

Manuel du langage `music-score` (`.mus`)

Manuel du `
langage music-score
(.mus)`

Introduction

Note pour produire des images pour ce manuel

Le langage `music-score` (`mus`)

Production de l'image

Détail du code

Code par paragraphe

Rognage automatique de l'image

Images produites

Dossier et nom des images produites

Inclusions

Statistiques

Remarques sur les statistiques

Le Tempo

Options principales

Résumé des options

Détail des options

Tonalité de la pièce

Numérotation des pages

Portées multiples

Espacement entre les systèmes

Espacement entre les portées

Taille des portées

Numérotation des mesures

Nommage des portées

Groupement des portées

Groupes à l'intérieur d'un même système

Cas spécial du PIANO

Fusion (merge) des silences

Arrêt ponctuel de la fusion des silences

Fichier de sortie MIDI

Espacement entre les notes (*proximity*)

Commandes

Commande de déclenchement de la compilation (START/STOP)

Nom du fichier de l'image (définition explicite)

LANGUAGE MUSIC-SCORE (*mus*)

Handy code

Répétition d'une note avec <note>*N

Répétition d'un code avec % ... %N

Notation LilyPond simplifiée

Les Barres

1re, 2e, etc. fois dans les reprises

Clé de l'expression

Tonalité de l'expression (armure et changement)

Transposition

Instruments transpositeurs

Triolet, quintolet et septolet

Ornements et signes d'interprétation

Signes d'interprétation

Trilles

Petites notes (grace notes)

Notes « mergées »

Arpège vers accord tenu

Changement de portée

Marques d'octave

Arpèges

Fonctions ruby (mode expert)

Constantes notes

Parenthèses dans les arguments

Langage LilyPond (aide mémoire)

Altérations

Accords

Liaisons

Attache des hampes des notes

Anacrouse

Changement de positions des éléments

Voix simultanées

Petites notes (grace notes)

Variable (aka « Définitions »)

Déclaration de la variable-définition

Variable dynamique

Hauteur de la variable
Répétition de la variable
Utilisation d'un rang de variables
NOTA BENE

Annexe

Le fichier « build » dans Sublime Text
Le fichier de coloration syntaxique pour Sublime Text
Étapes pour « arpège vers accord tenu »

Tests

Introduction

Note pour produire des images pour ce manuel

Pour produire facilement des images pour ce manuel :

- ouvrir un Terminal au dossier des images du manuel,
- jouer la commande :

```
1 | score-i[TAB] << CODE
2 | # ici le code mus
3 | # dont :
4 |
5 | -> nom-a-donner-a-image
6 | # code mus
7 | CODE
```

- récupérer l'image dans le dossier `scores` du dossier images du manuel,
- la glisser à l'endroit voulu dans ce manuel.

Le langage music-score (mus)

Le langage `music-score` (maintenant la commande **`score-image`**) 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` (langage **`mus`**) peut ressembler à :

```
1 | # Dans ma-musique.mus
2 | --barres
```

```

3  --time
4  --piano
5
6  mes12==
7  a'8 b cis d cis4 cis
8  <a, cis e>1
9
10 mes13==
11 b'8 cis d cis b4 a
12 <b, d fis>1
13
14 mes14==
15 cis8 d cis b a4 g
16 <cis, e gis>1
17
18 mes15==
19 d8 e fis g fis4 fis
20 <d, fis a>1
21
22 -> partition-12a15
23 --mesure 12
24 --proximity 7
25 mes12<->15
26 mes12<->15
27

```



Production de l'image

Pour produire l'image issue de ce code, on a utilisé la commande **score-image** en ligne de commande avec un *heredoc*. Comme ceci :

```

1  > score-image << MUS
2  --barres
3  --time
4  --piano
5
6  mes12==
7  a'8 b cis d cis4 cis

```

```

8 <a, cis e>1
9
10 mes13==
11 b'8 cis d cis b4 a
12 <b, d fis>1
13
14 mes14==
15 cis8 d cis b a4 g
16 <cis, e gis>1
17
18 mes15==
19 d8 e fis g fis4 fis
20 <d, fis a>1
21
22 -> partition-12a15
23 --measure 12
24 --proximity 7
25 mes12<->15
26 mes12<->15
27 MUS

```

Noter que pour taper **score-image** il suffit de taper **score-i** puis la touche tabulation. C'est valable pour toutes les applications de la suite Score (*Score-Builder*, *Score-Numbering*, etc.).

Détail du code

```

1
2 # Options préliminaires
3 # -----
4 # Option pour ouvrir le fichier après fabrication
5 --open
6 # Option pour afficher les barres de mesure
7 --barres
8 # Option pour afficher la métrique
9 --time
10 # ou (pour ne pas le mettre)
11 --time OFF
12 # ou (pour la préciser)
13 --time 3/4

```

```

14 # Option indiquant qu'il s'agit d'une partition de piano
15 --piano
16
17 # Définition des mesures
18 # -----
19 # Définition de la mesure 12
20 # La première ligne contient la main droite
21 # La seconde ligne définit la main gauche
22 mes12=
23 a'8 b cis d cis4 cis
24 <a, cis e>1
25
26 mes13=
27 etc.
28
29 # Définition des images (systèmes)
30 # -----
31 # Le nom de l'image SVG (affiche)
32 -> partition-12a15
33 # Le numéro de mesure à indiquer au début
34 --mesure 12
35 # L'éloignement horizontal entre les notes
36 --proximity 7
37 # Indique de la mesure 12 à la mesure 15, à la
38 # main gauche et à la main droite
39 mes12<->15
40

```

Code par paragraphe

Il est important de comprendre que ce code fonctionne **par paragraphe**, ce qui signifie que les éléments sont considérés comme « entier » lorsqu'ils ne sont séparés par aucune ligne vierge.

Cela permet entre autres choses d'avoir des partitions multipistes. Par exemple, un trio sera indiqué par :

```

1 # trio
2 -> trio
3 a b c d
4 a2.   a'
5 d8 d d d d d

```

... qui produira :



Tous les exemples de code donnés dans ce manuel font l'objet d'un test « checksum ». Pour lancer tous ces tests, il suffit de jouer en ligne de commande : **score-image tests -dir=manuel.**

Cela comporte quelques inconvénients :

Parmi les inconvénients, il faut faire attention à bien séparer les définitions. Par exemple :

```
1 mesure1=  
2 a b c d  
3 mesure2=  
4 e f g f
```

... ne produira pas les deux variables `mesure1` et `mesure2`, mais seulement la variable `mesure1` avec une erreur de `mesure2` inconnue.

Seule tolérance, pour la définition des noms d'image, reconnaissable à `->` en début de ligne :

```
1 mesure1=  
2 a b c d  
3 -> image_tolerable  
4 mesure1
```

Le code ci-dessus sera considéré comme :

```
1 mesure1=  
2 a b c d  
3  
4 -> image_tolerable  
5 mesure1
```

Rognage automatique de l'image

Après la production du code, l'image est automatiquement rognée par **Inskape** pour ne laisser aucun air autour.

Images produites

Un unique fichier `.mus` peut définir une ou plusieurs images. Par exemple, un fichier contenant le code suivant :

```
1 | # in durees.mus
2 |
3 | -> image1
4 | c d e f
5 |
6 | -> image2
7 | c2 d e f
8 |
9 | -> image3
10| c1 d e f
```

... produira 3 images contenant leur code respectif, `durees/image1.svg`, `durees/image2.svg` et `durees/image3.svg`.

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. Chaque ligne contenant du code `mus` produira une image.

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

Inclusions

Dans les fichiers `.mus`, on peut inclure d'autres fichiers `.mus` à l'aide de la commande :

```
1 | INCLUDE path/to/musFile
```

Le chemin `path/to/musFile` peut être relatif au fichier maître ou relatif au dossier `libmus` de l'application **Score-Image** qui définit des bibliothèques standards.

La première bibliothèque à avoir été créée est la bibliothèque **piano/Alberti.mus** qui définit toutes les basses d'Alberti dans des variables.

Un fichier inclus peut définir n'importe quoi, pourvu que ce soit du code `.mus` à commencer par :

- des options,
- des variables,
- des partitions.

Pour la gestion des variables, voir [Gestion dynamique des variables](#).

Statistiques

Le programme permet aussi de faire des **statistiques sur les notes** (nombre et durée). Il suffit :

- d'ajouter l'option `-s/--stats` à la ligne de commande (ou de mettre l'option `--stats` dans le fichier `mus`,
- de définir l'option `-t/--tempo=<val>[T]` en ligne de commande ou dans le code `mus`, pour calculer les durées réelles.

L'option et le tempo peut s'ajouter aussi directement dans le fichier **.mus** avec l'option :

```
1 | --stats
2 | --tempo 60T
```

Pour le tempo, voir [ici](#).

Ces options produisent un dossier **stats** contenant 4 fichiers avec toutes les notes classées 1) par ordre alphabétique, 2) pour nombre, 3) par durée, 4) en fichier CSV pour travail avec excel.

Note : cette option peut s'utiliser aussi avec l'application `score-extract` (**ScoreExtraction**) avec les mêmes options.

Remarques sur les statistiques

Les statistiques relèvent donc la fréquence d'utilisation de chaque note de la partition, ainsi que leur durée d'utilisation. Elles sortent le nombre de notes utilisées ainsi que la durée totale de chaque note mise bout-à-bout.

Certains partis-pris ont été adoptés :

- toutes **les répétitions** sont prises en compte (à l'avenir, si c'est nécessaire, on pourra imaginer une option qui permette de ne pas les prendre en compte)
- pour **les ornements**, on ne compte que la note elle-même (sauf pour les « grace notes » — cf. ci-dessous). En effet, comment considérer une trille par exemple ? Elle devrait comporter deux notes (les deux notes utilisées pour triller) et un certain nombre d'itérations indéfinissable de façon stricte avec des durées tout aussi indéfinissables. On pourrait se retrouver aussi avec des statistiques faussées qui amplifieraient l'utilisation d'une note simplement parce qu'elle est produite par la trille (on pourrait objecter que cette note n'est pas « amplifiée » puisqu'elle est, de fait, jouée dans la musique...).
- on fait une exception pour **les grace notes** (les **petites notes**), donc, c'est-à-dire les notes explicitement écrites, avec une durée définie, qui sont prises en compte. Conformément à la tradition de jeu, pour l'appogiature « barrée » (petites notes barrées), on définit sa durée au quart de la note qui la suit, en retirant cette durée à la note suivante.

Certaines erreurs découlent de l'écriture même et, pour le moment, ne peuvent pas être évitées. C'est le cas dans l'utilisation d'un arpège (ou similaire) conduisant à un accord, comme dans la partition suivante :



Dans cette partition, on comptera 2 Sol (celui en noire pointée et en croche lié à l'accord), 2 Do (celui en noire et celui en croche lié à l'accord).

Le Tempo

Le tempo (qui sert aussi pour jouer le fichier MIDI) doit toujours se mettre **en valeur de noire** (même si le tempo n'est pas à la noire). Si le tempo est écrit dans une autre valeur (blanche, croche...), alors faire la transposition (diviser par deux si c'est en croche, multiplier par deux si c'est en blanche).

On ajoute "**T**" au tempo lorsque c'est un rythme ternaire (on met alors en tempo la valeur de la **noire pointée**).

Par exemple, pour une 6/8 où la noire pointée (le temps) devra être joué à 100, on indiquera :

```
1 | --tempo 100T
```

Pour une mesure à 3/8 où les temps (donc la croche) doit être jouée à 100, on indiquera :

```
1 | --tempo 33T
```

Puisqu'une noire pointée, ici durera trois fois plus de temps qu'une croche.

Options principales

Toutes les options dont nous allons parler 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 sans barres de mesures, on ajoutera en haut du fichier l'option **--barres OFF** et si à partir d'une image on en veut, on pourra poser **--barres** et remettre plus loin **--barres OFF** pour spécifier qu'il faut à nouveau ne plus utiliser les barres de mesure.

Résumé des options

Pour désactiver une option (après l'avoir activée ou pas), il faut utiliser :

--<option> OFF

Par exemple :

--barres OFF

... pour ne plus afficher les barres de mesure pour toutes les images suivantes dans le fichier mus.

Effet recherché	Option	Notes
Tonalité de la pièce	<code>--key <tune></code> <code>--tune <tune></code>	Se définit par le nom de la note (a-z) en majuscule ou minuscule, l'altération et le mode. Cf. Tonalité de la pièce
Suppression de la gravure des barres	<code>--barres OFF</code>	
Ré-affichage des barres de mesure	<code>--barres</code>	S'emploie forcément après un <code>--barres OFF</code> , puisque par défaut les barres sont toujours gravées.
Afficher (ou non) la métrique	<code>--time</code> <code>--time OFF</code> <code>--time 3/4</code>	
Ne traiter que les images inexistantes	<code>--only_new</code>	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	<code>--no_stem</code>	
Transposition du fragment	<code>--transpose <from> <to></code>	Par exemple, <code>--transpose bes c'</code> va transposer le fragment, qui est en SI bémol, en Do, en prenant les notes les plus proches. Cf. Transposition
Taille de la page	<code>--page <format></code>	Par défaut, la partition s'affiche sur une page A0 en format paysage, ce qui permet d'avoir une très longue portée. <code><format></code> peut avoir des valeurs comme a4, b2 etc.
Espace vertical entre les portées	<code>--staves_vspace <x></code>	Pour avoir l'espace normal, mettre 9. Au-delà (11, 12 etc.) on obtient un écart plus grand que la normale. "Staves vspace" signifie (espace vertical entre les portées)
Espace vertical entre les systèmes	<code>--systems_vspace</code>	
Produire le fichier MIDI	<code>--midi</code>	Régler la valeur de tempo ci-dessous pour avoir le tempo.
Ouvrir le fichier image après production	<code>--open</code>	Ouvre tout de suite le fichier dans Affinity Designer, ce qui permet de l'améliorer si nécessaire.

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.
Portées multiples (cf. ci-dessous)	--staves_keys G,A,... --staves_names 1re,2e...	Permet de produire des portées empilées avec les clés et les noms voulus.
Nommage de la portée	--staves_names <nom>	Permet, notamment pour le piano, de préciser qu'il faut indiquer le nom (simplement en indiquant --staves_names Piano)
Taille de la portée	--staff_size <x>	Définit la taille de la portée (et de tous ses éléments). La valeur par défaut est 20 .
Affichage des numéros de page	--page_numbers <v>	<v> peut-être OFF (pas de numéro de page), arabic (chiffres arabes, roman-ij-lower (romain minuscules avec ligature), roman-ij-upper (romain majuscule avec ligature), roman-lower (romain minuscule sans ligature), roman-upper (romain majuscule sans ligature)
Statistiques	--stats	Produit toujours les statistiques en même temps que la partition, dans un dossier stats.
Tempo pour les statistiques et le fichier MIDI	--tempo <valeur>	Ne sert que pour les statistiques et le fichier MIDI. Si on doit ajouter l'indication « noire = valeur » au-dessus de la première portée, il ne faut pas le faire avec <i>ScoreImage</i> .
Arrêt de la fusion des silences	--merge_rests OFF	Cf. Fusion des silences .

Détail des options

Tonalité de la pièce

L'option **--tune** ou **--key** permet de définir l'armure ou la tonalité de la pièce.

La forme canonique est :

```
1 <note><altération><mode>
2
3 où :
4
5 <note> est une lettre de a à g (ou A à G)
6 <altération> est 'es' ou 'b' pour bémol, 'is' ou '#' ou 'd' pour
  dièse
7 <mode> est 'm' ou '-' pour le mineur (rien pour le majeur)
```

Voilà différentes formes valides de définitions :

- `--tune a` = tonalité de La majeur
- `--key Gm` = tonalité de Sol mineur
- `--tune bes-` = tonalité de Si bémol mineur
- `--tune c-` = tonalité de Do mineur

Numérotation des pages

On règle la numérotation des pages avec l'option **page_numbers**

```
1 --page_numbers OFF                # pas de numérotation
2 --page_numbers arabic             # chiffres arabes (défaut)
3 --page_numbers roman-lower        # Romains minuscules
4 --page_numbers roman-upper        # Romains majuscules
5 --page_numbers roman-ij-lower     # Romains minuscules avec ligatures (*)
6 --page_numbers roman-ij-upper     # Romains majuscules avec ligatures (*)
7
8 (*) Semble ne pas fonctionner, peut-être faut-il une police spéciale.
```

Portées multiples

On définit les portées multiples à l'aide de **--staves_keys** (pour les clés) et **--staves_names** (pour les noms — voir aussi [Nommage des portées](#)).

On doit les définir **de bas en haut**. C'est-à-dire que si on veut un violon au-dessus d'un piano, on doit définir :

```
1 | --staves 3
2 | --staves_keys F,G,G
3 | --staves_names Piano,Piano,Violon
```

Le simple fait qu'on trouve deux fois de suite le mot « piano » indique à ScoreImage de relier les deux portées.

Les noms des instruments doivent être mis en capitales si on veut qu'ils soient en capitales sur la partition.

En fait, ci-dessus, la marque « Piano,Piano » (qui pourrait être aussi « PIANO,PIANO » indique à ImageScore qu'on a une portée de piano. Il produit alors deux portées reliées dans un système propre au piano, avec une accolade, et une portée de violon. On l'appelle une « sonate avec piano » (sonate with piano).

Espacement entre les systèmes

L'espace vertical entre les systèmes se définit à l'aide de l'option **systems_vspace**.

Par exemple :

```
1 | --systems_vspace 30
```

Pour l'espacement vertical entre les portées d'un système, cf. ci-dessous.

Espacement entre les portées

L'espace vertical entre les portées se définit à l'aide de l'option **staves_vspace**.

Par exemple :

```
1 | --staves_vspace 40
```

Pour l'espacement vertical entre les systèmes, cf. [Espacement entre les systèmes](#).

Taille des portées

La taille des portées se règle à l'aide de l'option **--staff_size** suivie de la valeur à lui donner. La valeur par défaut est 20.

```
1 | --staff_size 22.5
```

Numérotation des mesures

Par défaut, LilyPond et donc *ScoreImage* numérote les mesures, au début de chaque système.

Pour supprimer toute numération, utiliser l'option :

```
1 | --measure OFF
```

Pour forcer la numérotation de la première mesure, utiliser :

```
1 | --first_measure
```

Noter que c'est nécessaire seulement si la première mesure est la 1. Dans le cas contraire, on numérote toujours la première.

Pour partir d'une autre mesure que 1, utiliser :

```
1 | -> mon_image  
2 | --measure 12  
3 | c d e f
```

La numérotation pour le code ci-dessus commencera à partir de la mesure 12.

Pour que les numéros se fassent **de 5 en 5** ajouter :

```
1 | --number_per_5
```

Ou par un autre numéro, quelconque :

```
1 | --number_per 12  
2 | # => numérotation des mesures de 12 en 12
```

Par défaut, les numéros de mesure se mettent au-dessus de la portée. Pour les mettre en dessous, utiliser l'option :

```
1 | --measure_number_under_staff
```

Nommage des portées

On peut nommer les portées à l'aide de l'option `--staves_names` (cf. [Portées multiples](#)).

On peut ajouter un vrai dièse ou un vrai bémol dans le nom en utilisant `_b_` pour le bémol et `_d_` ou `_#_` pour le dièse. Ils seront remplacés par de vrais signes bémols et dièse, pour un affichage parfait.

Groupement des portées

Il existe plusieurs façons de relier les portées et les barres de mesure. On trouve les valeurs suivantes :

- Portées
 - non reliées
 - reliées par un trait simple (type neutre)
 - reliées par un trait + crochet oblique (type quatuor, chœur)
 - reliées par un trait + accolade (type piano)
- Barres de mesure
 - reliées entre elles
 - non reliées entre elles

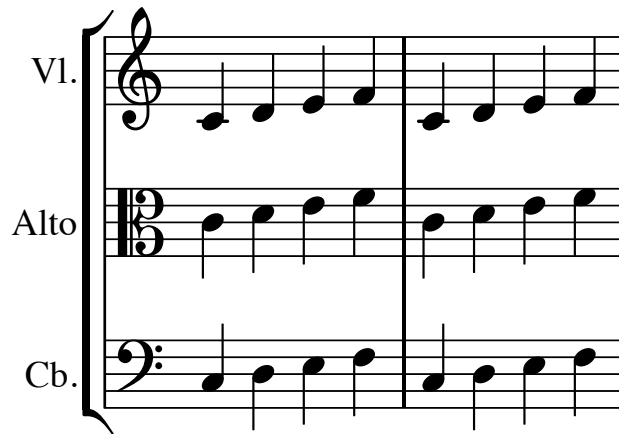
Par défaut, on utilise : **portées reliées par un trait simple avec les barres de mesure non reliées.**



Quand on veut un **crochet oblique**, on utilise « [...] ». Par exemple :

```
1 | --staves_names [Cb., Alto, Vl.]
2 | --staves_keys  F, UT3, G
```

... produira :



Quand on veut une accolade, on utilise « {...} ». Par exemple :

```
1 | --staves_names {Cb. Alto., Vl.}
2 | --staves_keys  F, UT3, G
```

... produira :

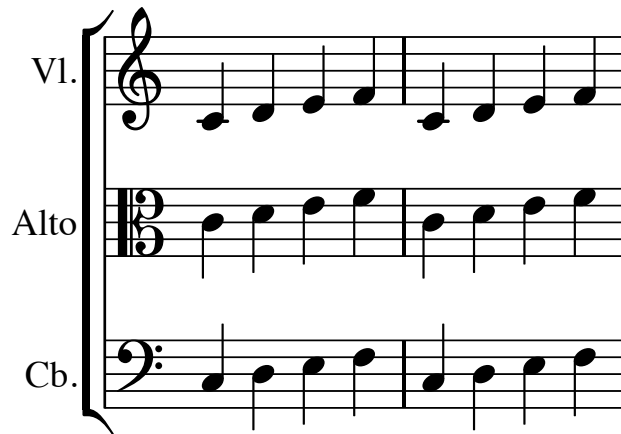


Comme on peut le voir, par défaut, les barres de mesure sont reliées entre elles. Pour utiliser **les barres de mesure non reliées**, on ajoute un « - » après l’accolade ou le crochet.

Par exemple :

```
1 | --staves_names [-Cb., Alto, Vl.]
2 | --staves_keys  F, UT3, G
```

... produira :



Tandis que :

```
1 | --staves_names {-Cb. Alto., Vl.}
2 | --staves_keys  F, UT3, G
```

... produira la même chose que les accolades seules, puisqu'**un groupement de ce type relie toujours ses barres de mesure.**

Si tous les instruments du groupe portent le même nom, il est utilisé comme **nom du groupe** et les portées ne sont plus nommées individuellement.

Comme dans :

```
1 | --staves_names {Bois, Bois, Bois}
2 | --staves_keys  F, G, G
```

... qui produira :



Groupes à l'intérieur d'un même système

On peut utiliser de la même manière les regroupements de portées à l'intérieur même d'un groupe, avec les accolades et les crochets, en indiquant par un « moins » l'absence de barres de mesure reliées.

Par exemple :

```
1 | --staves_names Cb. {Piano, Piano}, Vl.  
2 | --staves_keys F, F, G, G
```

... produira :

Cas spécial du PIANO

Le cas du piano est spécial car il possède sa propre option :

```
1 | --piano
```

C'est une écriture simplifiée pour :

```
1 | --staves_names {Piano, Piano}  
2 | --staves_keys F, G
```

Mais plus encore, ça simplifie l'écriture quand il y a des variables, puisque dans la définition de la partition on n'est pas contraint de préciser les deux mains :

```
1 | --barres  
2 | --piano  
3 |  
4 |  
5 | mesure1=  
6 | c'4 d e f  
7 | c,1  
8 |  
9 | mesure2=  
10 | g a b c  
11 | g1  
12 |  
13 | -> score  
14 | mesure1 mesure2
```

On pourra cependant utiliser l'écriture normale :

```
1 | --barres  
2 | --piano  
3 |  
4 |
```

```

5  mesure1=
6  c'4 d e f
7  c,1
8
9  mesure2=
10 g a b c
11 g1
12
13 -> score
14 mesure1 mesure2
15 mesure1 mesure2

```

... notamment dans le cas d'une utilisation d'autres variables. Par exemple :

```

1  --barres
2  --piano
3
4  mesure1=
5  c'4 d e f
6  c,1
7
8  mesure2=
9  g a b c
10 g1
11
12 mesure3=
13 r1
14 c,2 c'
15
16 -> score
17 mesure1 mesure2 mesure1
18 mesure1 mesure2 mesure3

```

... qui produira :



Noter ci-dessus que c'est seulement la main gauche de la mesure 3 qui a été utilisé, alors que la main droite a été empruntée à la mesure 1, conformément à la définition de la partition.

Fusion (merge) des silences

Par défaut, *ScoreImage* fusionne les silences (ce que ne fait pas LilyPond). Cela signifie qu'au lieu de graver :



... *ScoreImage* gravera :



On peut désactiver ce comportement par défaut en utilisant l'option **--merge_rests OFF** (note : « merge » signifie « fusionner » et « rest » signifie « repos », « silence » en musique).

```
1  --merge_rests OFF
2
3  -> score
4  << \stemDown g' r g r // \stemUp c r e r >>
5
6  # => Les silences seront conservés comme ci-dessus
```

Arrêt ponctuel de la fusion des silences

Si l'on conserve le comportement par défaut, on peut néanmoins désactiver localement la fusion des silences grâce à la marque **\not_merge_rests** (et la faire reprendre à l'aide de **\merge_rests**) :

```
1  << { \stemDown g' r } \\ { \stemUp c r } >> \not_merge_rests << {
   \stemDown g r } \\ { \stemUp e' r } >> \merge_rests << { \stemDown g, r
   } \\ { \stemUp c r } >>
```



Il peut être plus simple et plus lisible de fonctionner avec variable. Par exemple, pour produire le code ci-dessus (à quelques variations près, ajoutées pour bien montrer le traitement différent de la même variable) :

```

1 | b1=
2 | << \stemDown g' r // \stemUp c r >>
3 |
4 | b2=
5 | << \stemDown g' r // \stemUp e' r >>
6 |
7 | -> score
8 | b1 \not_merge_rests b2 \merge_rests b2

```

Ce code produira :



Notez plusieurs choses importantes ici :

- La marque `\not_merge_rests` s'applique à un « contexte », c'est-à-dire à un bout de code musical bien défini. On ne peut pas l'insérer par exemple de cette manière :

```

1 | << \stemDown g' r g \not_merge_rests r // \stemUp c r e r >>
2 | # => NE PRODUIRA PAS LE RÉSULTAT OBTENU

```

L'utilisation des variables, comme indiqué ci-dessus, permet de gérer ce problème, puisqu'une variable est forcément « isolée » dans le code LilyPond produit.

- La marque `\not_merge_rests` est *définitive*. C'est-à-dire que tant que le *ScoreImage* ne rencontre pas de `\merge_rests`, les silences ne seront plus fusionnés.

Fichier de sortie MIDI

Grâce à l'option `--midi` dans le code mus on peut produire un fichier MIDI à écouter (dans VLC par exemple).

Cette option marche de paire avec l'option **--tempo** qui détermine le tempo de l'écoute.

Par exemple :

```
1 | # ./score.mus
2 |
3 | --midi
4 | --tempo 120
5 |
6 | c d e f
```

... produira un fichier `./score/score.midi` qu'on pourra écouter dans VLC et qui jouera les notes au rythme de 120 à la noire.

À voir aussi : [le tempo](#).

Pour produire le fichier MIDI seul l'option `--midi` dans le code MUS ne suffit pas. Il faut l'utiliser dans la ligne de commande. Par exemple :

```
1 | score-image moncode.mus -midi
```

Bien noter que la commande ci-dessus ne produira QUE le fichier midi (à partir du code du fichier « moncode.mus »).

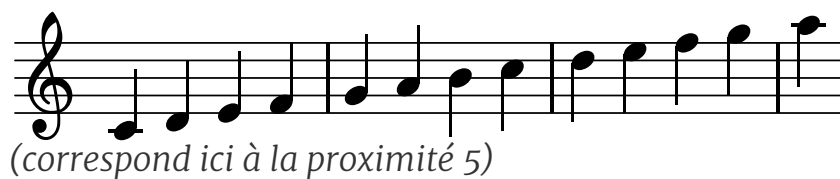
Espacement entre les notes (proximity)

On peut jouer sur la proximité entre les notes à l'aide de l'option **--proximity** suivie de la valeur d'éloignement (ou de rapprochement, c'est selon).

Valeur de
proximity

Rendu

Non définie



--proximity 1



--proximity 5



--proximity 10



--proximity 20



--proximity 50



Cette option s'applique à toutes les images suivantes dans le fichier MUS, mais peut être changée en cours de processus. Par exemple, le code :

```
1 | --proximity 2
2 |
3 | notes=
4 | g' f e d c1
5 |
6 | -> score_prox2
7 | notes
8 |
9 | --proximity 50
10 |
11 | -> score_prox50
12 | notes
13 |
```

... produira les images :



score_prox2.svg



score_prox50.svg

Commandes

Le code MUS répond à quelques commandes permettant de jouer sur la compilation.

Ces commandes sont reconnaissables au fait qu'elles sont toujours en capitales.

Commande de déclenchement de la compilation (START/STOP)

Lorsqu'un fichier est volumineux et contient de nombreuses images, on peut vouloir se concentrer parfois seulement sur quelques unes d'entre elles, sans avoir à les produire toutes à chaque fois, ou sans avoir à détruire celles qu'on veut revenir quand on utilise l'[option only_new](#).

On utilise alors la commande **START** pour indiquer le début du travail et la commande **STOP** pour indiquer où le fichier.

Par exemple, le fichier :

```
1  # in durees.mus
2
3  -> image1
4  c8 d e f
5
6  START
7
8  var=
9  c4 d e f
10
11 -> image2
12 var
13
14 STOP
15
16 -> image3
17 c2 d e f
```

... ne produira que l'image « durees/image2.svg ».

Noter que la commande se trouve sur une lignes « isolée », conformément au principe des paragraphes.

*Noter qu'il faut inclure la définition des variables dans le bloc « START ... STOP ».
Dans le cas contraire, une erreur de variable inconnue sera produite.*

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 :

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

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

LANGAGE MUSIC-SCORE (*mus*)

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

Handy code

Répétition d'une note avec <note>*N

De la même manière qu'on peut faire r8*4 en pur Lilypond, on peut répéter n'importe quelle note avec durée à l'aide de l'astérisque. Par exemple :

```
1 | cis16.*3
```

... produira le code

```
1 | cis16. cis16. cis16.
```

Noter que pour éviter toute confusion, cette possibilité se limite strictement à des notes avec ou sans altérations et pouvant définir leur durée. Tout autre groupe — par exemple présentant le doigté ou l'articulation — sera ignoré. Il faut alors utiliser d'autres moyens de répétitions (cf. ci-dessous).

Si une octave est appliquée à la note répétée, elle ne sera appliquée qu'au premier item.

Ainsi, le code :

```
1 | cis'16*4
```

... produira :

```
1 | cis'16. cis16. cis16. cis16.
```

Répétition d'un code avec % . . . %N

On peut utiliser le gabarit :

```
1 | % ... %N
```

... pour répéter un nombre illimité de fois un motif.

Par exemple, pour répéter 8 fois la séquence **c8 d e**, il suffit de faire :

```
1 | % c8 d e %8
```

Si le segment répété doit se trouver à une octave particulière, on peut enrouler le code dans un **relative** :

```
1 | \relative c,, { % c8 d e %8 }
```

Mais on peut le faire plus simplement avec :

```
1 | \cle F % c,,8 d e %4
```

... car l'octave de départ sera supprimée pour donner le code :

```
1 | \cle F c,,8 d e c d e c d e c d e
```

... qui produira :



Il est important, néanmoins, de surveiller l'octave à la fin de la chaîne, car si on fait :

```
1 | \cle F % c,,8 e g %3
```

... on produira le code :

```
1 | \cle F c,,8 e g c e g c e g
```

... qui verra monter sans redescendre l'arpège de Do majeur, car le deuxième Do repartira à l'octave du Sol précédent, etc. :



Dans ce cas, on peut préférer utiliser avec plus de sécurité le code normal de Lilypond :

```
1 | \cle F \repeat unfold 3 { c,,8 e g }
```

... qui produira :



Notation LilyPond simplifiée

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

Les Barres

Objet	Code	Description
Début de reprise	:	
Fin de reprise	:	
Fin et début de reprise	: :	
(Code Lilypond pour les autres barres)		
Fin de pièce	.	
Séparation de partie		

1re, 2e, etc. fois dans les reprises

Les premières, deuxième, etc. fois se gèrent à l'aide | <X> où <x> est le numéro de l'alternative :

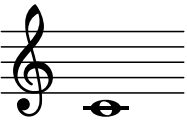
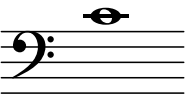
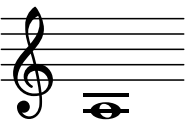

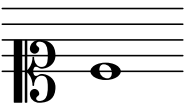



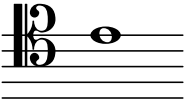
```
1 | |: .... |1 ... |2 ... |3,4 ... :|5 ... |6,7 || suite
```

- Note 1 :
- La barre « || » délimitant la dernière fois peut être aussi une autre reprise « |: » ou une barre de fin « |. ».
- Note 2 :

Il peut ne pas y avoir de barre de reprise de début, lorsqu'on revient au début pour faire la reprise.

Clé de l'expression

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

Objet	Code	Description
	<code>\cle G</code>	Clé de SOL 2 ^e ligne
	<code>\cle F</code>	Clé de FA 4 ^e ligne
	<code>\cle G1</code>	Clé de SOL 1 ^{ère} ligne
	<code>\cle F3</code>	Clé de FA 3 ^e ligne
	<code>\cle UT1</code>	Clé d'UT 1 ^{ère} ligne
	<code>\cle UT2</code>	Clé d'UT 2 ^e ligne
	<code>\cle UT3</code>	Clé d'UT 3 ^e ligne
	<code>\cle UT4</code>	Clé d'UT 4 ^e ligne
	<code>\cle UT5</code>	Clé d'UT 5 ^e ligne

Tonalité de l'expression (armure et changement)

Pour la définition de l'armure, voir [Tonalité de la pièce](#).

On marque un changement de tonalité au cours de la pièce simplement avec la marque `\tune` ou `\key`.

Pour ne pas ajouter la double barre, on écrit l'expression en « pur » LilyPond, c'est-à-dire avec la marque `\key` (pas `tune` cette fois), la note en minuscule et l'altération par `es` (bémol) ou `is` (dièse). On peut indiquer si c'est une tonalité mineure par `\minor` ou utiliser l'armure du relatif majeur.

Ainsi :

```
1 | c d e f \tune Ebm ges f ees d ees2
```

...produira :



Tandis que :

```
1 | c d e f \key ees\minor ges f ees d ees2
2
3 | # ou
4
5 | c d e f \key ges ges f ees d ees2
```

Noter que le `\minor` est « collé » à la note.

... produiront tous les deux :



Transposition

On peut transposer tout un fragment à l'aide de l'option `--transpose` en indiquant la note/tonalité de référence (aka la note de départ) puis la note/tonalité d'arrivée.

Par exemple :


```
1 | --transpose c d
2 |
3 | -> score
4 | c d e f g a b c
```

... transposera le fragment de Do majeur vers Ré majeur :



Tandis que :

```
1 | --transpose d c
2 |
3 | -> score
4 | d e fis g a b cis d
```

... transposera ce fragment en Ré majeur de Ré majeur vers Do majeur :

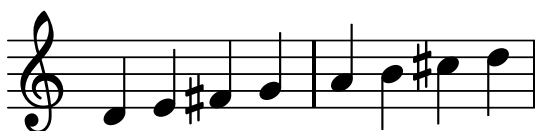


NB: Si vous voulez que l'armure tienne compte de la transposition quand la tonalité est Do majeur, il faut écrire explicitement cette tonalité à l'aide de l'option **--tune c**.

Ainsi, le code :

```
1 | --transpose c d
2 |
3 | -> score
4 | c d e f g a b c
```

... produira :



Tandis que le code :

```

1 | --tune c
2 | --transpose c d
3 |
4 | -> score
5 | c d e f g a b c

```

... produira lui :



Si la tonalité de départ est auteur de Do majeur, elle sera indiquée donc elle sera traitée de la même manière.

Instruments transpositeurs

Le traitement des instruments transpositeurs est extrêmement simple avec *music-score* : il suffit d'indiquer `\trans` suivi de la note de transposition (telle quelle) pour produire les bonnes notes avec la bonne armure.

Par exemple, pour une clarinette en La (A), il suffit d'indiquer `\trans a` :

```

1 | -> score
2 | \trans a { c d e f }

```

Exemple avec un saxophone en Eb, un cor anglais (qui est en F), une clarinette en Sib, une flûte :

```

1 | --barres
2 | --staves_names Sax Eb, Cor A., Cl.(Bb), Fl.
3 | --staves_keys G, G
4 |
5 | fl=
6 | e'1
7 |
8 | cl=
9 | c4 d e f
10 |
11 | cor_anglais=
12 | c4 e g c
13 |
14 | sax=

```

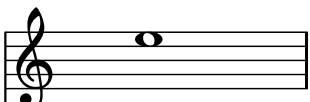
```

15 | g'4 f e d
16 |
17 | -> score
18 | fl
19 | \trans bes { cl }
20 | \trans f { cor_anglais }
21 | \trans ees { sax }
22 |


```

... produira la partition :


Fl.




Cl.(Bb)



Cor A.

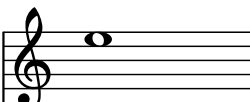


Sax Eb




qui sonnera


Fl.




Cl.(Bb)



Cor A.



Sax Eb



Triolet, quintolet et septolet

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

Objet	Code	Description
	<code>3{note<duree> note note}</code>	<p>TODO : il faudra traiter les duolets, quintuplets et autres sextolets de la même façon.</p>













Pour s'assurer que le rythme soit bon (et que les statistiques soient bien calculées), ne pas oublier de mettre la durée sur la note ou le groupe de notes suivant cette marque. Par exemple :

```










1 | 3{c8 d e} f8
2 | # => pour que le Fa suivant soit bien compté en croche simple.

```

Ornements et signes d'interprétation

Objet	Code	Description
	<code>c'\mordent</code>	Mordant inférieur (au-dessus de la note)
	<code>c_\mordent</code>	Mordant inférieur sous la note
	<code>c'\mordentb</code>	Mordant inférieur avec note bémolisée (au-dessus de la note) Le bémol est le « b » après « mordent »
	<code>c'\mordent#</code>	Mordant inférieur avec note diésée (au-dessus de la note) Le dièse est le « # » après « mordent »
	<code>c'\mordentn</code>	Mordant inférieur avec note bécarisée (au-dessus de la note) Le bécarre est le « n » après « mordent » (« n » pour « natural »)
	<code>c'\prall</code>	Mordant supérieur (au-dessus)
	<code>c'\prall#</code>	Mordant supérieur diésé
	<code>c'\prallb</code>	Mordant supérieur bémolisé
	<code>c'\pralln</code>	Mordant supérieur bécarisé
	<code>c_\prall</code>	Mordant supérieur sous la note
	<code>c'\turn</code>	Gruppetto sur la note.
	<code>c^\turn</code>	Pour forcer la gravure du gruppetto au-dessus de la note, on peut ajouter ^ conformément au langage LilyPond. Noter

que dans l'exemple, c'est superflu.

	c' \turnb	Gruppetto sur la note avec note bémolée au-dessus
	c' \turn#	Gruppetto sur la note avec note diésée au-dessus
	c' \turnn	Gruppetto sur la note avec note bécarriée au-dessus
	c' \turn/b	Gruppetto sur la note avec note bémolée au-dessous
	c' \turn/#	Gruppetto sur la note avec note diésée au-dessous
	c' \turn/n	Gruppetto sur la note avec note bécarriée au-dessous
Toutes les autres combinaisons sont possibles. Par exemple :		
	c' \turn#/b	Gruppetto sur la note avec note diésée au-dessus et note bécarriée au-dessous
	c' \turnb/#	Gruppetto sur la note avec note bémolée au-dessus et note diésées en dessous
	c' \turnn/n	Gruppetto sur la note avec note bécarriée au-dessous et au-dessus
	c' _\turn	Gruppetto sous la note.
	c' \reverseturn	Gruppetto inversé
	c' \reverseturn# / b	Gruppetto inversé avec notes altérées



`c'\reverseturn`

Gruppetto inversé sous la note



`c'\haydnturn`

Gruppetto haydnien



`c'\haydnturn#/#`

Gruppetto haydnien en dessous avec notes altérées



`c'\slashturn`

Gruppetto barré en dessous



`c^\slashturnb/b`



Gruppetto barré avec notes altérées

Pour d'autres ornements, voir






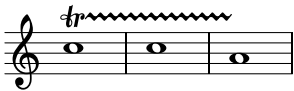

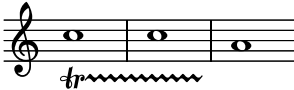




<https://lilypond.org/doc/v2.21/Documentation/notation/list-of-articulations>.

Voir aussi [Marques d'expression](#).

Signes d'interprétation

Objet	Code	Description
		<code>c'\fermata</code>
		<code>c_\fermata</code>

Trilles

Objet	Code	Description
	<code>\tr(c')</code>	Noter la note trillée entre parenthèses.
	<code>_tr(c1')</code>	Pour forcer la trille en dessous
	<code><< d'2 // \^tr(a) >></code>	Pour forcer la trille au-dessus. Bien sûr, c'est juste pour forcer la trille à s'afficher par défaut en dessous (et donc voir l'effet du circonflexe) que l'exemple ci-contre a été donné. Il n'est pas utilisable dans la réalité, puisqu'on penserait que c'est le Ré qui est trillé.
	<code>\tr(aes8.) (g32 aes)</code>	Noter la parenthèse qui commence la liaison sur la note trillée qui est "détachée" de la trille. Sinon la trille serait mal interprétée
	<code>\tr(cis'1 dis)</code>	Pour triller avec une autre note que la note naturelle.
	<code>\tr(c'1)- c a\tr</code>	Noter le "tr-" pour commencer et le "-tr" pour finir. Le programme signalera une erreur si on oublie la marque de fin. Noter que pour que la « vague » de la trille aille jusqu'à la note suivante, il faut introduire cette note dans la marque <code>\tr \tr</code> (ou utiliser le code un peu plus bas). Dans le cas contraire, on obtiendra l'image ci-dessous.
	<code>\tr(c'1)- c\tr a</code>	La « vague » de la trille s'arrête sur la dernière note comprise entre <code>\tr</code> et <code>\tr</code> .
	<code>_tr(c'1)- c a\tr</code>	La même chose en forçant la trille en dessous.
	<code>c'1 \startTr c a \stopTr \startTr a2 s \stopTr</code>	Les codes ci-dessus ne permettant pas d'enchaîner les trilles avec leur « vague », on utilise les marque <code>\startTr</code> et <code>\stopTr</code> comme ci-contre pour y parvenir.
	<code>\tr(cis'1)- (b16 cis)\tr d1</code>	Noter ici la tournure différente à la fin, avec les deux grâce-note entre parenthèses. Noter quand même la logique générale.
	<code>\tr(a'1)- (gis16 a)\tr bes1</code>	Même chose avec une liaison.
	<code>\tr(a'1)- (gis16(a))\tr bes1</code>	Même chose avec une liaison courte sur les deux petites notes



\tr(cis'1 dis)- (b16 cis)\-tr
d1

On ajoute une note trillée avec une note étrangère

Petites notes (grace notes)

... en :



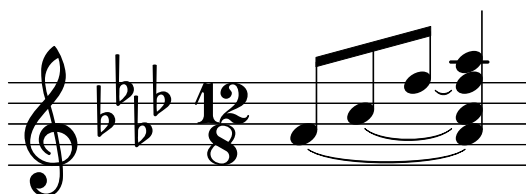
On obtient ce merge en ajoutant `\mergeNotes` avant les notes. Ce merge sera effectif jusqu'à la prochaine parenthèse de délimitation de groupe de note (donc c'est variable en fonction de la façon d'écrire la musique avec LilyPond).

Ce `\mergeNotes` provoque dans le moteur l'écriture de `\mergeDifferentlyHeadedOn` `\mergeDifferentlyDottedOn`. On pourra préférer utiliser ces deux marques, ou seulement l'une d'entre elles, dans le détail, pour un résultat différent attendu.

Arpège vers accord tenu

VERSION SIMPLIFIÉE

Pour obtenir l'image :



... utiliser le code simplifié suivant :

```
1 | \tieWait \stemUp aes'8~ \tieDown c~ f~ <aes, c f aes>4
```

On peut le simplifier encore en précisant par une capitale « U » ou « D » à la fin de `\tieWait` la direction des liaisons (tie) et par voie de conséquence des hampes (stem) si elles sont fixes comme ici. Cette lettre concernera la position des liaisons, la position des hampes étant à l'inverse. « D » signifie « down » (bas) et « U » signifie « up » (vers le haut).

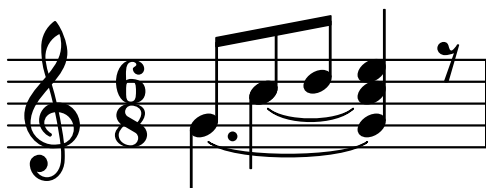
Donc on peut encore simplifier le code ci-dessus par :

```
1 | \tieWaitD aes'8~ c~ f~ <aes, c f aes>4
```

Note : Les liaisons ne semblent malheureusement pas obéir à tous les coups.

VERSION AVEC DURÉE DE NOTES

On peut également obtenir une version avec les notes « doublées » avec leur durée, ainsi :



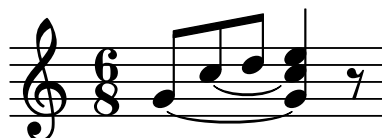
Ci-dessus, comment comment le Sol et le Do indiquent *explicitement* leur durée contrairement à l'exemple ci-dessus qui le suggérait seulement.

L'ordre de construction de cette cellule est importante, on la détaille ci-dessous pour arriver jusqu'au code voulu.

D'abord, il nous faut les croches liées à l'accord, c'est ce que nous avons vu plus haut. Nous allons utiliser `\tieWait` mais en ajoutant un « D » pour indiquer que les liaisons (« tie ») doivent être en bas (sinon, à cause de la position haute des notes, elles seraient au-dessus). On obtiendra donc :

```
1 | --time 6/8
2 |
3 | \tieWaitD g'8~ c~ d <g, c e>4 r8
```

On obtient dans un premier temps :



On ajoute **à la suite de ce code** les notes avec leur durée explicite en considérant des voix simultanées :

```
1 | --time 6/8
2 |
3 | << { \tieWaitD g'8~ c~ d <g, c e>4 r8 } \\ { g4. s } \\ { s8 c4 s4. }
   >>
```

Ce code produira :



Il faut donc corriger les erreurs :

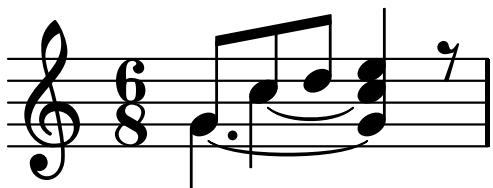
- mettre la hampe (stem) de Do dans le bon sens,

- « merger » les notes pour ne pas doubler les têtes.

On arrive donc au code :

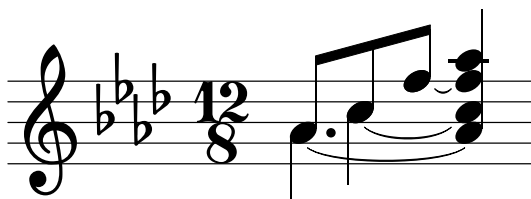
```
1 | --time 6/8
2 |
3 | << { \tieWaitD g'8~ c~ d <g, c e>4 r8 } \\ { \mergeNotes g4. s } \\ {
   | \mergeNotes \stemDown s8 c4 s4. } >>
```

... qui produit le code suivant :

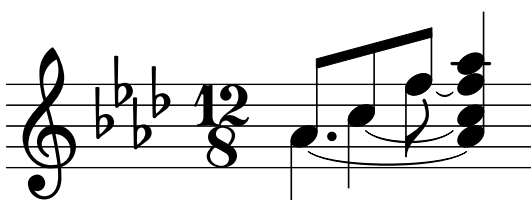


DESCRIPTION D'UN PROBLÈME

Sauf erreur de notre part, il est extrêmement difficile, avec LilyPond, d'obtenir l'image suivante, qui ressemble à la précédente, mais se différencie au niveau de la dernière note de l'arpège, le Fa en croche lié à l'accord :



Le mieux que nous puissions réussir :



Notez ci-dessous la note Fa qui possède sa hampe doublée.

Pour obtenir ce code, nous devons utiliser le code très couteux suivant :

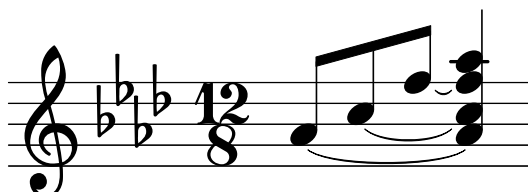
```
1 | \mergeNotes << { \stemUp aes'8 c f } \\ { \set tieWaitForNote = ##t <<
   | { \stemDown aes,4.~ } { s8 \tieDown \stemDown c4~ } { s4 \tieDown
   | \stemDown f8~ } { s4. \stemUp <aes, c f aes>4 } >> } >>
```

Il utilise notamment la propriété **tieWaitForNote** qui permet de lier des notes à un accord de façon simple.

Si l'on ne tient pas à donner explicitement la durée exacte de chaque note, on peut simplement utiliser :

```
1 \set tieWaitForNote = ##t \stemUp aes'8~ \tieDown c~ f~ <aes, c f aes>4
2
3 # Et donc la version réduite :
4
5 \tieWaitD aes'8~ c~ f~ <aes, c f aes>4
```

... qui produira :



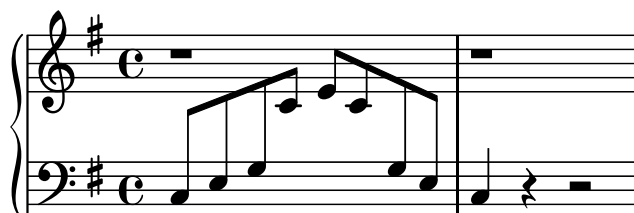
Voir en annexe les [étapes pour obtenir ce code](#) pour bien le comprendre.

Changement de portée

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

```
1 --piano
2 r1
3 c, e g \up c e c \down g e c
```

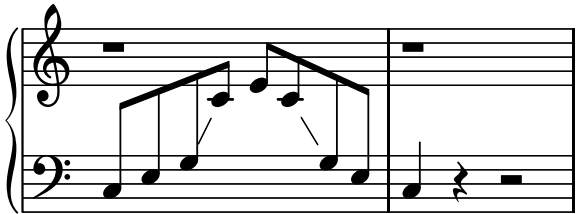
... produira :



On peut indiquer explicitement le lien entre deux notes qui changent de portée (par un trait) en ajoutant l'indication `\showStaffSwitch`

```
1 | --piano
2 |
3 | r1 r1
4 | c e g \showStaffSwitch \up c e c \showStaffSwitch \down g e c4 r r2
```

Produira :



Marques d’octave

Pour inscrire la marque d’octave, on peut utiliser `\8ve` (descendra les notes d’une octave et ajoutera la marque), `\15ve` (descendra les notes de deux octaves et ajoutera la double marque).

On fait l’inverse avec `\-8ve` (pour remonter les notes d’une octave) et `\-15ve` (pour remonter les notes de deux octaves).

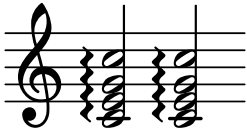
On terminera toutes les marques précédentes avec `\0ve`.

Arpèges

On peut simplifier la marque `\arpeggio` par la marque `\arp`.

```
1 | <c e g c>2\arpeggio <c e g c>2\arp
```

... produira :



Fonctions ruby (mode expert)

En mode expert, on peut produire du code de toute pièce avec des fonctions ruby. On peut penser par exemple au premier prélude de Bach dans le premier clavier tempéré. Presque d'un bout à l'autre on retrouve le même motif qu'il serait laborieux de répéter. Il se présente dans la première mesure de cette manière, écrit de toutes notes

:

```
1 -> score
2 r8 g'16 c e g, c e r8 g'16 c e g, c e
3 << r16 e8.~ e4 // c2 >> << r16 e8.~ e4 // c2 >>
```

Voilà la méthode qu'il est possible de créer dans un fichier qu'on appellera par exemple **module_image.rb** (peu importe son nom, c'est son extension qui fait qu'il sera chargé par ScoreImage).

```
1 module ScoreImage # ce nom est impératif
2
3   def motif(basse, contrebasse, notes_sup)
4     ns = notes_sup.split(' ')
5     <<~TEXT
6     % #{ns[0]}'16 #{ns[1]} #{ns[2]} #{ns[0]}, #{ns[1]} #{ns[2]} %2
7     % << r16 #{contrebasse}8.~ #{contrebasse}4 // #{basse}2 >> %2
8     TEXT
9   end
10
11 end
```

Il nous suffit maintenant d'appeler la méthode `#motif` dans le code `.mus` à l'aide du préfix **fn_** (« fn » comme « fonction »).

```
1 -> score
2 fn_motif("c", "e", "g c e")
```

Noter que contrairement à du pur ruby, il faut obligatoirement utiliser les parenthèses pour délimiter les arguments même lorsqu'il n'y en a pas.

Constantes notes

On peut utiliser les constantes au lieu de guillemets pour simplifier l'écriture :

```
1 -> score
2 fn_motif(c, e, g)
```

Parenthèses dans les arguments

Si les arguments contiennent des parenthèses, pour éviter toute confusion, on utilise des double-parenthèses pour délimiter les arguments de la fonction :

```
1 | fn_motif(( "a( b c d)" ))
```



Langage LilyPond (aide mémoire)

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


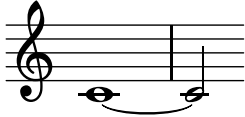


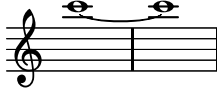

Altérations

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
	!	Altération de prudence. Par exemple a! pour forcer la marque du bécarré lorsque le La a été diésé dans la mesure précédente.
	?	Altération entre parenthèse. Par exemple a? pour mettre un bécarré avant le La entre parenthèses pour bien indiquer qu'il est bécarré.








Accords

Objet	Code	Description
	<code>< notes > duree</code>	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	<code>2.</code>	
Arpège	<code><c e g>\arpeggio</code>	Il suffit d'ajouter la marque <code>\arpeggio</code> après l'accord (et la durée) pour obtenir un arpège. 
Snippet :	<code><</code>	

Liaisons

Objet	Code	Description
Liaisons de jeu	<code>note1(autres notes)</code>	
exemple	<code>a' (b c d)</code>	
forcer en dessous	<code>a'_(b c d)</code>	
forcer au-dessus	<code>a'^ (b c d)</code>	
Forcer en haut	<code>\slurUp</code>	Pour revenir au comportement par défaut : <code>\slurNeutral</code>
Forcer en bas	<code>\slurDown</code>	Pour revenir au comportement par défaut : <code>\slurNeutral</code>
Liaison de durée	<code>note~ note</code>	
Exemple simple	<code>c1~ c2</code>	
Exemple avec des accords	<code><c c'>1~ <c c'>4 <c~ g'~>2. <c e g>2</code>	
Forcer la liaison en haut	<code>\tieUp</code>	
Forcer la liaison en bas	<code>\tieDown</code>	
Pour revenir au comportement par défaut : <code>\tieNeutral</code>	<code>\tieNeutral</code>	 <p>Dans l'exemple ci-dessus, le <code>\tieNeutral</code> est inséré entre les deux notes.</p>

Attache des hampes des notes

Objet	Code	Description
Forcer l'attache	<code>note[notes]</code>	
Exemple	<code>a'16[a a a a a a]</code>	
Forcer l'attache vers le haut	<code>note^[notes]</code>	
Exemple	<code>e'16^[e e e] e</code>	
Forcer l'attache vers le bas	<code>note_[notes]</code>	
Exemple	<code>a'16_[a a a] a</code>	
Forcer une hampe seule en haut	<code>\stemUp e'4 \stemNeutral e'4^[]</code>	
Forcer les hampes de plusieurs notes non attachées (noires et blanches)	<code>\stemUp e'4 f g f \stemNeutral e'4^[] f^[] g^[] f^[]</code>	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 <code>e'4^[f g f]</code> 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.
	<code>e'4^[f g b, d f]</code>	
	<code>e'4^[] f^[] g^[] b,^[] d^[] f^[]</code>	
Forcer une hampe seule en bas	<code>g4_[]</code>	

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>`

Anacrouse


Démarrage en levée de la mélodie, sans utiliser de silences invisibles avant (r) :

`\partial <durée de l'anacrouse>`

Changement de positions des éléments





1	<code>\slurUp</code>	<code>\slurDown</code>	<code>\slurNeutral</code>	Lignes de liaison
2	<code>\stemUp</code>	<code>\stemDown</code>	<code>\stemNeutral</code>	Hampes de notes

Voix simultanées

Objet	Code	Description
	<code><< { note note note } \ { note note note } >></code>	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
Version simplifiée	<code><< note note // note note >></code>	C'est la version simplifiée de la précédente.
Exemple	<code><< { e'2 f e f } \ { c,4 g' d g a e' d c } >></code>	
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	<code>\grace note note</code> <code>\gr(note) note</code>	
Exemple	<code>\grace ais'16 b4</code> <code>\gr(ais'16) b4</code>	
Non liées barrées	<code>\slashedGrace note note</code> <code>\gr(note/)</code>	
Exemple	<code>\slashedGrace ais'16 b4</code> <code>\gr(ais'16/)</code>	
Liées non barrées	<code>\appoggiatura note note</code> <code>\gr(note-)</code>	
Exemple	<code>\appoggiatura ais'16 b4</code> <code>\gr(ais'16-)</code>	
Liées barrées	<code>\acciaccatura note note</code> <code>\gr(note/-) note</code>	
Exemple	<code>\acciaccatura ais'16 b4</code> <code>\gr(ais'16/-) b4</code>	
Quand plusieurs notes	<code>\grace note[note note note]</code>	

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 :

1	--piano
---	---------

```

2  --barres
3  --times
4
5  # La définition de la première mesure
6  # <nom de la variable-définition>
7  # <notes de main droite>
8  # <notes de main gauche>
9  mes1=
10 c'2 e4 g
11 c8 g' e g c, g' e g
12
13 mes2=
14 b'4.( c16 d) c4 r
15 d8 g f g d g f g
16
17 # Un segment comprenant ces deux mesures se définirait par :
18 -> mesures-1-a-2
19 mes1 mes2
20 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 :

```

1
2 mesure1=
3 c d e f g a b c
4 c b a g f e d c
5
6 # La ligne vide ci-dessus met fin à la définition

```

Variable dynamique

Depuis 2024, les variables sont « dynamique », c'est-à-dire qu'elles peuvent varier en fonction de paramètres. À commencer par leur hauteur.

Hauteur de la variable

Par exemple, soit la variable `maVar` définie par :

```
1 | maVar=  
2 | c d e f g
```

Alors si on utilise dans le code :

```
1 | -> partition  
2 | maVar' maVar maVar,
```

Cela produira les notes :

```
1 | \relative c' { c' d e f g } \relative c' { c d e f g } \relative c' {  
   c, d e f g }
```

C'est-à-dire que les marques « ' » et « , », comme pour les notes, permettent de définir la hauteur où sera joué la variable.

Répétition de la variable

De la même manière, on peut définir le nombre de fois où la variable doit être répétée avec `*N`. Par exemple :

```
1 | -> partition  
2 | maVar*4
```

Produira :

```
1 | \relative c' { c d e f g c d e f g c d e f g c d e f g }
```

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 :

```
1 mes1==
2 ... notes ...
3
4 mes2==
5 ... notes ...
6
7 mes3==
8 ... notes ...
9
10 mes4==
11 ... notes ...
```

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

```
1 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 :

```
1 # Un extrait avec définition d'une seule main
2
3 --piano
4
5 mg1=
6 c1 e g
7
8 md1=
9 g8( a b c ) c2
10
11 -> essai_par_mains
12 mg1
13 md1
```



```
14
15 mgd1=
16 c1 e g
17 g8( a b c) c2
18
19 -> essai_deux_mains
20 mgd1
21
```

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)

```
1
2 md1=
3 c2 e4 g
4
5 mg1=
6 g8( a b c) c2
7
8 md2=
9 b4. c8 c4 r
10
11 mg2=
12 d g f g c, g' e g
13
14 # Définitions à voix multiples
15 tout=
16 md1 md2
17 mg1 mg2
18
19 -> fichier
20 tout
21
```

Annexe

Le fichier « build » dans Sublime Text

Note : je n'ai pas réussi à le faire remarcher, même en modifiant la 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.

```
1 // "shell_cmd": "make"
2 "cmd": [
3   "/Users/philippeperret/.rbenv/versions/2.7.1/bin/ruby",
4   "/Users/philippeperret/Documents/ICARE_EDITIONS/Musique/xDev/scripts/music-score/music-score.rb",
5   "$file"
6 ],
7 "selector": "source.music",
8 "file_patterns": ["*.mus"],
9 "target": "ansi_color_build",
10 "syntax": "Packages/ANSIescape/ANSI.sublime-syntax"
11 }
```

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

```
1 %YAML 1.2
2 ---
3 # See http://www.sublimetext.com/docs/3/syntax.html
4 file_extensions:
5   - mus
6 scope: source.music
7
8 contexts:
```

```

9      # The prototype context is prepended to all contexts but those
setting
10     # meta_include_prototype: false.
11     prototype:
12         - include: comments
13
14     main:
15         # The main context is the initial starting point of our syntax.
16         # Include other contexts from here (or specify them directly).
17         - include: numbers
18         - include: notes
19
20     numbers:
21         - match: '[^,a-g][0-9]+(\-[0-9]+)?\b'
22           scope: constant.numeric.example-c
23
24     notes:
25         # Les notes
26         - match: "\\b[a-gr](es|is)?(es|is)?[',](16|32|64|128|1|2|4|8)?"
27           scope: music.note
28         - match: "\\b[a-gr](es|is)?(es|is)?\\b?[',]?
(16|32|64|128|1|2|4|8)?\\b"
29           scope: music.note
30         # Les accords
31         - match: '<.+?>(16|32|64|128|1|2|4|8)?'
32           scope: music.note.chord
33         # Les liaisons
34         - match: '\\( .+?\\)'
35           scope: music.note
36         # Les options qu'on peut trouver
37         - match: '\\-\\-
(verbose|keep|barres|time|piano|only_new|stop|start|open|big-|mini-
|biggest)(hspace)?( (OFF|ON))?\b'
38           scope: music.option
39         # Numéro de mesure et proximité
40         - match: '\\-\\-(mesure|proximity) [0-9]+\b'
41           scope: music.mesure.numero
42         # Tonalité
43         - match: '\\-\\-(tune|key) [a-zA-Z][#b]?'
44         # Format de page
45         - match: '\\-\\-page [a-zA-Z0-8]+'
46           scope: music.mesure.numero
47         # Définition (variable)
48         - match: '[a-zA-Z0-9_]+\\=\\=?'
49           scope: music.mesure.numero

```

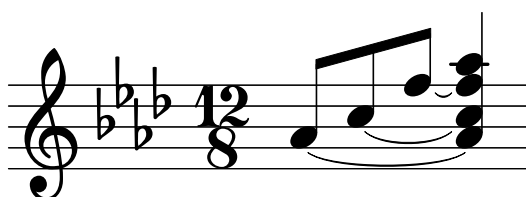
```

50     # Nom de fichier
51     - match: '^\\-> (.+)$'
52     scope: music.output.name
53     # Les clés
54     - match: '\\clef? ("
      (treble|bass|ut)"|G1|G|F3|F|UT1|UT2|UT3|UT4|UT5) '
55     scope: music.clef
56     # Les barres de mesure
57     - match: ':?\\|:?'
58     scope: music.mesure.barre
59     # Special words
60     - match: '\\
      (trill|slashedGrace|grace|appoggiatura|acciaccatura|break) ?'
61     scope: music.clef
62
63     inside_string:
64     - meta_include_prototype: false
65     - meta_scope: string.quoted.double.example-c
66     - match: '\\.'
67     scope: constant.character.escape.example-c
68     - match: '"'
69     scope: punctuation.definition.string.end.example-c
70     pop: true
71
72     comments:
73     # Comments begin with a '#' ' and finish at the end of the line.
74     - match: '# '
75     scope: punctuation.definition.comment.example-c
76     push:
77     # This is an anonymous context push for brevity.
78     - meta_scope: comment.line.double-slash.example-c
79     - match: $\\n?
80     pop: true
81

```

Étapes pour « arpège vers accord tenu »

L'image suivante :



... est obtenue à partir du code :

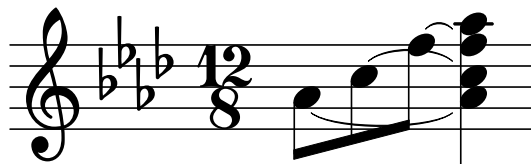
```
1 | \set tieWaitForNote = ##t \stemUp aes'8~ \tieDown c~ f~ <aes, c f aes>4
```

Voir la [version simplifiée](#).

Pour obtenir ce code, nous avons procédé ainsi. Nous sommes partis de :

```
1 | \set tieWaitForNote = ##t aes'8~ c~ f~ <aes, c f aes>4
```

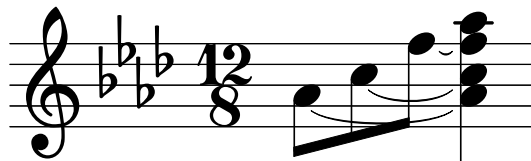
... qui a produit :



Pour avoir les liaisons dans le bon sens avec `\tieDown`, nous avons fait :

```
1 | \set tieWaitForNote = ##t aes'8~ \tieDown c~ f~ <aes, c f aes>4
```

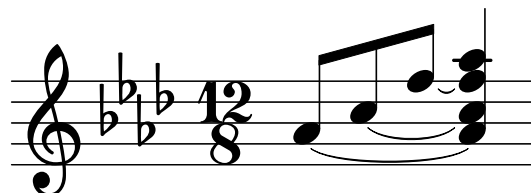
... qui a produit :



Et enfin nous avons mis les hampes dans le bon sens avec `\stemUp` :

```
1 | \set tieWaitForNote = ##t \stemUp aes'8~ \tieDown c~ f~ <aes, c f aes>4
```

... qui a produit :



Tests

Score-Image est testée par des comparaisons entre les images attendues et les images produites, pour être sûr d'un résultat optimum. Cf. le dossier `tests/checksums_tests` qui contient tous les fichiers.

Pour lancer tous les tests :

```
1 | score-image tests _
```

Pour filtrer les tests à jouer :

```
1 | score-image tests /<filtre régulier>/
```

Pour ne lancer que les tests d'un dossier :

```
1 | score-image tests _ -dir=dossier
```

Et en les filtrant :

```
1 | score-image /ceuxla/ -dir=mon/dossier
```

En plus de l'image produite, les tests s'assurent aussi que toutes les statistiques soient correctes, ce qui permet un tour d'horizon parfait.

Pour de plus amples informations, lire le fichier `tests/_ReadMe_.md`