The Representation of Rhythmic Structures in µO

Stéphane Rollandin
hepta@zogotounga.net

draft - 14 November 2013

Abstract

We discuss the metaphors used in µO to represent rhythm in its various aspects. While rhythm is an implicit part of any `MusicalCollection` where it is defined by the notes onsets, it can be made explicit in specific classes, namely `RhythmicCell` and `RhythmicCanvas`.

Notation

In the following, the printed evaluation of a Smalltalk expression is represented following a ▶ symbol. When a graphic representation is available (a screenshot of a µO editor in most cases), it is displayed after a ▶. All code is written in `Consolas font`.

1. `MusicalCollection` and `RhythmicCell`

In µO every subclass of the `MusicalCollection` abstract class is an ordered set of note-like musical elements; such a subclass is `MusicalPhrase`, a phrase of `MusicalNotes`, which usage is discussed at length in another paper[1]. In a musical collection the rhythm of notes is fully accessible by querying for their onsets and durations.

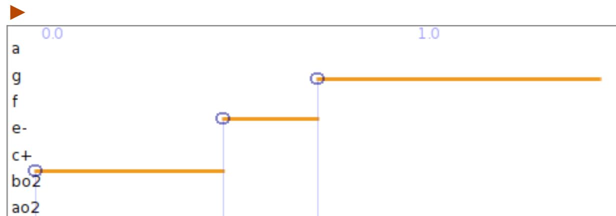
When working on the different rhythmic aspects of a musical composition however, it is very convenient to use representations of rhythm dissociated from any motivic instanciation, in other words views of rhythm by itself , defined purely as a structuration of time.

In µO the fundamental pulsating aspect of musical time is reified in class `RhythmicCell`, while more metrically elaborated rhythmic structures can be represented by an instance of class `RhythmicCanvas`, itself composed of one or more cells.

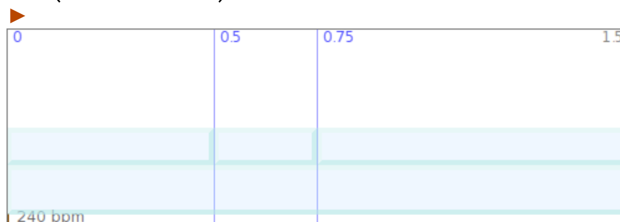
We can instanciate a `RhythmicCell` from any `MusicalCollection` by sending it the message `#asRhythmicCell`; what we will obtain is an object basically representing the notes onsets and some information about their amplitudes.

1 See "The String Representation of Musical Phrases in µO".

```
'c,e!,g&.' kphrase
▶ 'c,ed48,gd144'
▶
```




```
'c,e!,g&.' kphrase asRhythmicCell
▶ R(T0 T0.5 T0.75)D1.5
▶
```



2. Beat strengths

`RhythmicCell` is itself a musical collection, whose notes are the cell beats. Four different beat accents are defined: `strong`, `weak`, `strongest` and `void`. They are related to notes amplitude, although they can be used arbitrarily.

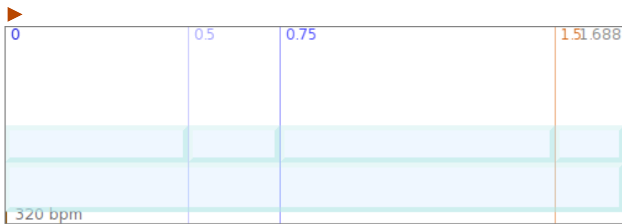
A downbeat is `strongest`. On-beats are `strong`, off-beats are `weak`. Void beats are `void`; while there is no defined meaning for a void beat in western music, in northern classical indian music it would be a khali[2].

When obtaining a `RhythmicCell` from a `MusicalCollection` the notes amplitudes will translate into the corresponding beats accents:
- an amplitude of 0 gives a `void` beat.
- an amplitude below 0.5 gives a `weak` beat.
- an amplitude above or equal to 0.5 gives a `strong` beat.
- an amplitude of 1 gives a `strongest` beat.

In the string representation of a cell, a `void` beat is marked as `V`, a `weak` beat as `t`, a strong `beat` as `T` and a `strongest` beat as `S`:

```
'cv1.0,ev0.3!,gv0.5&.,cv0!!' kphrase asRhythmicCell
▶ R(S0 t0.5 T0.75 v1.5)D1.69
```

2 http://chandrakantha.com/articles/indian_music/khali.html



A beat by itself is an instance of class `Tick`; any musical note can be converted into a beat:

```
'cv0.7' knote asTick accent
▶ #strong
```

3. `RhythmicCell` as a time signature

A `RhythmicCell` is actually the reification of the musical concept of time signature.

A specific format allows the definition of a rhythmic cell via an `Array` specification very close to the standard way to write a time signature.

The array has the form `#(b v)` for a `b`/`v` signature, where `v` is the beat note value and `b` the number of beats; `b` can itself be decomposed into an array `(b1 b2 ...)` having the actual `b` decomposed in `b1+b2+...` for an additive meter where each segment starts with a strong beat.

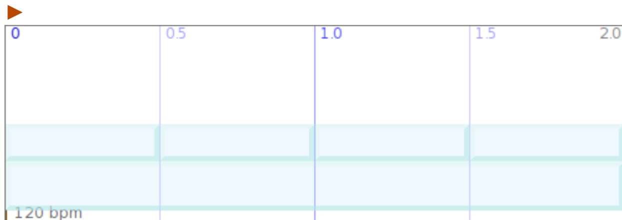
The first beat is always a downbeat, except if `b` is negative in which case it is void; negative values are also allowed in the segmented specification.

The note value `v` is an integer, 4 for a quarter note (crotchet) , 8 for an eighth note (quaver), etc.

Sending `#sig` to such an array returns the corresponding rhythmic cell.

For example the usual 4/4 signature is

```
#((2 2) 4) sig
▶ R(S0 t0.5 T1.0 t1.5)D2.0
```



where the upper 4 is written in the additive form `(2 2)` instead of a plain 4 so that the third beat is made strong.

Because a time signature is a first-class musical element in µO (a subclass of `MusicalCollection`), it has an actual extension in time; consequently its tempo is well-defined and can be changed by the regular scaling operators. See below for more about tempo.

More surprisingly maybe, a time signature also has a starting time. This makes sense in the rhythmic canvas framework which is discussed below, where the starting time specifies when a time signature is to be applied and replace the previous one.

4. Tempo

A `RhythmicCell` knows about its tempo by maintaining its own note values. Sending it `#quarter` or `#crotchet` returns the length (in seconds) of a quarter note for that cell.

```
RhythmicCell new quarter
▶ 0.5
```

The cell also knows what note value is considered to define the beat:

```
#(4 4) sig beat
▶ 0.5

#(4 4) sig beatValue
▶ #quarter

#(4 8) sig beat
▶ 0.25

#(4 8) sig beatValue
▶ #eighth
```

The cell tempo can be changed by any operation scaling a `MusicalElement` or more specifically by directly setting the BPM (beats per minute) value:

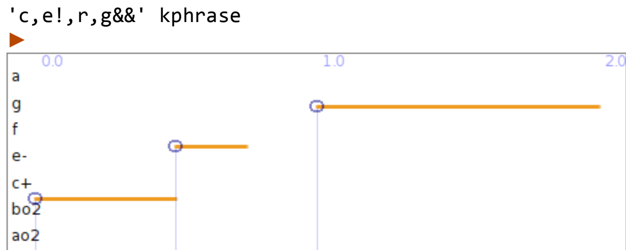
```
cell := #(4 4) sig.
cell bpm
cell bpm: 200

cell beat
▶ 0.3

cell beatValue
▶ #quarter

cell quarter
▶ 0.3

cell bpm
▶ 200
```

5. `RhythmicCell` as a motivic rhythm

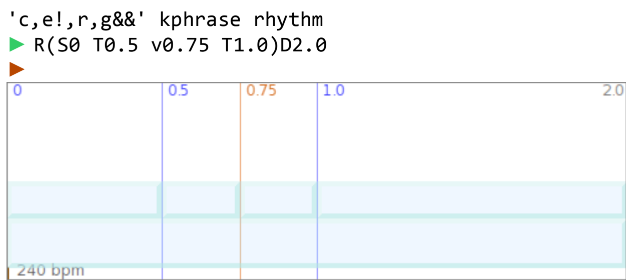
A `RhythmicCell` can represent the rhythm of a musical motif. In that case each note defines a beat, each rest define a void beat.

Sending `#rhythm` to a musical phrase[3] returns its rhythm:

```
'c,e!,r,g&&' kphrase
```

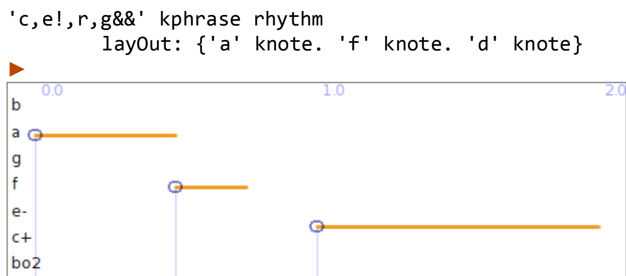



```
'c,e!,r,g&&' kphrase rhythm
▶ R(S0 T0.5 v0.75 T1.0)D2.0
```



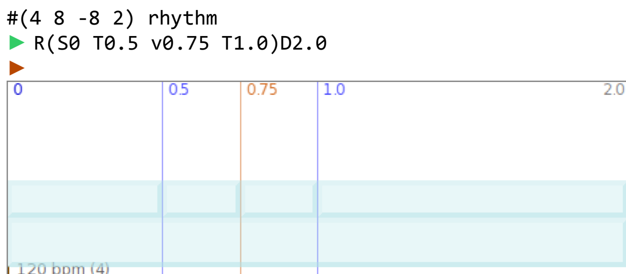
It is then possible to give this rhythm to another phrase:

```
'c,e!,r,g&&' kphrase rhythm
        layOut: {'a' knote. 'f' knote. 'd' knote}
```



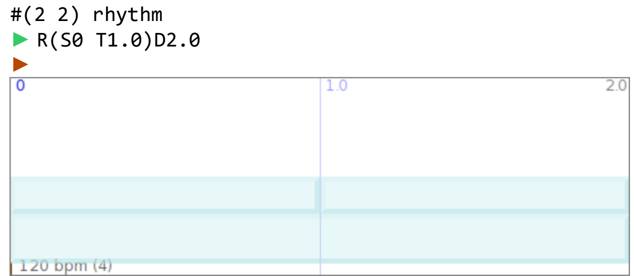
A rhythm can be defined from scratch using an array format similar to the one used for time signatures and rhythmic canvases (see above). Here the array simply contains the list of note values making up the rhythm, a negative value marking a void beat.

```
#(4 8 -8 2) rhythm
▶ R(S0 T0.5 v0.75 T1.0)D2.0
```



3 Actually, to any `MusicalCollection` subclasses instance

```
#(2 2) rhythm
▶ R(S0 T1.0)D2.0
```



Here is how a random phrase with a given rhythm could be built:

```
mode := Mode harmonicMinor.
cell := #(4 8 -8 2) rhythm bpm: 140.
phrase := cell layOut: [mode noteAt: 7 atRandom].
```

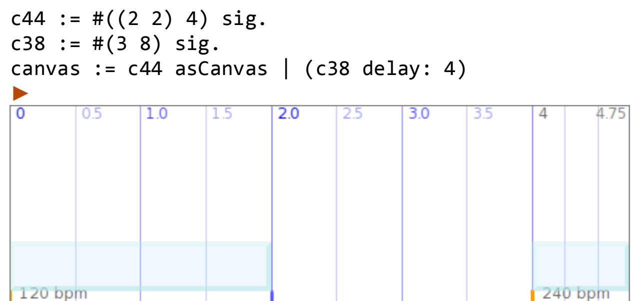
6. `RhythmicCanvas`

A `RhythmicCanvas` is composed of one or several `RhythmicCells`. At any time the effective time signature is set by the latest cell. The time before the first cell in the canvas is structured by that first cell.

A simple canvas based on one cell can be obtained by sending `#asCanvas` to that cell.

For example

```
c44 := #((2 2) 4) sig.
c38 := #(3 8) sig.
canvas := c44 asCanvas | (c38 delay: 4)
```

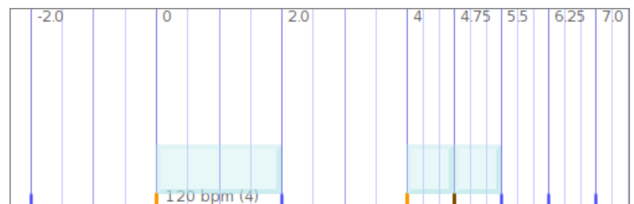


The above example canvas, because it is simple, could be defined directly in a format similar to time signatures:

```
#(2 ((2 2) 4) 1 (3 8)) sig
```

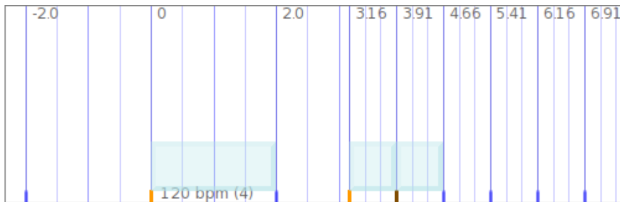
which reads: "take two measures of 4/4 then turn to 3/8".

If we zoom out the editor view above we can better see how the canvas structures time:



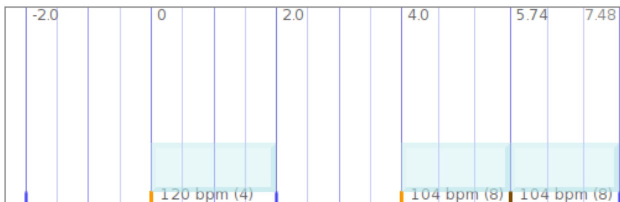
Before time 4 seconds, we are in 4/4. After that time, we are in 3/8. The fact that there are actually two adjacent 3/8 cells in the canvas is an artefact from the way `RhythmicCanvas` implements the `MusicalElement` protocol; this will not be discussed in this paper[4].

Note that the cells making up a canvas can be at arbitrary positions. In the above example if time 3.16 had been choosen instead of 4 we would have had the canvas:



where the second 4/4 cell is interrupted.

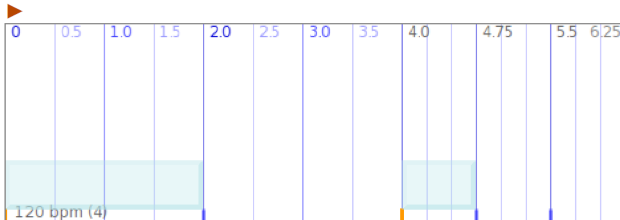
The canvas cells can also have arbitrary tempos: with a 104 bpm tempo for the 3/8 cell, the canvas looks like:



6.1 Rhythmic canvas places

It is easy to get access to the ticks of a rhythmic canvas by using "places". Let's consider the following canvas:

```
canvas := #(2 ((2 2) 4) 3 (3 8)) sig
```

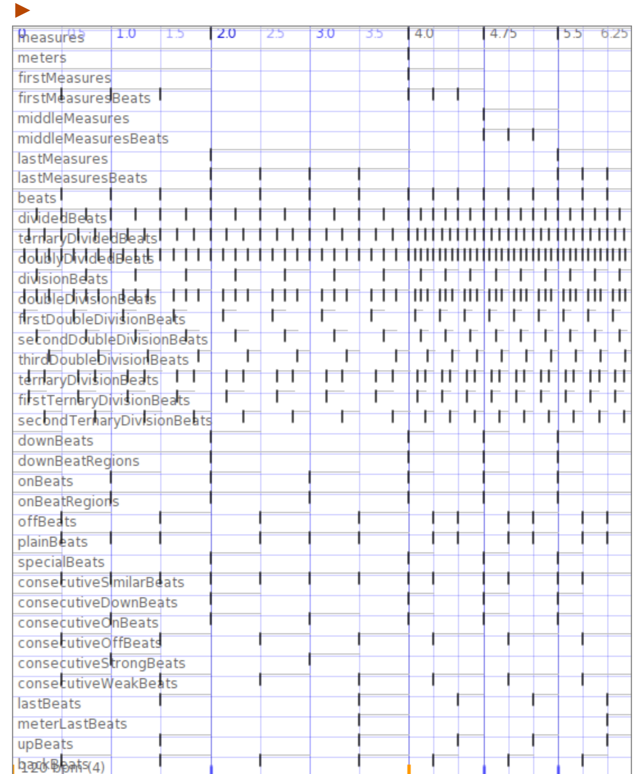


It is made of two measures in 4/4 followed by three measures in 3/8.

Places are symbolic locations within the canvas. For example `#measures` is a place referencing all measures in the canvas; `#downBeats` is a place referencing the first beat of all measures; `#backBeats` is a place referencing all beats right after a on-beat; etc.

Many more places are defined:

```
canvas displayPlaces
```

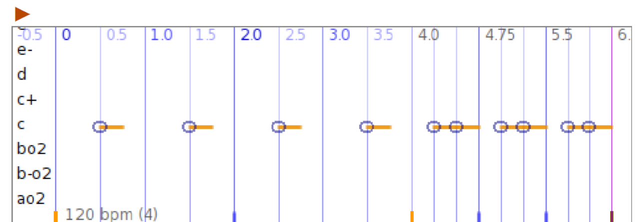


Many ways to iterate over places are implemented. The more generally useful are provided by methods `#on:mix:`, `#on:scaleAndMix:` and `#on:place:`. We detail their usages in the following.

1) `canvas on: somePlace mix: aMusicalElement`

copies `aMusicalElement` for every instance of `somePlace` in `canvas`:

```
canvas on: #offBeats mix: 'c!' knote
```

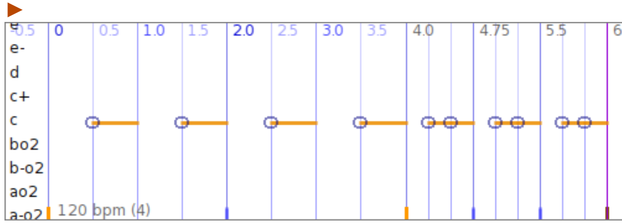


2) `canvas on: somePlace scaleAndMix: aMusicalElement`

copies `aMusicalElement` for every instance of `somePlace` in `canvas`: and scales the copy so that it fits exactly within the corresponding beat.

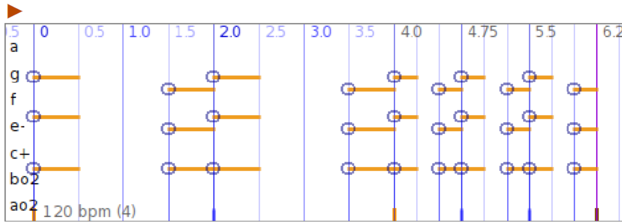
4 In short: because it is a `MusicalElement`, a rhythmic canvas must have a starting time and a settable duration, even though it is in effect infinite in both time directions. For one-cell canvases, starting time and duration are taken from the cell; for many-cells canvases, the starting time comes from the first cell, the duration is the starting time of the last cell.

```
canvas on: #offBeats scaleAndMix: 'c!' knote
```



Here is how one could play a major chord on each downbeat in `canvas`, and a dimished chord on each up beat:

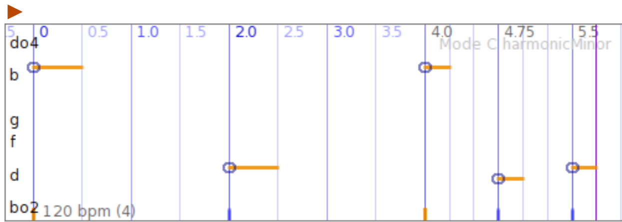
```
ph1 := canvas on: #downBeats
        scaleAndMix: 'c:maj' kphrase.
ph2 := canvas on: #upBeats
        scaleAndMix: 'c:dim' kphrase.
ph1 | ph2
```



In the above 1) and 2) syntaxes, `aMusicalElement` can also be a block, or a collection.

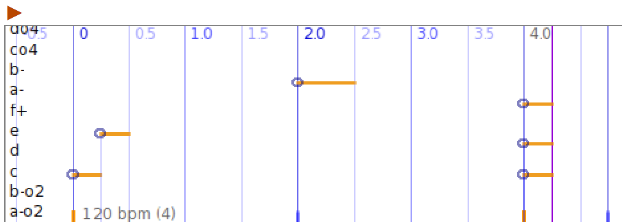
If a block, it should return a `MusicalElement` and it will be evaluated for each occurence of the place:

```
mode := Mode harmonicMinor.
canvas on: #downBeats
        scaleAndMix: [mode noteAt: 7 atRandom]
```



If a collection, its elements are used in turn to populate the corresponding beats, until either the end of the collection or the end time of the canvas is reached:

```
canvas on: #downBeats
        scaleAndMix: {'c,e' kphrase .
                        'a' knote .
                        'c:min' kphrase}
```

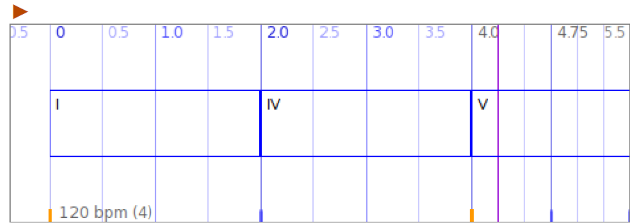


3) `canvas on: somePlace place: something`

is used to create a `BolPhrase`. A `BolPhrase` is a `MusicalCollection` of consecutive arbitrary objects, each one wrapped into a `BolWord`. Originally it has been implemented to represent actual bols, which are syllables used by tabla drummers in Indian classical music, but it has many more usages; basically it allows to structure arbitrary information in a time-wise manner.

We could for example define a chord progression this way:

```
canvas on: #downBeats place: #(I IV V)
```



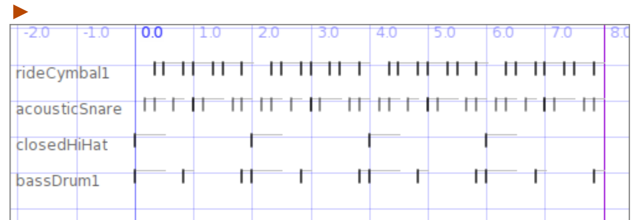
Discussing further details about bol phrases in this paper would lead us astray from its topic, so we will stop here.

6.2 Grooves

`Groove` subclasses implement another way to populate specific canvas places; they define full-fledge drum patterns.

For example `HalfTimeShuffle`:

```
HalfTimeShuffle busy on: #(4 ((2 2) 4)) sig
```



The `Groove` subclass `GrooveOnDemand` implements a domain-specific language allowing a very compact representation of arbitrary grooves:

```
groove := GrooveOnDemand with:
      #((onBeats addLouder: bass)
        (downBeats erase: bass)
        ((beats TDb2) add: rideCymbal)
        (downBeats erase: rideCymbal)
        (downBeats add: hiHat)
        (TDb1 addGhost: snare)
        (TDb2 onMSieve: 2 2 add: bass)
        (onBeats atCounts: 2 add: bass)
        (TDb1 atCounts: 3 erase: snare)).
```

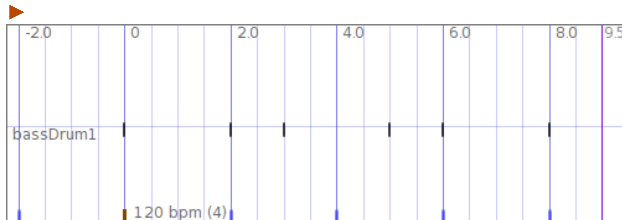
```
groove on: #(4 ((2 2) 4)) sig
```

6.3 String representation of rhythmic patterns

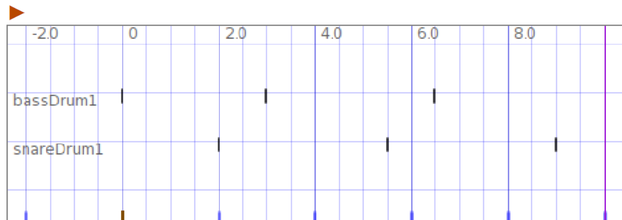
A common way to represent simply a rhythmic pattern is to write it down as a string such as `'o___o_o___o_o___o_'`, where a `_` would stand for a rest and a `o` for a drum stroke.

It is easy to use this kind of notation in muO. We just have to provide the string with a dictionary associating each character with a musical element, and a base rhythm. Any undefined character will be interpreted as a rest:

```
'o___o_o___o_o___o_'
  interpretIn: #(4 4) sig
  with: ({$o -> #bassDrum1 asStroke} as: Dictionary)
```

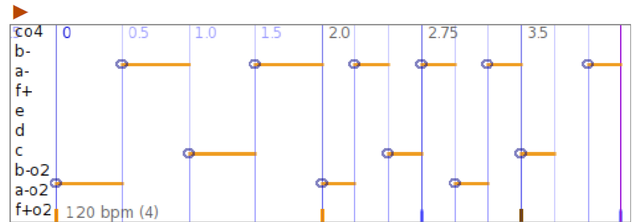


```
'o...x.o....x.o....x.'
  interpretIn: #(4 4) sig
  with: ({$o -> #bassDrum1 asStroke .
         $x -> #snareDrum1 asStroke} as: Dictionary)
```



When the musical elements have to be scaled to the beats lengths, one can use `#interpretIn:scaledWith:`

```
'ohxhohxhohx.h'
  interpretIn: #(1 (4 4) 3 (3 8)) sig
  scaledWith: ({$o -> 'ao2' knote .
               $h -> 'a' knote .
               $x -> 'c' knote} as: Dictionary)
```



7. Metered musical elements

Any `MusicalElement` can be associated with a rhythmic canvas. The resulting object is a `CompositeMix` of the element and the canvas[5].

... to be continued

#withMeter:
#withMeterLayOut:

5 See "Usages of `CompositeMix` in µO"