

The String Representation of Musical Phrases in μO

Stéphane Rollandin
hepta@zogotounga.net

draft - 27 January 2012

Abstract

While `MusicalNotes` objects can be composed into a `MusicalPhrase` object by evaluating plain Smalltalk code, it is convenient to have a compact string representation for fast input of more or less complex motives. We present here the main format for such a representation, in effect an expressive domain-specific language. It is extensible in the sense that its vocabulary dynamically reflects specifically tagged selectors, so that it can be enriched by implementing new methods in `MusicalPhrase`, `MusicalNote` and `RhythmicCell`.

1. Creation of musical notes and phrases

The E musical note can be instantiated in μO by the Smalltalk code `'e' knote`. This already uses the string representation we are describing in this paper. `'e'` is the string and the `knote` selector invokes a parser to read this string as a "key note"¹. Similarly, a musical phrase such as the major chord can be instantiated by the code `'c e g' kphrase`.

It would be cumbersome to create a new E note without using the string parser. This would look like :

```
MusicalNote new midiPitch: 64  
or  
Mode major mediant  
or  
MusicalNote new pitch: 329.628
```

The major chord could be created by

```
Mode major I  
or  
MusicalNote new  
  chordCollect: [:n : mp |  
    MusicalNote new midiPitch: mp]  
  with: #(60 64 67)
```

which is not really readable.

These examples illustrate that a note pitch has to come from somewhere : it is either provided as a MIDI key

¹ The KeyKit format designed by Tim Thomson is a subset of the language implemented by the parser, so `knote` can also be taken as "KeyKit note". See <http://nosuch.com/keykit/>

number, or crudely given as a frequency in Hz, or come from the associated scale of a specific musical mode.

In μO it is the responsibility of a mode to name pitches .

2. Notes and modes

The first job of the string parser is to read a pitch via a note name. The way it does this is more sophisticated than simply mapping a symbol to a frequency ; it involves a musical mode, by default `Mode major`.

The main work here is performed by the `readNote:` method of class `Mode`. For example,

```
Mode major readNote: 'e'  
and  
Mode major perfectFifths readNote: 'e'
```

do not return the same note. The `#perfectFifths` selector set the mode scale to a 12-TET a bit larger than one octave, so that it has a just perfect fifth interval; since the default tuning of 440 Hz for A is unchanged, the pitch for E is 329.3617 where it was 329.628 in the standard major mode.

Some modes provide a list of note names mapping their underlying scale steps². This is a rather complex matter in the case of diatonic modes of a chromatic scale in which there is some support for enharmonic notation, French solfège syllables, and Indian sargam notation. We will not go into details about this here.

Whatever the mode however, even if it does not have a naming scheme, it is always possible to refer to the notes directly via their step numbers in the scale, offset by 1 so that the tonic is at step 1. Thus, the major chord can also be instantiated by `'1 3 5' kphrase`.

3. Representation of a single note

A note pitch name can be completed by one or more accidentals and an octave number. The accidentals symbols are + and - (or, equivalently, # and b) ; they transpose the note respectively one scale step up and one scale step down, so that `'e+' knote` is equivalent to

² In μO a mode is build upon a scale by selecting a subset of its steps

'f' knote while 'c+' knote is 'd-' knote. Accidentals can be mixed freely : 'c+-+' knote is a C#.

Quarter-tone inflection is supported via symbols > and < (respectively up and down) so that 'c>>' knote is the same as 'c+' knote, while 'e>' knote is equivalent to 'f<' knote.

It is legal to write something like 'a>-<+>' knote, which is 'b->' knote.

The octave number, if present, is the first modifier to appear. It can be a plain number, or be indicated with a o. The octave number corresponding to the scale index 0 is, by convention, always 3. When no octave number is specified, 3 is assumed : 'c+o4' knote is one octave above 'c+' knote, and can be spelled 'c+4'.

Last but not least, the note pitch can be offset by any interval or just ratio, specified within parentheses. For example 'c(m3)' knote is a minor third above C, that is Eb. 'c(-m3)' knote is a minor third below C, equivalent to 'ao2' knote. Just intervals are allowed, for example in 'c(h7)' knote which is an harmonic seventh above C. The list of supported interval names (several hundreds accounting synonyms) can be accessed by inspecting `MusicTheory.intervalsDictionary`. Intervals can also be specified directly as ratios : 'c(6/5)' knote.

An interval can appear by itself, like in '(m2)' knote. In this case, the pitch reference is the mode tonic, C in the example. But `Mode D minor readPhrase` : '(U)' returns a D (U stands for unisson).

Here ends the overview of the different ways to describe a note pitch.

A note without pitch, a rest, can be defined using r instead of a pitch specification.

The other note modifiers can appear in any order :

v	and a volume
d	and a note length
r	and a note value
c	and a channel
t	and a starting time
D	and a duration
A	and an articulation factor

Volume is either an integer from 0 to 127, or a float from 0.0 to 1.0. Alternatively it can be a dynamics keyword from :pppp to :ffff.

Note length, duration and starting time are either integers expressing time in clicks (there is 192 clicks per second) or floats expressing time in seconds ; note length is always positive.

Channel must be a positive integer.

Note value is an integer, a code number detailed below within the rhythmic canvas discussion.

The articulation factor is a positive number.

The default values are 0.5 for note length, 0.5 for volume (:mp dynamics), 0 for starting time, 1 for channel.

'c' fully explicited would then be 'ct0d0.5v0.5c1'.

Note value is an alternative way to specify the note length ; on the other hand duration is a completely different thing.

Any musical element in μO has a duration used as reference for scaling, mixing and concatenation. Unless explicitly set, the duration of a musical element is equal to its end time. Giving a note a specific duration dissociates the note duration from its end time, allowing articulation control. A D modifier not followed by a numeric specification will make the note duration be equivalent to its end time again. In terms of articulation this is a return to legato.

The articulation modifier A provides a handy way to set both note length and duration in order to achieve a specific articulation : A1 is a legato, A0.7 a staccato, A1.8 a fermata.

Volume, note length, duration and starting time modifiers accept relative numeric arguments : instead of an absolute number, an offset can be provided, which must begins with a + or a -. For example, 'cv+5' knote is equivalent to 'cv68' knote, and 'cv-5' knote is 'cv58' knote. As a special case, a relative argument for starting time must always begins with a +, even if it is negative, since a starting time can be negative. This way the parser can differentiate between 'ct-2.0', a C note starting at time -2 seconds, and 'ct+-2.0', a C note starting two seconds before the default time.

Relative arguments are not really useful when describing single notes, but they are in the context of a musical phrase (see below).

More relative modifiers for volume and note length are defined as single characters. They are :

&	double note length
!	halve note length
.	scale note length by 3/2 (dot)
_	scale note length by 2/3 (triplet)
*	scale note length by 3
?	scale note length by 1/3
^	dynamics up (e.g. from :mf to :f)
"	dynamics down (e.g. from :mf to :mp)

4. Notes in a musical phrase

Within a musical phrase string representation, the default values for absent note modifiers are determined by the previous note :

```
'cd0.2 e g' kphrase
```

is equivalent to

```
'cd0.2 ed0.2 gd0.2' kphrase
```

This applies to the pitch, so a major chord could also be expressed as list of intervals :

```
'(U) (M3) (m3)' kphrase
```

Here is its just intonation version :

```
'(U) (5/4) (6/5)' kphrase
```

The interest of relative arguments now appears :

```
'c ed+0.1 gd+0.1' kphrase
```

is equivalent to

```
'c ed0.6 gd0.7' kphrase
```

Notes are separated by commas or white space ; the type of separator determines the default starting time of the following note : if it is a white space, the above rule applies and it is the same as the previous note starting time. If the separator is a comma then the default starting time is the duration of the previous note which in most cases is its end time :

```
'c,e,g' kphrase
```

is an arpeggio, equivalent to

```
'c et0.5 gt1.0' kphrase
```

The note duration may not be its end time, for example when using the articulation modifier A :

```
'cA0.8,e,g' kphrase
```

defines a staccato equivalent to

```
'cd0.4 et0.5 gt1.0 l1.5' kphrase
```

In the presence of a rhythmic canvas (see below), multiple commas can be used to skip beats, otherwise they count as a single comma :

```
'c,,e , , g' kphrase
```

is the same as

```
'c,e,g' kphrase
```

For the sake of clarity carriage return, linefeed and tabulation characters act as white space.

5. Phrase duration

By default the duration of a musical phrase is equal to its length, that is the largest of the ending times of its notes. The duration can be explicitly set by using an **l** (or a **L**) followed by the duration.

A standalone **l** (or **L**) can alternatively appear anywhere among the notes in which case the duration of the phrase will be set to the default starting time at that position.

Note that 'c,e,g,l' kphrase is not exactly equivalent to 'c,e,g' kphrase : both do have the same duration (1.5 seconds) but the first one has an explicit duration while the second one does not :

'c,e,g' kphrase solidDuration	is	nil
'c,e,g,l' kphrase solidDuration	is	1.5

6. Chords and arpeggios

A single note is expanded into a chord or an arpeggio with the **:** and **::** modifiers. For example **c:maj** or **d::h7**. Unless explicitly cancelled, the modifier applies to the following notes :

```
c:maj, e, g
```

stands for

```
c:maj, e:maj, g:maj
```

To cancel simply use **:** (resp. **::**) without a chord name.

The list of implemented chords is available by inspecting or printing `MusicTheory knownChords`. Many chords have synonyms ; to see them inspect or print `MusicTheory knownChordsWithSynonyms`. Any chord name may be prepended with an underscore **_** : this indicate a negative chords, where intervals are stacked downwards³.

A chord name may be completed by a bass note and a voicing. Both are separated by a **/** .

The bass note, when present, should appear first. If the note is present in the chord, it will be transposed one octave down ; if it is not it will be added.

³ From jazz musician Steve Coleman ; see its essay at http://m-base.com/symmetrical_movement.html

The voicing is a code, a combination of :

s	skip (move an octave up)
d	drop (move an octave down)
o	double an octave up
b	double an octave down
m	omit

each followed by an integer, the index of a note in the chord, bass included, 1 being the index for the lowest note.

Examples :

```
'c:maj/A/o1' kphrase
is
'ao2 co3 e g a' kphrase

'c:7sus4/o1s2' kphrase
is
'c g b- co4 f' kphrase
```

7. Rhythmic canvas

At any position between notes in the string representation of a musical phrase it is possible to insert a time signature.

[...] defines a rhythmic canvas⁴ upon which the following notes will be read: when not explicitly given a time or length attribute, they are aligned in the canvas: each comma , note separator moves the reading time to the next beat in the rhythm. The semicolon ; note separators works as usual, setting the reading time to the duration of the previous note.

Within a rhythmic canvas, a note duration can be specified as a note value using the *r* attribute: *r0* for a double whole note, *r1* for a whole note, *r2* for a half-note, etc down to *r128*.

Dotted and triplet notes can be specified by appending a 0 (resp. a 3) to a note value: *r80* is a dotted eighth, *r43* a triplet quarter. Appending a 6 stands for a double triplet note (with the very special case of *r106* for a double triplet whole note, *r16* being a plain sixteenth).

A *r* alone, with no numeric argument, will stretch the note up to the next beat in the rhythm.

After a comma , special operators can be used to jump to a specific following beat:

	next downbeat (in most cases the beginning of next measure)
--	--

⁴ For details about rhythmic cells and canvases, see "The Representation of Rhythmic Structures in μO "

S	next on beat that is not a downbeat
s	next on beat, possibly a downbeat
w	next off beat, possibly void
W	next off beat that is not void
v or V	next void beat
b	the next beat, whatever it is

These operators have no effect if the current beat already fits the specification.

The content of [...] may be:

- nothing, [], for free rhythm
- a key from the `RhythmicCell` library
e.g. [4/4] or [3/8]
- a `RhythmicCanvas` signature,
e.g. (1 ((2 2) 4) 2 (3 8))]
- a `RhythmicCell` signature,
e.g. [((2 2) 4)] or [(M ((2 2) 4) 2 (3 8))]
- a `MusicalPhrase`, in which case it will be considered as a rhythmic cell template.
- if a nested phrase has been stored in variable *var*, it can be used via [*@var*] (see below for nested phrases).

8. Nested phrases

A {...} section can be used to define a part in a phrase as a nested relative phrase.

What this means is that all the text within brackets is parsed first as a full phrase, then inserted at the current read time. The nested phrase is relative in the sense that upon parsing it inherits the current notes modifiers and the current rhythmic canvas (if any).

The nested phrase can be stored in a variable *var* by using the syntax {...}@*var*

A subsequent relative phrase with syntax {@*var*} or {/.../ @*var*} can then recall it.

The last nested phrase can always be recalled by a plain{} or {/.../}.

In the context of a rhythmic canvas, the exact way a relative phrase is inserted is controlled by one or more optional symbols:

{...} (no symbol) parallel insertion within a rhythmic pattern: the nested phrase may get ahead the current beat.

{...}! overwriting insertion within a rhythmic pattern: the nested phrase claims as many beats as required to fit.

`{...}#` force alignment of the relative phrase to the rhythmic pattern: the temporal structure of the nested phrase is reconstructed so that it follows the current rhythm.

`{...}##` force alignment in a similar manner as above, but also take care that no rest happens between notes.

`{...}1` force scaling of the relative phrase into the current beat

`{...}n` force scaling into n (an integer) beats

Other symbols defines specific behaviors :

`{...}%` the relative phrase defines the new rhythmic cell

`{...}%n` ... from its first n notes only

`{...}%-n` ... from its last n notes only

`{...}>` keep attributes: all notes attributes (except time) are passed back to the parent phrase.

`{...}~style` insert the relative phrase as a single inflected note, where *style* (optional) defines the pitch envelope shape; see below for more about inflected notes.

Several symbols can be used; they must appear in the following order : 1..9 @ ! # @ % > ~

9. Pluggable modifiers

What has been described so far could be considered as the static part of the string format for musical phrases and notes. Technically, it is mostly implemented in classes `MusicalPhrasePrinter` and `RelativeMusicalPhrase`.

The dynamic (or pluggable) part of the format is a three-fold system of routing allowing to modify either a whole phrase, a single note or the current rhythmic cell by directly sending it Smalltalk messages.

For example, `'c|cut| e g'` kphrase is a major chord where all notes have been sent the message `cutAttack`.

The implementation of method `#cutAttack:` in class `MusicalNote` features the pragma

```
<muOPluggableAPI:
  #(Operation
    category: 'expression'
    keyword: 'cut')
    parameters:
      #((Number positive default: 0.1))>
```

Because of this pragma (a meta-statement not interpreted as Smalltalk code) the parser knows that when it finds a `cut` note modifier it has to send the message

`cutAttack:` with argument 0.1 to the note⁵.

The code `'c|cut0.2| e g'` kphrase would use argument 0.2 instead of the default value.

As another example the phrase `'a,b,c'` kphrase can be reversed by writing `'/rev/ a,b,c'` kphrase.

Again this is because the `#reverse` method in class `MusicalCollection` (parent of `MusicalPhrase`) includes pragma

```
<muOPluggableAPI:
  #(Operation category: 'basics'
    keyword: 'rev')>
```

which binds the `rev` keyword to selector `#reverse`.

Several modifiers can be chained, separated by semicolumns: `'/r5;acc/ c,e,g'` kphrase repeats notes C,E,G five times then shape the notes onsets into a *accelerando*.

A note about arguments: possibly optional, they are always numeric. Floats and integers, positive or negative, can be written as is. Fraction must be written in the form $n:m$ instead of n/m .

In summary, modifiers provide a way to inject Smalltalk code into a phrase representation. Although this is limited to specific tagged methods of at most one argument, it is infinitely extensible: the composer familiar enough with Smalltalk can very easily implement its own methods, enriching the vocabulary of modifiers according to his style.

9.1. Note modifiers

Single note modifiers must appear in a `|...|` section immediately following the note specification.

The library of available modifiers is visible by inspecting `MusicalNote modifiers`.

Like most other note modifiers, the `|...|` section is passed implicitly from a note to the following one. The propagation is stopped by any new section, including the empty one `||`.

A `|...|` section is allowed after a chord or arpeggio definition. In that case the modifiers will apply to the expanded note, a musical phrase. The corresponding library is returned by `MusicalPhrase modifiers`.

For example,

⁵ This registration mechanism has other usages in `μO`. Notably it allows extensible menus in interactive editors.

```
'c:maj|strum|,a,b' kphrase
```

has the method `strum` sent to the three major chords.

To have only the first two chords strummed, we can write

```
'c:maj|strum|,a,b|' kphrase
```

Modifiers specified with syntax `|+...|` are local for a single note: they are not propagated; for example,

```
'c,e|+cut|,g' kphrase
```

only cuts the attack of note E.

If note modifiers are already defined, a new set of modifier specified with `|+...|` is added to them for the corresponding note; syntax `|-...|` is similar, with the difference that the local modifiers are prepended to the previously defined ones.

Finally, modifiers can be changed for all following notes using forms `|+=...|` and `|-...|`, which work as described above with the only difference that their effect is not local.

9.2. Phrase modifiers

Full phrase modifiers must appear in a `/.../` section at the beginning of the phrase representation.

Several sections can appear consecutively. In that case, the sections are considered from right to left, the rationale being that it makes it possible to further process a musical phrase by simply appending a modifier section to its string representation.

Phrase modifiers are a definite part of the string format ; as such, they can appear within nested phrases :

```
 '/r2/ a,b, {/r3/ c, d}' kphrase
```

is

```
'a,b,c,d,c,d,c,d,a,b,c,d,c,d,c,d' kphrase
```

The library of available modifiers is returned by the expression `MusicalPhrase modifiers`.

9.3. Rhythmic cell modifiers

Rhythmic cell modifiers change the current cell ; they can do so by either replacing it altogether or by first creating a child cell, in which case it is later possible to switch back to the parent cell.

The modifiers appear in a `[;...]` section. For example,

```
[3/8][;div;bpm160]
```

is a 6/8 time signature

A child cell is created with `[:]` while the return to the parent cell is done via `[.]` ; to create a new child of the current parent, use `[.:]`.

In summary we have the forms :

<code>[;...]</code>	modify current cell
<code>[:...]</code>	create a child cell, modify it
<code>[.:...]</code>	back to parent cell, modify it
<code>[.::...]</code>	create a child of parent, modify it

10. Inflected notes

An inflected note is a note whose pitch is not constant over its duration. In the μO implementation such a pitch is represented by an envelope.

The envelope can be either read from a library :

```
c~up
```

or it can be built by connecting notes in a phrase :

```
{c,g,e}~
```

10.1. Pitch envelope structure

To do

10.2. Pitch envelope libraries

To do

```
c~up*3
```

10.3. Musical phrase as inflected note

To do

```
{c,g,e}~smooth
```

```
{/stacc/ c,g,e}~smooth
```

11. MIDI drums

To do

12. MIDI messages

To do

13. Mode changes

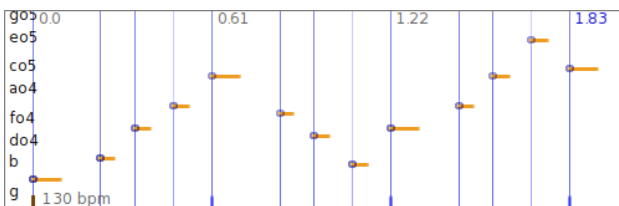
To do

14. Examples

14.1. A modal exploration

Here is a real-world example of musical phrase defined by a complex string representation. It looks obfuscated, but one must keep in mind that it was elaborated incrementally with real-time feedback in a interactive notebook: actually it came very naturally.

```
M[E harmonicMinor]
[(M 1 (1 8) 1 (1 16) 1 (2 14))] [;bpm130]
{/art0.4/
  {4|mf;ninth;ap;dim;dim|,
    1|+rev;fn3|,
    4|+ln3|,
    7|+rev;fn2|}##}
```



The compactness of the format is better appreciated when one look at the equivalent Smalltalk code, albeit very concise in itself:

```
mode := Mode E harmonicMinor.
rcanvas := (#(M 1 (1 8) 1 (1 16) 1 (2 14)) sig
            bpm: 130) asCanvas.
phrase :=
  (mode subdominant mf ninth
    arpeggiate diminuendo diminuendo),
  (mode tonic mf ninth arpeggiate diminuendo
    diminuendo reverse firstNotes: 3),
  (mode subdominant mf ninth arpeggiate diminuendo
```

```
diminuendo lastNotes: 3) at0,
  (mode subtonic mf ninth arpeggiate diminuendo
    diminuendo reverse firstNotes: 2).
phrase followCanvas: rcanvas.
phrase staccato: 0.4; removeSolidDuration.
```

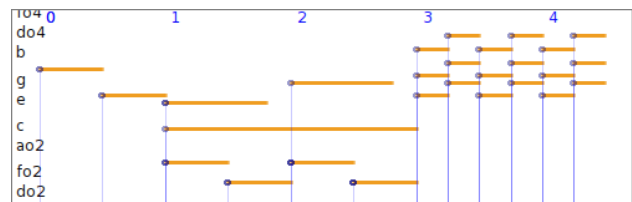
Of course it is also possible to mix Smalltalk code and string representation:

```
phrase := Mode E harmonicMinor readPhrase:
  '4|ninth;ap;dim;dim|,1|+rev|,4,7|+rev|'.
phrase followCanvas:
  (#(M 1 (1 8) 1 (1 16) 1 (2 14)) sig
    bpm: 130) asCanvas.
phrase staccato: 0.4; removeSolidDuration.
```

14.2. Nested phrases

Here we use three simultaneous nested phrases for a complex chord structure (including dynamics variations), followed by another nested phrase repeating two chords three times:

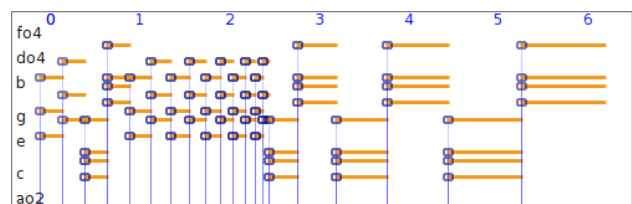
```
a,f,{c"&&}{go2^,e,g^,e}{/nonleg/ e^&,g""},
{/r3/ f!" :min,g}
```



14.3. hi & ho

This example demonstrates how nested phrases can be bound to an symbol and recalled later.

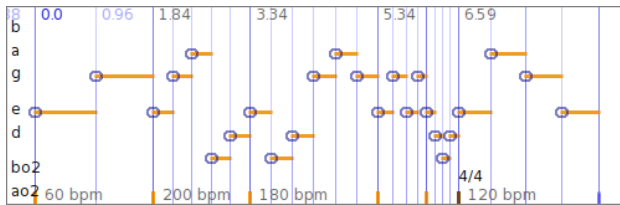
```
{f!" :min,g}@hi,{c!" :min2,a}@ho,
{/mf;r5;acc4;cresc/ @hi}, {/mf;r3;rit4;dim/ @ho}
```



14.4. Rhythmic variations

In this example we start with a plain 4/4 signature and play with it via rhythmic cell modifiers:

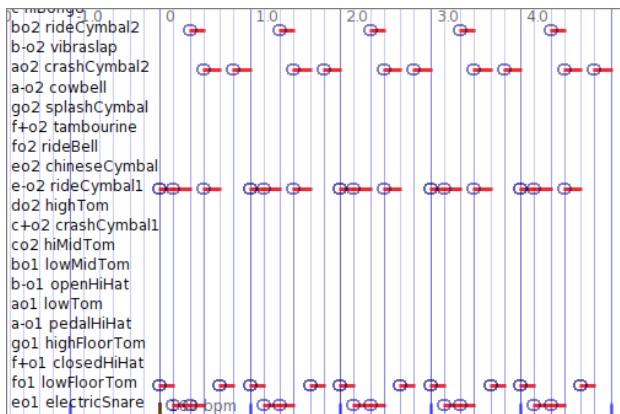
```
[4/4] [:tup2:4;acc] e,g, [:tup5:3] e,g,a,c,d,
[:tup6:4] e,c,d,g,a,g, [:tup4:2;acc2] e,g,e,g,
[:tup4:1] e,d,c,d, [:rit] e,a,g,e
```



14.5. Drumming

6/6 drumming with some swing:

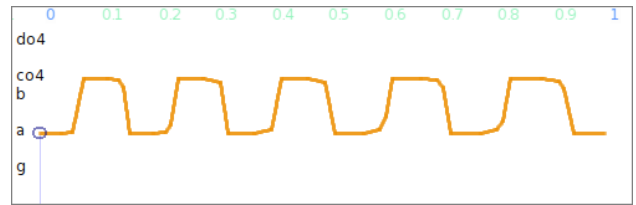
```
/r5/ [3/6][;dsw0.45] +(loFTom)^ +(riCy1), +(riCy1)"
+(eSnare), +(eSnare) +(riCy2), +(riCy1) +(crshCy2),
+(loFTom)^, +(crshCy2)"
```



14.6. A vibrato

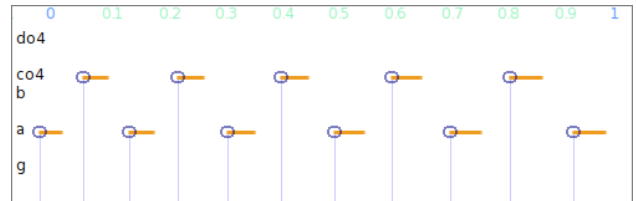
Pitch inflexion can be sculpted in very precise way; in this example we define all details of a vibrato:

```
{/d1.0/ {/r5.5;stacc;rit1.5/ a,b+}}~slur
```



The vibrato skeleton is the part within brackets:

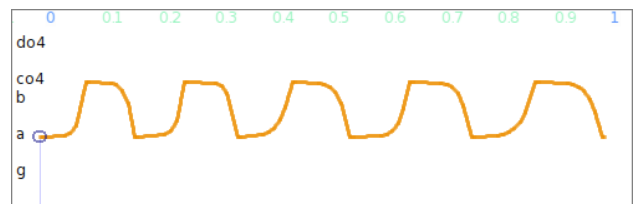
```
/d1.0/ {/r5.5;stacc;rit1.5/ a,b+}
```



Over the course of 1 second we repeat 5.5 times the sequence A C with a slight ritardando. The staccato articulation defines which proportion of the notes will have a stable pitch in the eventual vibrato,

See for example with a much shorter staccato:

```
{/d1.0/ {/r5.5;dots;rit1.5/ a,b+}}~slur
```



15. Inline documentation

Notebook, cheat sheets

Test cases