Hikari

Chlorie

Jul 12, 2023

CONTENTS

A funny way to notate music excerpts.

CHAPTER

ONE

SYNTAX GUIDE

An illustrative guide of the Hikari notation syntax. The score images are generated with Lilypond.

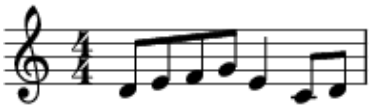
1.1 Character Set

The string should contain only ASCII characters. Spaces and returns are ignored entirely, even when they separate syntactical tokens.

1.2 Notes and Chords

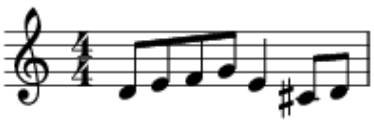
Notes in *Hikari* are represented by *upper-cased* letters A to G.

```
DE,FG,E,CD,
```



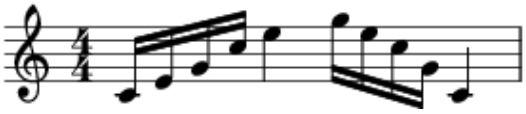
Accidentals of the notes are notated with `#` for sharps, `b` for flats, `x` for double sharps, and `bb` for double flats. The accidentals come *after* note letters.

```
DE,FG,E,C#D,
```



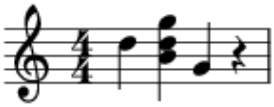
The notes are placed in the 4th octave by default (from the middle C *C4* to the note at a major 7th above it *B4*). To change the octave range, use an integer after a note name to modify the default octave range, or less than/greater than signs < > to temporarily modify the octave range of a single note.

```
CEGC>,E>,
G5ECG4,C,
```



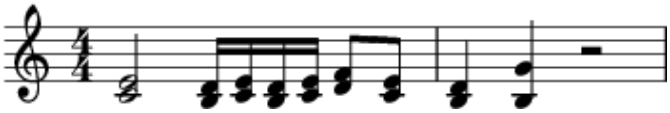
A chord consisting of multiple notes is notated by surrounding the notes in a pair of parentheses `()`, and rests are notated with dots (`.`).

```
D5, (B<DG), G<, .,
```



Prolongations of notes or chords are notated with dashes (`-`).

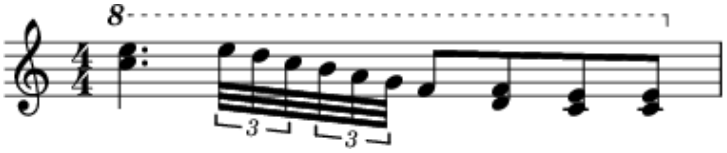
```
(CE),-, (B<D)(CE)(B<D)(CE), (DF)(CE),
(B<D), (B<G), ., .,
```



1.3 Musical Structure

The most basic component of a piece of music is a *beat*. The beats are separated by commas (`,`). The notes (chords, rests) in a single beat divide the beat into parts of equal length. All beats should be terminated by a comma, in other words, extra notes that aren't ended with a comma are not allowed. A beat containing no notes or chords is defaulted to have a rest inserted.

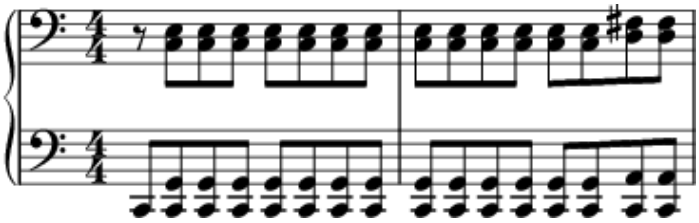
```
(C6E), ------ EDCB5AG, F(DF), (CE)(CE),
```



The beats are grouped into measures automatically. To change the time signature, refer to the later section on Attributes.

A score may also contain multiple *staves*, in which case a *section* can be created by separating scores for the respective staff with semicolons (`;`) and surrounding the whole group with curly braces (`{}`). The staves should be written top-to-bottom.

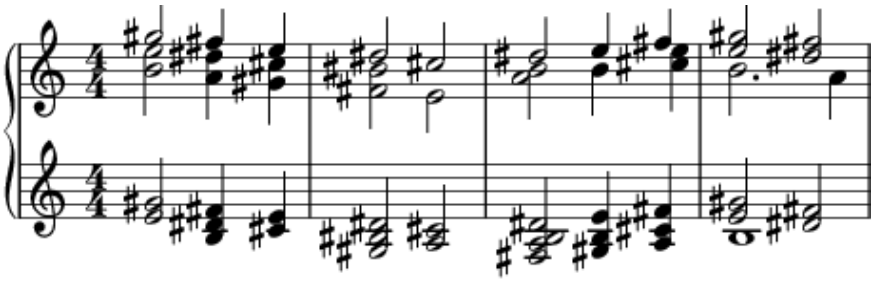
```
{.(C3E),(CE)(CE),(CE)(CE),(CE)(CE),
 (CE)(CE),(CE)(CE),(CE)(CE),(DF#)(DF#),;
 C2(CG),(CG)(CG),(CG)(CG),(CG)(CG),
 (CG)(CG),(CG)(CG),(CG)(CG),(CA)(CA),}
```



In some polyphonic passages, a single staff might be further divided into multiple *voices*. In this scenario, divide a staff into voices by separating the voices with semicolons (`;`) and surrounding the group with square brackets (`[]`). The voices should be written from top to bottom similarly.

```
{[G#5,-,F#,E, D#,-,C#,-,
  D#,-,E,F#, (EG#),-,(D#F#),-,;
  (B4E>),-,(AD#>),(G#C#>), (F#B#),-,E,-,
  (AB),-,B,(C#>E>), B,-,-,A,];

  (EG#),-,(B<D#F#),(C#E), (G#<B#<D#),-,(A<C#),-,
  (F#3ABD#>),-,(G#BE>),(AC#4F#),
  [(EG#),-,(D#F#),-,; B3,-,-,-,]}
```

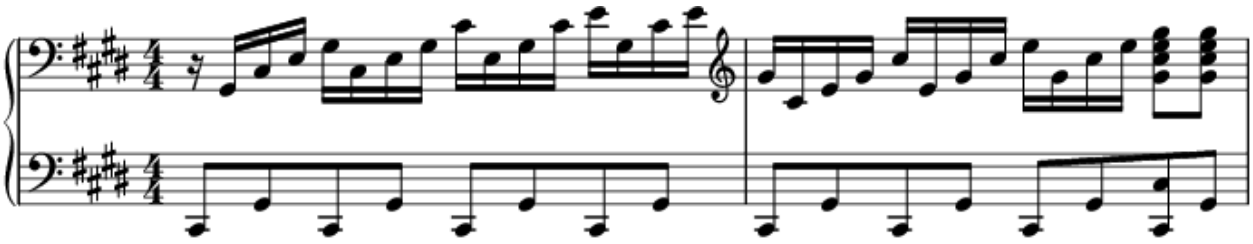


1.4 Attributes

Attributes can be applied to chords and measures. An *attribute set* is a list of attributes separated by commas (`,`) and surrounded by a pair of percent signs (`%%`). *Measure attributes* are accepted only at the end of a measure or at the beginning of a measure, while *chord attributes* can be placed anywhere.

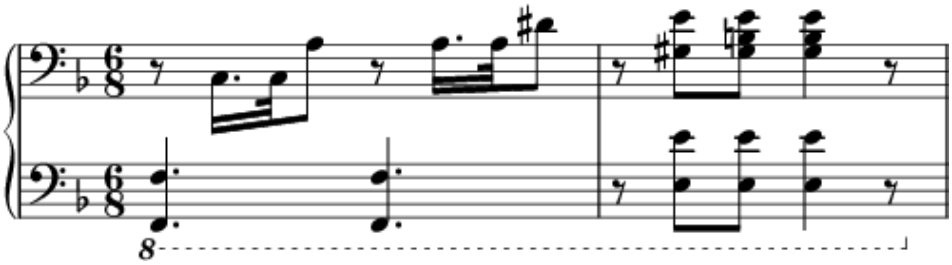
The *key signature attribute* is a kind of measure attribute, consisting of an integer from 0 to 7 and a character `s` or `f`. The number represents how many accidentals are there in the key signature, while the character indicates the type of accidentals (`s` for sharps and `f` for flats).

```
%4s%
{.G#2C#3E,G#C#EG#,C#>EG#C#4,EG#<C#E,
 G#C#EG#,C#>EG#C#5,EG#<C#E,(G#<C#EG#)(G#<C#EG#),;
 C#2G#,C#G#,C#G#,C#G#,C#G#,C#G#,C#G#,(C#C#>)G#,}
```



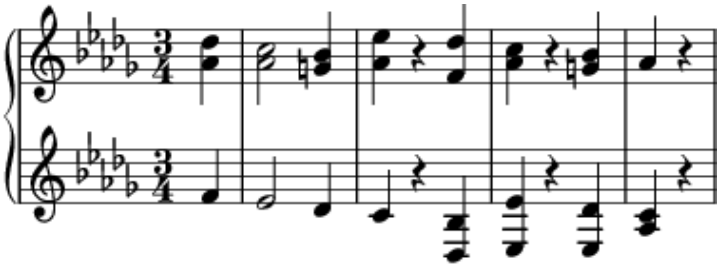
The *time signature attribute* is another kind of measure attribute, notated with the common fraction form.

```
%6/8, 1f%
{,C3--C,A, ,A--A,D#>, ,(G#E>),(G#BE>),(G#BE>),-,,;
 (F1F2),-,-, (F1F2),-,-, ,(E2E3),(E2E3),(E2E3),-,,}
```



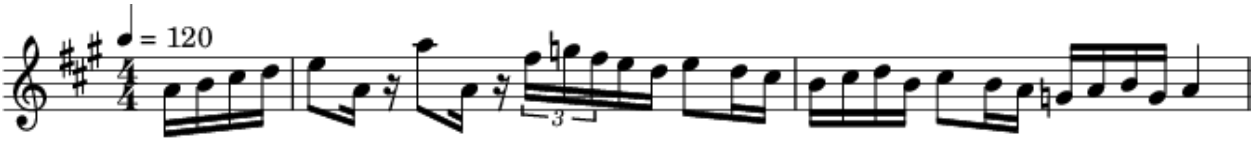
Use two slashes for the fraction to temporarily modify the time of a measure (used for pick-up beats).

```
%3/4, 1//4, 5f%
{(AbDb>),(AbC>),-,(GBb),(AbEb>),,
 (FDb>),(AbC>),,(GBb),%2//4% Ab,,;
 F,Eb,-,Db,C,,(Db3Bb),(EbEb>),,(EbDb>),(AbC>),,}
```



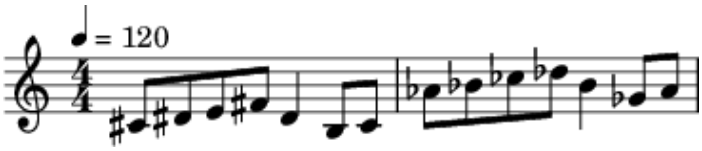
The *tempo attribute* is a kind of chord attribute. It is notated with just a single number indicating the BPM of the piece from this chord onwards.

```
%120, 4/4, 1//4, 3s%
A5<B<C#D,
E-A<.,A-A<., F#-G-F#-E--D--,E-DC#,
B<C#DB<,C#-B<A<, G<A<B<G<,A<,
```



The *transposition attribute* is another kind of chord attribute, meaning to transpose the piece by a given interval from this chord onwards. Such an attribute starts with a plus sign (+) or a minus sign (−) indicating whether to tranpose up or down; it is then followed by a letter (case-sensitive) denoting the quality of the interval, where `m` is for minor, `M` for major, `P` for perfect, `d` for diminished, and `A` for augmented; it is then ended with an integer, representing the diatonic number of the interval.

```
%120, -m2% DE,FG,E,CD,
%+d5% DE,FG,E,CD,
```



1.5 Macros

Macros can be used to avoid repetitions of identical text fragments. All the macros are substituted into their expanded form by a *preprocessor* before the parser gets to process the input.

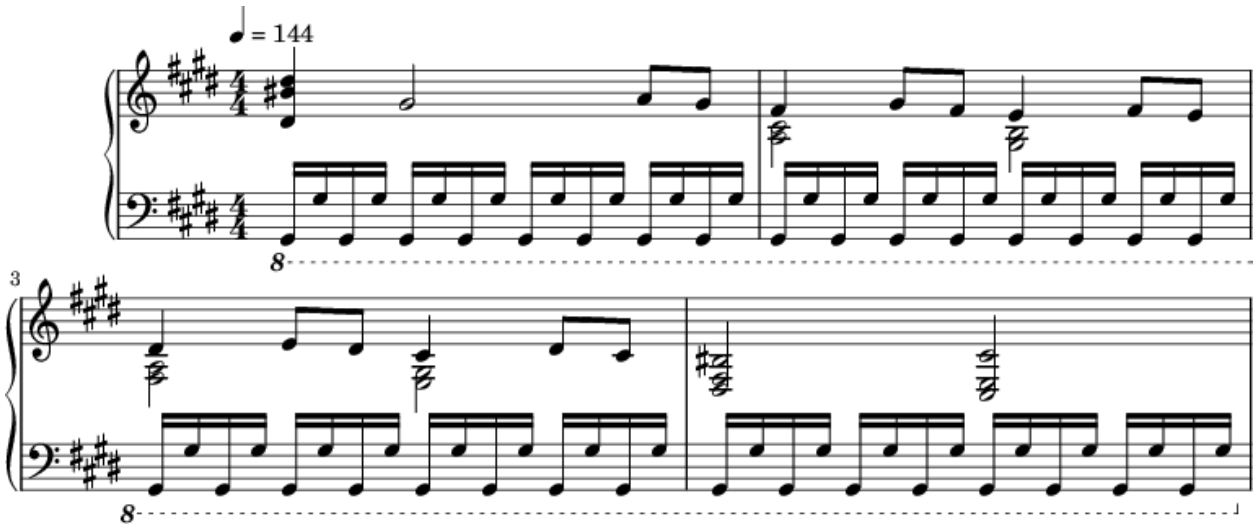
A *macro definition* starts with an exclamation point (`!`), followed by a macro name, then a colon (`:`), followed by the contents of the macro, and closes with another exclamation point.

```
!tr: E1E>EE>,EE>EE>,EE>EE>,EE>EE>,!
```

A *macro expansion* is the macro name surrounded by a pair of asterisks (`**`). The preprocessor will substitute such constructs with the corresponding macros' content.

```
!tr: E1E>EE>,EE>EE>,EE>EE>,EE>EE>,!

%144, 4s, +M3%
{(B<G#B),E,-,FE,
 [D,ED,C,DC, B<,CB3,A,BA,;
  (FA),-,(EG),-,(DF),-,(CE),-,]
 (B<DG#),-,(A<CA),-,;
 *tr* *tr* *tr* *tr*}
```



Re-defining a macro will not cause error, instead, the macro's content will be overwritten by the new definition.

```
!tr: E1E>EE>,!
!tr: *tr* *tr* *tr* *tr*!
```

CHAPTER

TWO

LIBRARY API

2.1 Conversion API

Music hkr::**parse_music**(std::string text)

Parse a string into a structured form.

Please refer to the syntax guide for more details.

Parameters

text –Text input.

Returns

Parsed music structure.

void hkr::**export_to_lilypond**(std::ostream &stream, *Music* music)

Convert structured music into Lilypond notation.

Parameters

- **stream** –The output stream to write into.
- **music** –The music to export.

2.2 Music Structures

enum hkr::**NoteBase**

The base part of a note name (C, D, E, F, G, A, B).

Values:

enumerator **c**

enumerator **d**

enumerator **e**

enumerator **f**

enumerator **g**

enumerator **a**

enumerator **b**

enum `hkr::IntervalQuality`

Different qualities for an interval.

Values:

enumerator **diminished**

enumerator **minor**

enumerator **perfect**

enumerator **major**

enumerator **augmented**

struct **Interval**

A music interval between two notes.

Public Functions

int **semitones**() const

Count how many semitones are there in the interval.

Public Members

int **number** = 1

Diatonic number.

IntervalQuality **quality** = *IntervalQuality*::*perfect*

Interval quality.

struct **Time**

Time signature.

Public Members

int **numerator** = 4

Numerator of the time signature, or how many beats are there in a measure.

int **denominator** = 4

Denominator of the time signature, or the duration of a single beat.

struct **Note**

A musical note.

Public Functions

Note **transposed_up**(int semitones) const noexcept

Find the note n seminotes above this one.

Note **transposed_down**(int semitones) const noexcept

Find the note n seminotes below this one.

Note **transposed_up**(*Interval* interval) const noexcept

Find the note at some interval above this one.

Note **transposed_down**(*Interval* interval) const noexcept

Find the note at some interval below this one.

std::int8_t **pitch_id**() const

Get the MIDI pitch ID of the note.

Public Members

NoteBase **base**

Base of the note.

int **octave**

Octave of the note, C4 (octave=4) is the middle C.

int **accidental**

Accidental of a note, positive integer for the amount of sharps, or flats if negative.

struct **Chord**

A chord containing multiple notes.

Public Members

std::vector<*Note*> **notes**

Constituents of this chord.

bool **sustained** = false

Whether this chord is a prolongation of the previous one.

Attributes **attributes**

Attributes of this chord.

struct **Attributes**

Attributes of a chord.

Public Members

std::optional<float> **tempo**

Tempo marking of this chord.

using `hkr::` **Voice** = std::vector<*Chord*>

A voice containing multiple chords.

using hkr::**Beat** = std::vector<*Voice*>

A beat containing multiple voices.

using hkr::**Staff** = std::vector<*Beat*>

A staff containing multiple beats.

struct **Measure**

Information about a measure.

Public Members

std::size_t **start_beat** = 0

Index of the first beat in this measure.

Attributes **attributes**

Attributes of this measure.

struct **Attributes**

Attributes of a measure.

Public Functions

void **merge_with**(const *Attributes* &other)

Merge another set of measure attributes into this one.

Any non-null attribute in the other attribute set will be overwritten upon this set.

Parameters

other –The other measure.

inline bool **is_null**() const noexcept

Checks whether this attribute set is completely empty.

Public Members

std::optional<int> **key**

The amount of sharps in the key signature, negative integer for flats.

std::optional<*Time*> **time**

Time signature of the current measure.

std::optional<*Time*> **partial**

The actual time of the current measure (for pick-up beats).

struct **Section**

A music section, containing multiple staves.

Public Functions

std::pair<std::size_t, std::size_t> **beat_index_range_of_measure**(std::size_t measure) const

Find the starting and ending beat indices of a measure in this section.

Parameters

measure –Index of the measure.

Returns

A pair of std::size_t being the starting (inclusive) and ending (exclusive) beats' indices.

Public Members

std::vector<*Staff*> **staves**

The staves.

std::vector<*Measure*> **measures**

Measure information.

using `hkr::Music` = std::vector<*Section*>

Music structure, containing multiple sections.

INDEX

T

V