Manuel du langage `music-score`

Introduction

Le langage `music-score` est un langage de programmation qui permet de produire très facilement des images de partitions simples (simple portée ou portée piano — pour le moment) en utilisant dans son moteur le langage LiliPond.

Une page en `music-score` peut ressembler à :

```
# Dans ma-musique.mus
--open
--barres
--time
--piano

mes12==
a'8 b cis d cis4 cis
<a, cis e>1

mes13==
a'8 b cis d cis4 cis
<a, cis e>1

mes14==
a'8 b cis d cis4 cis
<a, cis e>1

mes15==
a'8 b cis d cis4 cis
<a, cis e>1

-> partition-12a15
--mesure 12
--proximity 7
mes12<->15
mes12<->15
```

Production de l'image

Pour produire l'image issue de ce code, il suffit de la « construire » (`build`) dans Sublime Text avec `Cmd B` (`Cmd Maj B` la première fois pour choisir le langage (si c'est vraiment nécessaire car normalement la commande doit repérer qu'il s'agit d'un fichier « music-score » à son extension `.mus`.

Ce code, traité par le script `music-score.rb`, va produire l'image suivante :

Détail du code

```
# Options préliminaires
# ---------------------
# Option pour ouvrir le fichier après fabrication
--open
# Option pour afficher les barres de mesure
--barres
# Option pour afficher la métrique
--time
# ou (pour ne pas le mettre)
--time OFF
# ou (pour la préciser)
--time 3/4
# Option indiquant qu'il s'agit d'une partition de piano
--piano

# Définition des mesures
# ----------------------
# Définition de la mesure 12
# La première ligne contient la main droite
# La seconde ligne définit la main gauche
mes12==
a'8 b cis d cis4 cis
<a, cis e>1

mes13==
a'8 b cis d cis4 cis
<a, cis e>1

mes14==
a'8 b cis d cis4 cis
<a, cis e>1

mes15==
a'8 b cis d cis4 cis
<a, cis e>1

# Définition des images (systèmes)
# --------------------------------
# Le nom de l'image SVG (affixe)
-> partition-12a15
# Le numéro de mesure à indiquer au début
```

```
--mesure 12
# L'éloignement horizontal entre les notes
--proximity 7
# Indique de la mesure 12 à la mesure 15, à la
# main gauche et à la main droite
mes12<->15
mes12<->15
```

Après la production de l'image

Après la production du code, l'image est automatiquement rognée par inskape.

Dossier et nom des images produites

Par défaut (car il est possible de le déterminer explicitement), le nom des images et le dossier de leur destination sont définis par le nom du fichier `.mus` contenant le code music-score.

Soit le nom de fichier `partition.mus` contenant le code « music-score ».

Un dossier `partition` sera créé au même niveau que ce fichier, et contiendra les images produites.

Les images porteront le nom `partition-1.svg` `partition-2.svg`... `partition-N.svg` et seront placées dans le dossier `partition` ci-dessus.

Options principales

Toutes ces options peuvent être utilisées au début du code ou à n'importe quel endroit du fichier pour être modifiées à la volée. Par exemple, si on veut que les premières images soient produites avec des barres de mesures, on ajoutera en haut du fichier l'option `--barres` et si à partir d'une image on n'en veut plus, on pourra poser `--barres OFF` et remettre plus loin `--barres` pour spécifier qu'il faut à nouveau utiliser des barres de mesure.

> Pour désactiver une option après l'avoir activée, il faut utiliser :
> `--<option> OFF`

Effet recherché	Option	Notes
Affichage des barres de mesure	`--barres`	

Afficher la métrique	`--time`
	`--time OFF`
	`--time 3/4`	

Ne traiter que les images inexistantes	`--only_new`	Dans le cas contraire, toutes les images seront toujours traitées, qu'elles existent ou non, ce qui peut être très consommateur en énergie.
Ne pas afficher les hampes des notes	`--no_stem`	
Taille de la page	`--page <format>`	Par défaut, la partition s'affiche sur une page a0 en format paysage, ce qui permet d'avoir une très longue portée. `<format>` peut avoir des valeurs comme `a4`, `b2` etc.
Espace vertical entre les portées	`--staves_vspace <distance>`	Pour avoir l"espace normal, mettre 9. Au-delà (11, 12 etc.) on obtient un écart plus grand que la normale. "Staves vspaces" signifie (espace vertical entre les portées)
Commencer la relève après cette balise	`--start`	Permet de se concentrer sur un certain nombre d'images seulement. Tip : désactiver l'option `--only_new` pour refaire toujours les images, même si elles existent déjà.
Mettre fin à la relève-traitement des images	`--stop`	Après cette marque, `music-score` interrompra son traitement.
Ouvrir le fichier image après production	`--open`	Ouvre tout de suite le fichier dans Affinity Designer, ce qui permet de le « simplifier ».
Conserver le fichier LilyPond (`.ly`)	`--keep`	Cela permet de tester du code ou de voir où se situe un problème compliqué.
Détail des erreurs	`--verbose`	Permet de donner les messages d'erreur dans leur intégralité et notamment avec leur backtrace.

Options musicales

Effet recherché	Option	Notes
Définir la tonalité (armure du morceau)	`--tune` ou `--key` suivi de `A-G#b`	La lettre doit obligatoirement être en majuscule. Contrairement à Lilypond, qui permet d'indiquer les tonalités mineures (pour le chiffrage des chorus par exemple), ici, on met vraiment l'armure de la portée.
Définir le premier numéro de mesure	`--mesure [0-9]+`	C'est le numéro de mesure de la toute première mesure écrite (même si elle n'est pas complète, contrairement à la tradition idiote en musique).
Espacement horizontal entre les notes	`--proximity XXX`	`XXX` peut avoir une valeur de 1 à 50. Cf. les exemples de proximités ci-dessous

Exemples de proximités de notes (rendu)

Valeur de `proximity`	**Rendu**
Non définie	*(correspond ici à la proximité 5)*
`--proximity 1`	
`--proximity 5`	
`--proximity 10`	
`--proximity 20`	
`--proximity 50`	

Espacement horizontal automatique entre les notes--proximity xxx

(pour un espacement ponctuel, cf. la suite)

Note : c'est une option "ponctuelle", qui est abandonnée dès la
première utilisation.

Grâce à l'option --proximity, qui peut avoir les valeurs :
1 Le plus proche
4 Proche de la valeur naturelle, à voir
10Un peu éloigné
50Le plus éloigné
... on peut jouer sur le traitement de l'espacement entre les notes.
C'est très utile lorsque l'on veut par exemple mettre quatre mesures sur la même ligne mais qu'elles passent à la ligne.

```
-> partition-tres-serree
--page a3
--proximity 10
mesures1<->4
# => Entrainera un resserrement maximal entre les portées
```

Pour produire plusieurs images avec des espacements différents
(pour choisir le meilleur par rapport à l'affichage), on utilise
la formule : --proximity 1-10
Cela produira toutes les proximités de 1 à 10

Espacement entre les notes--<..>hspace

Parfois, on peut avoir besoin d'augmenter l'espacement entre les notes individuelles. On peut le faire, de façon de plus en plus importante avec les options :
--mini_hspace
--hspace
--big_hspace
--biggest_hspace
On les désactive pour la suite en ajoutant OFF comme pour toutes les options.
--hspace OFF

Nom du fichier de l'image (définition explicite)

Si une ligne commençant par `->` est placée avant l'expression musicale, elle contiendra le nom du fichier de sortie.

Par exemple :

```
-> monfichier
c d e f
```

... placera dans le fichier //monfichier.svg la partition résultant de l'expression `c d e f`.

Langage music-score

La partie ci-dessous présente les termes propres au langage « music-score ».

Notation LilyPond simplifiée

Cette section présente les notations de l'expression pseudo-lilypond qui diffèrent du langage original (toujours pour simplifier).

Barres de reprise

Objet	Code	Description
Début de reprise	`|:`	
Fin de reprise	`:|`	
Fin et début de reprise	`:|:`	
(*Code Lilypond pour les autres barres*)		
Fin de pièce	`|.`	

TODO La gestion des reprises avec première et autres fois

Clé de l'expression

On peut utiliser les marques normale de LilyPond mais il peut être plus pratique d'utiliser :

Objet	Code	Description
	`\cle G`	Clé de SOL 2e ligne
	`\cle F`	Clé de FA 4e ligne
	`\cle G1`	Clé de SOL 1ère ligne
	`\cle F3`	Clé de FA 3e ligne
	`\cle UT1`	Clé d'UT 1ère ligne
	`\cle UT2`	Clé d'UT 2e ligne
	`\cle UT3`	Clé d'UT 3e ligne
	`\cle UT4`	Clé d'UT 4e ligne
	`\cle UT5`	Clé d'UT 5e ligne

Tonalité de l’expression (armure)

Cf. les options musicales.

Numéro de mesure

Cf. les options musicales.

Triolet, quintolet et septolet

On les notes `3{note<duree> note note}`

Objet	Code	Description
	`3{note<duree> note note}`	TODO : il faudra traiter les quintuplet et autres sextolets de la même façon.

Signes d'interprétation/ornement

Objet	Code	Description
	c\mordent	Mordant inférieur
	c\prall	Mordant supérieur
	c\turn	Gruppeto
	c\reverseturn	Gruppeto inversé
	c\fermata	Point d'orgue

Pour d'autres ornements, voir https://lilypond.org/doc/v2.21/Documentation/notation/list-of-articulations.

Trilles

Objet	Code	Description
	`\tr(c')`	Noter la note trillée entre parenthèses.
	`\tr(cis' dis)`	Pour triller avec une autre note que la note naturelle.
	`\tr(c'1)- c a\-tr`	Noter le "tr-" pour commencer et le "-tr" pour finir
	`\tr(cis'1)- (b16 cis)\-tr d1`	Noter ici la tournure différente à la fin, avec les deux grâce-note entre parenthèses. Note quand même la logique générale.
	`\tr(cis'1 dis)- (b16 cis)\-tr d1`	On ajoute une note trillée avec une note étrangère

Petites notes (grace notes)

Objet	Code	Description
Non liées non barrées	`\gr(notes) note`	
Exemple simple	`\gr(b'16) a8 gis16 fais`	
Exemple multiple	`\gr(b'16 gis) a4`	
Non liées barrées	`\gr(note/)`	Remarquer la barre finale qui symbolise la note barrée
Exemple	`\gr(b'8/) a4`	
Exemple multiple	`\gr(b'16 gis/) a4`	(noter : non barré)
Appoggiature	`\gr(note-)`	
Exemple	`\gr(b'8-) a gis16 fis e4`	
Exemple multiple	`\gr(b'8 gis-) a4`	
Acciaccature	`\gr(note/-) note`	
Exemple	`\gr(ais'16/-) b4`	
Quand plusieurs notes	`\grace note[ note note note]`	

Changement de portée

Pour inscrire provisoirement les notes sur la portée au-dessus ou en dessous, utiliser `\up` et `\down`. Par exemple :

```
--piano
r1
c, e g \up c e c \down g e c
```

... produira :

Langage LilyPond (aide mémoire)

Ci-dessous la syntaxe propre à Lilypond, pour mémoire.

Altérations

Objet	Code	Description
`#`	**`is`**	Par exemple `fis` pour fa dièse
`b`	**`es`**	Par exemple `ees` pour mi bémol
`x`	**`isis`**	Par exemple `gisis` pour sol double-dièses
`bb`	**`eses`**	Par exemple `aeses` pour la double bémols

Accords

Objet	Code	Description
	`< notes >duree`	Bien noter que la durée est à l'extérieur de l'accord. Noter aussi que c'est la hauteur de la première note qui détermine la hauteur de référence pour la note suivante
Exemple	**2.**	
Snippet :	**`<`**	

SNIPPET: '<'

Liaisons

Objet	Code	Description
Liaisons de jeu	**`note1( autres notes)`**	
exemple	**`a'( b c d)`**	
Liaison de durée	**`note~ note`**	
Exemple simple	**`c1~ c2`**	
Exemple avec des accords	**`<c c'>1~ <c c'>4 <c~ g'~>2. <c e g>2`**	

Attache des hampes des notes

Objet	Code	Description
Forcer l'attache	`note[ notes]`	
Exemple	`a'16[ a a a a a a a]`	
Forcer l'attache vers le haut	`note^[ notes]`	
Exemple	`e'16^[ e e e] e`	
Forcer l'attache vers le bas	`note_[ notes]`	
Exemple	`a'16_[ a a a] a`	
Forcer une hampe seule en haut	`\stemUp e'4 \stemNeutral e'4^[]`	
Forcer les hampes de plusieurs notes non attachées (noires et blanches)	`\stemUp e'4 f g f \stemNeutral e'4^[] f^[] g^[] f^[]`	N1 : Noter que si plusieurs notes (plusieurs noires par exemple) doivent être traitées ensemble et que ce ne sont pas les mêmes hauteurs, il ne faut pas utiliser `e'4^[ f g f]` car dans ce cas tous les hauts de hampes s'aligneraient à la même hauteur. Il est impératif d'utiliser le code ci-contre. Cf. ci-dessous.
	`e'4^[ f g b, d f]`	
	`e'4^[] f^[] g^[] b,^[] d^[] f^[]`	
Forcer une hampe seule en bas	`g4_[]`	
		Pour plusieurs noires ou plusieurs blanches, cf. la note N1 ci-dessus.

Voir la page suivante pour la gestion des deux en même temps :
https://lilypond.org/doc/v2.19/Documentation/notation/beams.fr.html

Il semble qu'il faille utiliser :
`\override Beam.auto-knee-gap = #<INTEGER>`

Changement de positions des éléments

```
\slurUp     \slurDown     \slurNeutral     Lignes de liaison
\stemUp     \stemDown     \stemNeutral     Hampes de notes
```

Voix simultanées

Objet	Code	Description
	`<< { note note note } \\ { note note note } >>`	Le plus clair et le plus simple est d'utiliser des variables à la place des notes. La hauteur de la première note du second membre est calculée à partir de la première note du premier membre
Exemple	`<< { e'2 f e f } \\ { c,4 g' d g a e' d c } >>`	
Snippet	`2v`	

Dans cette formule, les deux voix auront leur propre 'voice'.
Mais il existe d'autres possibilités (cf. le mode d'emploi)

Petites notes (grace notes)

Objet	Code	Description
Non liées non barrées	**`\grace note note`** **`\gr(note) note`**	
Exemple	`\grace ais'16 b4` `\gr(ais'16) b4`	
Non liées barrées	**`\slashedGrace note note`** **`\gr(note/)`**	
Exemple	`\slashedGrace ais'16 b4` `\gr(ais'16/)`	
Liées non barrées	**`\appoggiatura note note`** **`\gr(note-)`**	
Exemple	`\appoggiatura ais'16 b4` `\gr(ais'16-)`	
Liées barrées	**`\acciaccatura note note`** **`\gr(note/-) note`**	
Exemple	`\acciaccatura ais'16 b4` `\gr(ais'16/-) b4`	
Quand plusieurs notes	**`\grace note[ note note note]`**	

Variable (aka « Definitions »)

On peut créer des « définitions » qui pourront être ensuite utilisées dans l'expression LilyPond fournie. Ceci permet d'écrire de façon plus modulaire et de pouvoir composer des segments différents très facilement.

Typiquement, on peut faire une définition pour chaque mesure. Dans une partition pour piano du premier mouvement de la Sonate facile de Mozart, on pourrait avoir par exemple :

```
--piano
--barres
--times

# La définition de la première mesure
# <nom de la variable-définition>
# <notes de main droite>
# <notes de main gauche>
mes1=
c'2 e4 g
c8 g' e g c, g' e g

mes2=
b'4.( c16 d) c4 r
d8 g f g d g f g
```

```
# Un segment comprenant ces deux mesures se définirait par :
-> mesures-1-a-2
mes1 mes2
```

Déclaration de la variable-définition

Sur une seule ligne, un nom ne contenant que des lettres majuscules ou minuscules et des chiffres, terminé par un ou deux signes « égal ».

- Avec **un seul signe égal**, c'est une **variable locale** (elle sera supprimée tout de suite après la réalisation de la première image).
- Avec **deux signes « égal »**, c'est une **variable globale** qui restera utilisable jusqu'à la fin du fichier.

La définition, qui peut tenir sur plusieurs lignes (une — monodie — ou deux — piano — pour le moment) et contenir des options, se termine à la première ligne vide rencontrée.

Par exemple :

```
mesure1=
c d e f g a b c
c b a g f e d c

# La ligne vide ci-dessus met fin à la définition
```

Utilisation d'un rang de variables

L'utilisation des variables-définitions prend tout son intérêt avec la **définition de l'expression par rang de variables**.

Très simplement, cela signifie que si on déclare ces variables-définitions :

```
mes1==
... notes ...

mes2==
... notes ...

mes3==
... notes ...

mes4==
... notes ...
```

… on peut déclarer facilement un segment (une image, donc) avec :

```
mes1<->4
```

Cela signifie que le segment sera constitué des mesures 1 à 4.

> Noter que pour le moment, on ne peut pas utiliser en même temps, en mode --piano, des définitions d'une seule main et des définitions des deux mains. Si on adopte un mode, il doit être utilisé pour tout l'extrait.
> Mais deux extraits différents peuvent utiliser deux modes qui diffèrent, par exemple :

```
# Un extrait avec définition d'une seule main

--piano

mg1=
c1 e g

md1=
g8( a b c) c2

-> essai_par_mains
mg1
md1

mgd1=
c1 e g
g8( a b c) c2

-> essai_deux_mains
mgd1
```

NOTA BENE

Noter un point très important : lors de l'utilisation de variables à plusieurs voix, l'expression lilypond ne peut qu'être exclusivement constituée de variables (sur une ligne, donc, puisque ce sont les définitions qui contiennent les différentes voix)

```
md1=
c2 e4 g

mg1=
g8( a b c) c2

md2=
b4. c8 c4 r
```

```
mg2=
d g f g c, g' e g

# Définitions à voix multiples
tout=
md1 md2
mg1 mg2

-> fichier
tout
```

Annexe

Le fichier « build » dans Sublime Text

> Note : je n'ai pas réussi à le faire remarcher, même en modifiant le première commande fautive qui appelle la version ruby 2.7.1.

C'est le fichier qui permet de jouer `Cmd B` avec le fichier music-score (`.mus`) actif et de produire les images qu'il définit.

Ce fichier est à mettre dans `/Library/Applications Support/Sublime Text 3/Packages/User/music-score.sublime-build`.

```
  // "shell_cmd": "make"
"cmd": [
  "/Users/philippeperret/.rbenv/versions/2.7.1/bin/ruby",
  "/Users/philippeperret/Documents/ICARE_EDITIONS/Musique/xDev/scripts/music-score/music-score.rb",
  "$file"
],
"selector": "source.music",
"file_patterns": ["*.mus"],
"target": "ansi_color_build",
"syntax": "Packages/ANSIescape/ANSI.sublime-syntax"
}
```

> Depuis le crash de 2021, ce fichier fait partie des backups universels.

Le fichier de coloration syntaxique pour Sublime Text

Ce code est à placer dans `/Library/Applications Support/Sublime Text 3/Packages/User/music-score.sublime-syntax`

> Note : depuis le crash de 2021, ce fichier fait partie des backups universels.
>
> Mais il ne fonctionne plus bien non plus

```
%YAML 1.2
---
# See http://www.sublimetext.com/docs/3/syntax.html
file_extensions:
  - mus
scope: source.music

contexts:
  # The prototype context is prepended to all contexts but those setting
  # meta_include_prototype: false.
  prototype:
    - include: comments

  main:
    # The main context is the initial starting point of our syntax.
    # Include other contexts from here (or specify them directly).
    - include: numbers
    - include: notes

  numbers:
    - match: '[^,a-g][0-9]+(\-[0-9]+)?\b'
      scope: constant.numeric.example-c

  notes:
    # Les notes
    - match: "\\b[a-gr](es|is)?(es|is)?[',](16|32|64|128|1|2|4|8)?"
      scope: music.note
    - match: "\\b[a-gr](es|is)?(es|is)?\\b?[',]?(16|32|64|128|1|2|4|8)?\\b"
      scope: music.note
    # Les accords
    - match: '<.+?>(16|32|64|128|1|2|4|8)?'
      scope: music.note.chord
    # Les liaisons
    - match: '\( .+?\)'
      scope: music.note
    # Les options qu'on peut trouver
    - match: '\-\-(verbose|keep|barres|time|piano|only_new|stop|start|open|big-|mini-
|biggest)(hspace)?( (OFF|ON))?\b'
      scope: music.option
    # Numéro de mesure et proximité
    - match: '\-\-(mesure|proximity) [0-9]+\b'
```

```
      scope: music.mesure.numero
    # Tonalité
    - match: '\-\-(tune|key) [a-zA-Z][#b]?'
    # Format de page
    - match: '\-\-page [a-zA-Z0-8]+'
      scope: music.mesure.numero
    # Définition (variable)
    - match: '[a-zA-Z0-9_]+\=\=?'
      scope: music.mesure.numero
    # Nom de fichier
    - match: '^\-> (.+)$'
      scope: music.output.name
    # Les clés
    - match: '\\clef? ("(treble|bass|ut)"|G1|G|F3|F|UT1|UT2|UT3|UT4|UT5)'
      scope: music.clef
    # Les barres de mesure
    - match: ':?\|:?'
      scope: music.mesure.barre
    # Special words
    - match: '\\(trill|slashedGrace|grace|appoggiatura|acciaccatura|break) ?'
      scope: music.clef

  inside_string:
    - meta_include_prototype: false
    - meta_scope: string.quoted.double.example-c
    - match: '\.'
      scope: constant.character.escape.example-c
    - match: '"'
      scope: punctuation.definition.string.end.example-c
      pop: true

  comments:
    # Comments begin with a '# ' and finish at the end of the line.
    - match: '# '
      scope: punctuation.definition.comment.example-c
      push:
        # This is an anonymous context push for brevity.
        - meta_scope: comment.line.double-slash.example-c
        - match: $\n?
          pop: true
```