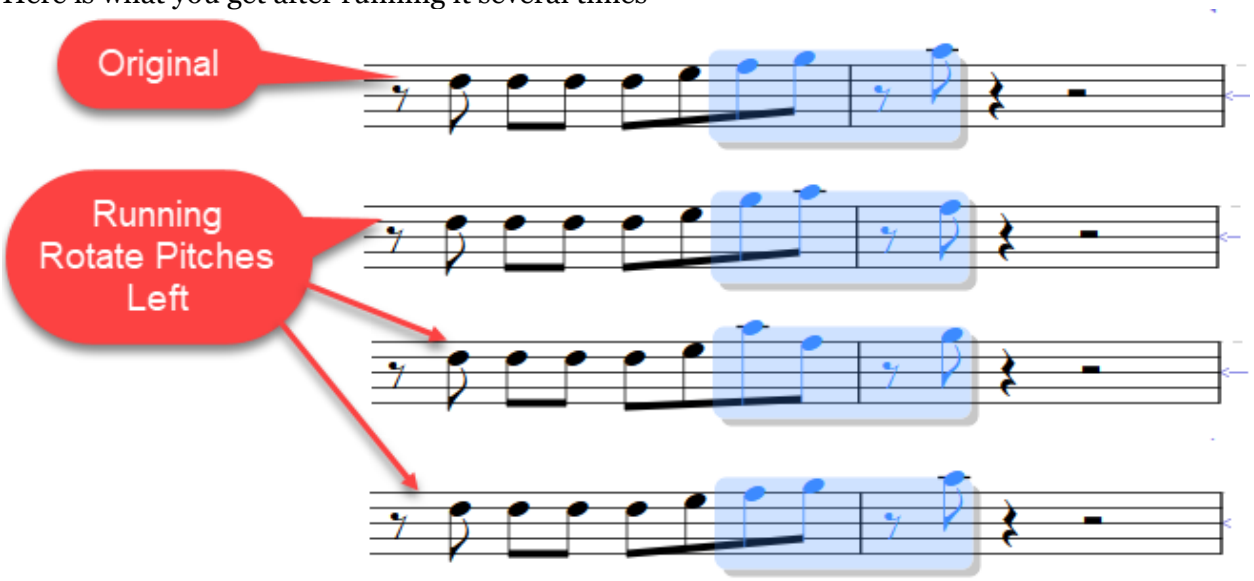
123
312 (first run)
231 (2nd run)

You could write a plugin that will figure out how many "NoteRests" with notes are selected, the use that number to run the plugin you want the appropriate number of times.

One could go to File>Plugins>Edit Plugins and use New to create and install a new plugin, and then in the Run method, replace the MessageBox call with something like this:

```
score = Sibelius.ActiveScore;
selection = score.Selection;
numNRWithNotes = 0;
for each NoteRest nr in selection
{
   if (nr.NoteCount != 0)
   {
      numNRWithNotes = numNRWithNotes + 1;
   }
}

// Simulate left rotate by rotating right (numNRWithNotes - 1) times

score.Redraw = False;

strPluginToRun = "RotatePitches";

for i = 0 to (numNRWithNotes - 1)
{
   @strPluginToRun.Run();
}

score.Redraw = True;
```

Here is what you get after running it several times



If we call F 1, G 2, and A 3, the sequence we see is

123
231
312
123

just as we wanted.

So what is this doing?

The first block is getting the selection and counting up how many notes/chords there are, skipping over rests.

The part starting with // Simulate left rotate by rotating right (numNRWithNotes - 1) times

- Sets score.Redraw false to prevent every changed note from redrawing onscreen.
- Calls the Run() routine of RotatePitches the appropriate number of times.

One bit of weirdness.

I could have said RotatePitches.Run(); instead of @strPluginToRun.Run();
and gotten the same result. To call a plugin you use its filename, with no path and no extension, followed by a period, followed by a method to call. Run() is the method that gets called when a plugin is run from a menu, so it is appropriate here.

There are several Rotate plugins, though, and just to make things easier if I wanted to make a plugin for one of the other Rotates, I isolated the file name from the rest of the code

strPluginToRun = "RotatePitches";

Now I could change that line to

strPluginToRun = "RotateRhythmsAndPitches";

and not have to change anything else to change the plugin that is called. I can't just use

strPluginToRun.Run();, though. I need to "indirect" the variable strPluginToRun so if functions the same way as the plugin name would in RotatePitches.Run();

It is weird stuff that does not come up often, but it is useful at times. You can read up on it in the ManuScript Reference section "Indirection, sparse arrays and user properties". But for now, you can just accept that it will work.