*ly*LuaTEX

1.1.8

Fr. Jacques Peron Urs Liska† Br. Samuel Springuel

2026-11-24

†Deceased, 2024.

Contents

Introduction **2**
- Installation and Requirements . . . 3
  - Prerequisites . . . 3
  - TeXLive and MiKTeX . . . 3
  - Latest version . . . 4
  - For a single document . . . 4

Usage **5**
- The Big Picture and caveats . . . 5
- Basic Operation . . . 6
- Option Handling . . . 8
- System-by-System, Fullpage, and Inline Scores (Insertion Mode) . . . 9
  - System-by-System . . . 9
  - Fullpage . . . 10
  - Inline . . . 11
  - Choosing Systems/Pages . . . 11
  - Choosing a range of measures . . . 12
- Score Layout . . . 12
  - Dimensions . . . 12
    - General . . . 13
    - Fullpage . . . 14
  - Alignment . . . 14
    - Protrusion (system-by-system Scores) . . . 14
    - Managing indentation . . . 16
    - Vertical Alignment of Fullpage Scores . . . 18
- Score Options . . . 18
  - Automatic Wrapping of Music Expressions . . . 18

Font Handling . . . 19
Staff Display . . . 20
Relative or Absolute Pitches . . . 21
Input Language . . . 21
Labels . . . 21
Printing the Filename . . . 22
Printing LilyPond Code . . . 22
Miscellaneous Options . . . 23
Include Paths . . . 23
LilyPond Executable . . . 24
Temporary Directory for Scores . . . 24
Writing headers to include file . . . 25
PDF optimization . . . 26
Handling LilyPond Failures . . . 26
Forcing (Re-)Compilation . . . 26
Bug workaround . . . 27
MusicXML options . . . 27

Using *ly*LuaTEX in Classes or Style Files **27**
Wrapping *ly*LuaTEX commands . . . 27
Providing Raw filenames . . . 27

Index **31**

Examples **31**
Insert Systems . . . 31
Insert Inline . . . 33
Alignment and padding . . . 34
Bare Inline scores . . . 34
Choosing Systems . . . 34
Dynamic Indent Handling . . . 38
Right protrusion . . . 41
Performance considerations . . . 41
Font Handling . . . 41
Wrapping Commands . . . 44
Wrapping Raw PDF Filenames . . . 48

Introduction

*ly*LuaTEX is a comprehensive LuaLATEX package to manage the inclusion of musical scores in text documents. It uses the GNU LilyPond[1] score writer to produce beautiful music elements in beautifully typeset text documents. *ly*LuaTEX supports a wide range of use cases and lends itself equally well to authoring musicological texts with music examples and preparing musical editions

[1] http://lilypond.org

with interspersed text parts, to creating song booklets used in service and to provide work sheets for teaching and exams.

*ly*LuaTEX is inspired by and provides a fully compatible drop-in replacement to lilypond-book, a LATEX document preprocessor shipping with LilyPond, which it actually is a *superset* of. However, thanks to the use of LuaLATEX it can overcome substantial usability limitations of the scripted solution and provide numerous additional features.

*ly*LuaTEX's main features include:

- Using LilyPond to compile musical scores directly from within the LATEX run
- Intelligent caching of engraved scores, avoiding recompilation when possible
- Matching of layout and appearance to perfectly fit the scores into the text document
- Comprehensive configuration through global and per-score options

What *ly*LuaTEX does *not* try to do is managing the handling of floating environments, counters, and lists of music examples. The *ly*LuaTEXMP package[2] is currently under construction and practical testing and will eventually be released to become a suitable wrapper for using *ly*LuaTEX to create numbered music examples.

Installation and Requirements

Prerequisites

As the name *ly*LuaTEX implies this package can only be used with the LuaLATEX engine. For more information on this please refer to Usage below.

Musical scores are created in real-time (instead of incorporating pre-built *image* files) using the GNU LilyPond[3] score writer, so of course this has to be installed too. *ly*LuaTEX should work with any versions of LilyPond, but it has been developed against the stable and development versions that were current at the time of this writing: 2.18.2 and 2.19.83.

TeXLive and MiKTeX

*ly*LuaTEX is included in both the TEX Live and MiKTeX LATEX distributions and can be installed through their package management systems. In TEX Live it is included in the `texlive-music` collection and – of course – in `texlive-full`. If neither of these collections is installed *ly*LuaTEX can be added to a TEX Live installation by running

```
tlmgr install lyluatex
```

[2] https://github.com/uliska/lyluatexmp
[3] http://lilypond.org

from the command line.

MiKTEX users normally do not need to install *ly*LuaTEX manually if on-the-fly package installation is enabled. MiKTEX can install the LATEX package automatically when `\usepackage{lyluatex}` is encountered. However, LilyPond itself remains an external dependency and must be installed/configured separately.

Latest version

For normal use, use the version of *ly*LuaTEX provided by your TEX distribution, such as TEX Live or MiKTEX. Source code, issue reports, and development versions are available from the project repository.[4]

Copy `lyluatex.sty` and `lyluatex.lua` from this repository into your `$TEXMFHOME` tree, and place them both in the appropriate directory.

- `lyluatex.sty` goes into `$TEXMFHOME/tex/lualatex/lyluatex/`
- `lyluatex.lua` goes into `$TEXMFHOME/scripts/lua/lyluatex/`

In many installations, `$TEXMFHOME` defaults to `$HOME/texmf`. Run `kpsewhich --var-value=TEXMFHOME` to determine its exact location.

After copying the files, verify which version will be used:

```
kpsewhich lyluatex.{sty,lua}
```

Note:

It may be useful to clone the Git repository not into the `$TEXMFHOME` tree directly but to some arbitrary location and link to that. Please note that LuaLATEX will only follow such symbolic links if there is at least one *real* subdirectory in each directory. So if there is a directory `$TEXMFHOME/tex/lualatex` containing *only* symbolic links it is necessary to create a dummy subdirectory in it.

For a single document

Copy `lyluatex.sty` and `lyluatex.lua` into the folder containing the document you wish to typeset.

[4] https://github.com/jperon/lyluatex

Usage

The Big Picture and caveats

Once *ly*LuaTEX is loaded it provides commands and environments to include musical scores and score fragments which are produced using the GNU LilyPond score writer. They are encoded in LilyPond input language, either directly in the `.tex` document or in referenced standalone files. *ly*LuaTEX will automatically take care of compiling the scores if necessary – making use of an intelligent caching mechanism –, and it will match the score's layout to that of the text document. *ly*LuaTEX will produce PDF image files which are automatically included within the current paragraph, in their own paragraphs or as full pages. The behaviour of *ly*LuaTEX and the appearance of the resulting scores are highly configurable through package, global, and per-score options which are globally described in the Option Handling section below and in detail throughout the rest of this manual.

*ly*LuaTEX aims at being an upwards-compatible drop-in replacement for the lilypond-book preprocessor shipping with LilyPond.[5] which means that any documents prepared for use with `lilypond-book` should be directly usable with *ly*LuaTEX, with some caveats:

- `fragment` is the default: see Automatic wrapping for more details about this;
- *ly*LuaTEX has an option `insert`, which defaults to `systems` for `\begin{lilypond} \end{lilypond}`, but to `inline` for `\lilypond`; the last one by default reduces staff size and includes only the first system if there are several ones;
- `\musicxmlfile` has `no-articulation-directions`, `no-beaming`, `no-page-layout` and `no-rest-positions` set to `true` by default, to increase chances of getting something acceptable. Nevertheless, please read the note about this command below.

So, if you want *ly*LuaTEX to mimic as much as possible lilypond-book, you should load it with options as follows: `\usepackage[nofragment, insert=systems]{lyluatex}`.

Note:

By default *ly*LuaTEX invokes LilyPond simply as `lilypond`. If LilyPond is installed in another location or a specific version of LilyPond should be used the invocation is controlled with the `program` option, see The LilyPond Executable.

Note: *ly*LuaTEX can only be used with LuaLaTEX, and compiling with any other LaTEX engine will fail.

Note: In order to avoid unexpected behaviour it is strongly suggested that documents are generally compiled from their actual directory, i.e. without referring to it through a path. This is because in many places during the compilation process relative paths are calculated from this starting point. Building *out-of-tree* isn't supported, though it should be possible with the following workarounds:

1. `includepaths={..}`: (for example, if you build from a subdirectory of main directory) tell lyluatex to search the parent directory;

[5] http://lilypond.org/doc/v2.18/Documentation/usage/lilypond_002dbook

2. if the book contain figures, enter only the name of the files, then set two paths, e.g.: `\graphicspath{{images/}{../images/}}`, so images will be found either way.

NOTE: *ly*LuaTEX requires that LuaLATEX is started with the `--shell-escape` command line option to enable the execution of arbitrary shell commands, which is necessary to let LilyPond compile the inserted scores on-the-fly and to perform some auxiliary shell operations. However, this opens a significant security hole, and only fully trusted input files should be compiled. You may mitigate (but not totally remove) this security hole by adding `lilypond` and `gs` to `shell_escape_commands`, and using `--shell-restricted` instead of `--shell-escape`: look at the documentation of your TEX distribution. For example, on Debian Linux with TeXLive:

```
% export shell_escape_commands=$(kpsewhich -expand-var '$shell_escape_commands'),lilypond,gs
% lualatex --shell-restricted DOCUMENT.tex
```

Basic Operation

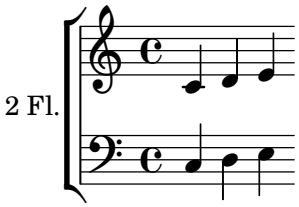
*ly*LuaTEX is loaded with the command `\usepackage{lyluatex}` which also accepts a number of `key=value` options. Their general use is described in the Option Handling section below. If LilyPond can be invoked through `lilypond` on the given system using the package without any options already provides a usable system.

lilypond

The basic mode of inserting scores into text documents is the `lilypond` environment:

```
\begin{lilypond}
music = \relative {
  c d e
}

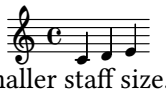
\score {
  \new ChoirStaff \with {
    instrumentName = "2 Fl."
  } <<
    \new Staff \transpose c c' \music
    \new Staff {
      \clef bass
      \music
    }
    >>
}
\end{lilypond}
```



*ly*LuaTEX will now collect the given content and wrap it in additional LilyPond code to create the layout and appearance according to the text document and the user's configuration. The resulting file is compiled with LilyPond and saved in a temporary directory, from where it is included in the text document. A hash value including the full content and all options will be used to determine if the score has already been compiled earlier, so unnecessary recompilations are avoided.

Note: Despite its familiar appearance, this environment is very special, using a mechanism specific to LuaLATEX. One consequence is that you necessarily need a newline after `\begin{lilypond}`, and before `\end{lilypond}`; another is that you have to be careful when you want to wrap this environment in a custom one: see Wrapping *ly*LuaTEX commands and the examples at the end of the manual.

`\lilypond`

Very short fragments of LilyPond code can be entered inline using the `\lilypond` command: `\lilypond{ c' d' e' }`  Fragments specified with `\lilypond` are by default inserted as *inline* scores with a smaller staff size. For further information about the different insertion modes read the section about insertion modes.

`\lilypondfile`

External files of arbitrary complexity can be referenced with

```
\lilypondfile{path/to/file}
```

Absolute and relative paths can be given. Relative paths are searched in the following order:

- relative to the current file's directory
- relative to all given include paths (see LilyPond Include Paths))
- relative to all paths visible to LATEX (like the package search)

`\musicxmlfile` Finally there is a command to include scores encoded as MusicXML files. These will be converted to LilyPond input by LilyPond's `musicxml2ly` script and then compiled by LilyPond.

Note:

This command has been added to provide compatibility with `lilypond-book`, but it is discouraged to use it since its use implies substantial problems:

- The conversion process with `musicxml2ly` is somewhat fragile and can crash in unpredictable ways due to encoding problems between various versions of Python and Lua involved

- `musicxml2ly` itself doesn't provide totally reliable conversion results, even if the conversion reports successful operation. In this case LilyPond may produce inferior results or may fail to compile the score completely

If there is the need to include music scores that are only available as MusicXML files it will nearly always be the better option to independently convert the source using `musicxml2ly` and then manually post-process the resulting Lilypond input files.

Option Handling

All aspects of *ly*LuaTEX's behaviour can be configured in detail through *options*. Through a unified interface all options can be set as *package options* or as *local options*, and they can be changed anywhere in the document. Note that not each approach is suitable for every option: the option to clean the temporary directory only makes sense as a package option for example, or you can't reasonably apply a label other than locally to a single score.

All options are `key=value` options, and options that are *not* set explicitly will use their default value which is documented with each option. Boolean options don't have to be set to `true` explicitly, using the option alone will do that as well, for example: `[debug=true]` is equivalent to `[debug]`.

Options can be unset (i.e. reset to their default value) through the syntax `key=`. This can for example be used to use the default value locally when a value has been specified globally.

Some options are complemented by a corresponding `no<option>`. Using this alternative is equivalent to setting an option to `false`: `nofragment` is the same as `fragment=false`.

Finally it has to be mentioned that some options have side-effects on other options. For example, setting `indent` to some value implicitly will set `autoindent=false`, or `max-protrusion` will define `max-left-protrusion` and `max-right-protrusion` if these are not set explicitly.

Package Options

Options can be set globally through package options, which are used with

```
\usepackage[key1=value1,key2]{lyluatex}
```

Local Options

Options can also be applied on a per-score basis through optional arguments to the individual command or environments:

```
\lilypondfile[key1=value1]{path/to/file.ly}

\lilypond[key1=value1]{ c' d' e' }

\begin{lilypond}[key1=value1]
{
  c' d' e'
}
```

```
\end{lilypond}
```

`\setluaoptionly`

At any place in the document the value of an option can be changed using

```
\setluaoption{ly}{key}{new-value}
```

The option will take effect from now on as a package option until it is changed again. Note that this may or may not make sense with a given option. For example the `tmpdir` option should only be modified in very special and sophisticated set-ups.

Local options will override this value as with package options.

System-by-System, Fullpage, and Inline Scores (Insertion Mode)

`insert` *(systems)*

Scores can be included in documents in three basic modes: system-by-system, fullpage, and inline. The system-by-system mode is the default and includes a score as a sequence of images, one for each system. This allows LaTeX to have the systems flow over page breaks and to adjust the space between systems to vertically justify the systems on the page.

Insertion mode can be controlled with the `insert` option, whose valid values are `systems` (default), `fullpage`, `inline`, and `bare-inline`.

System-by-System

`insert=systems`

With this default option each score is compiled as a sequence of PDF files representing one system each. By default the systems are separated by a paragraph and a variable skip depending on the staffsize.

Note:

`insert=systems` implies the use of `lilypond-book-preamble.ly`. It is worth pointing out that the score will *not* have any notion of pages anymore - resulting in the staff-staff spacing to be minimal/natural. Usually LilyPond will space out the inter-staff (not -system!) space when the page is not filled, but with `insert=systems` this will not happen. While this behavior is usually desirable when including score examples in text, it may result in suboptimal output for multi-page scores, when there's the typical issues of how many systems will fit on a page.

Also notice that by default pages will be ragged-bottom, and LilyPond will not make any efforts to optimize page breaks. `\betweenLilyPondSystem` can be used when the space between systems seems too tight, for example using something like `\vfill`.

`\betweenLilyPondSystem`

If a macro `\betweenLilyPondSystem` is defined it will be expanded between each system. This macro is documented in LilyPond documentation. It must accept one argument, which will be the number of systems already printed in the score ('1' after the first system). With this information it is possible to respond individually to systems (e.g. "print a horizontal rule after each third system" or "force page breaks after the third and seventh system"). But a more typical use case is to insert different space between the systems or using simple line breaks while ignoring the system count:

```
\newcommand{\betweenLilyPondSystem}[1]{\linebreak}
```

`\preLilyPondExample, \postLilyPondExample`

If either of these macros is defined it will be expanded immediately before or after the score; this may for example be used to wrap the example in environments, though there probably are better ways to do so. With `verbatim`, `\preLilyPondExample` will take place after the verbatim block, just before the score.

Examples:

For a demonstration of the system-by-system options see Insert Systems.

Fullpage

`insert=fullpage`

With `insert` set to `fullpage` the score is compiled to a single PDF file that is included through `\includepdf`. The layout of such scores can be configured through a number of alignment options.

`fullpagestyle` *()*

`print-page-number` *(false)*

These two options work together basically deciding who is responsible for printing headers and footers, LilyPond or LaTeX. `fullpagestyle` is equivalent to LaTeX's `\pagestyle` and accepts anything that the current pagestyle can be set to. By default the current pagestyle will be continued throughout the score. *NOTE:* This is different from the usual behaviour of `\includepdf` which sets the pagestyle to `empty`. So by default LaTeX will continue to print headers and footers, including page numbers.

`print-page-number` decides whether LilyPond prints page numbers in the score. By default this is set to `false`, so the default setting of these two options means that LilyPond does *not* print page numbers while LaTeX continues to print headers and footers.

The same may be achieved only for first page with `print-first-page-number`.

`first-page-number` *(false)*

Normally, *ly*LuaTeX should automatically determine the first page number of the score to match its place in the document. Should you like to force it to another value, you may do it thanks to this option.

Inline

`insert=inline|bare-inline`

With `insert=inline` or `insert=bare-inline` scores can be included *within* paragraphs. They are basically the same with regard to the inclusion, but `bare-inline` implicitly sets the `nostaff` option to suppress staff symbol, time signature and clef.

`inline-staffsize` *(default)*

By default the staff size of inline scores is determined as 2/3 of the default staff size of regular scores, so the effective size of an inline score will depend both on the text's font size and the current `staffsize` setting. The `inline-staffsize` option sets an absolute staffsize in `pt` (omitting the "`pt`").

`valign` *(center)*

Controls the vertical alignment of the score against the current line of text. The default value `center` will align the vertical center of the image to a virtual line `1/2em` above the baseline of the text. `top` will align the top edge of the image with the X-height of the text (actually: `1em` above the baseline). `bottom` aligns the bottom of the image with the text baseline.

Note: The alignment works with the edges of the *image file*, there is no notion of an "optical" center or aligning with the staff lines.

`voffset` *(0pt)*

Can be used to *add* a vertical offset to the automatic alignment.

`hpadding` *(0.75ex)*

Inserts some space to the left and right of the included score (except at line start or end).

Examples:

Examples can be found in Insert Inline.

Choosing Systems/Pages

`print-only` *()*

With the option `print-only` it is possible to choose which pages or systems of a score will be included in the document. This can for example be used to comment on individual parts of a score without having to specify them – potentially redundantly – as separate scores. Another use case is printing a selection of scores from a PDF containing multiple scores, such as a song book for example.

Depending on the setting of the `insert` option this will affect systems or pages. The selection of systems/pages can be specified as

- `<empty>` (default): include the whole score

- a single number: include a single page/system
- a range of numbers: include a range of pages/systems `{M-N}` or `{N-M}` (to print backwards)
- the special range `N-`, including all systems/pages from N throughout the end
- a comma-separated list of numbers or ranges `{A,B , C,D-E, F, C- B}` (freely mixed, in arbitrary order)

Note:

It is the user's responsibility to only request pages/systems that are actually present in the score, otherwise LaTeX will raise an error.

Examples:

Usage examples for this option can be found in Choosing Systems.

Choosing a range of measures

`clip-regions` *()*

The option `clip-regions` extracts specific measure ranges as clipped snippets using LilyPond's `-dclip-systems`. Values use inclusive range syntax (e.g., `clip-regions=2-14` includes only measures 2 through 14). The fragment is forced to *begin a new system*. Usage with `ragged-right=true` is recommended. This option requires LilyPond 2.24 or newer; with older versions, *ly*LuaTeX stops with an error.

Score Layout

One of the most obvious features of *ly*LuaTeX is its ability to configure the layout and appearance of LilyPond scores from within the `.tex` document. Without further configuration *ly*LuaTeX will try to match the score as closely as possible to the layout of the surrounding text, but there are numerous options to tweak the layout in detail.

Dimensions

If not stated otherwise dimensions can be given in arbitrary TeX units, e.g. `200pt`, `1ex` or `3cm` or as TeX lengths, e.g. `0.4\textwidth`.

General`line-width` *(default)*

Set the line width of the score. By default this exactly matches the current actual line width of the text, which also works in multicolumn settings. See Alignment for a discussion of the details of the alignment of staves to the text.

`staffsize` *(default)*

Set the staffsize of the score. By default (`[staffsize=default]`, `[staffsize]` or simply omitted) this is calculated relative to the size of the current text font, so it will give a consistent relation to the text at any font size. Absolute sizes can be given as a number, which is interpreted as `pt`. For example LilyPond's own default staff size is `20`.

`ragged-right` *(default)*

Set the score to ragged-right systems. By default, single-system scores will not be justified but printed at their "natural" width, while scores with multiple systems by default will be justified. With this option set to true, all systems are printed at their natural width; with this option set to false, all systems are justified (even for single-system scores). `noragged-right` is equivalent to `raggedright=false`.

`indent` *()*

Defines indentation of first system (same as LilyPond's `indent`). By default, with `insert=fullpage`, scores are indented; otherwise, they aren't. `noindent` is equivalent to `indent=0pt`. Please also see the section about Dynamic Indentation.

`system-count` *()*

Forces LilyPond to produce a fixed number of systems. This may be useful when LilyPond breaks a score that can manually be squeezed to one system less, but it is also possible to spread out a score to more systems than LilyPond would consider necessary.

`quote` *(false)*

This option, which is there for compatibility with `lilypond-book`, reduces line length of a music snippet by $2 \times 0.4\,in$ and puts the output into a quotation block. The value $0.4\,in$ can be controlled with following options.

This option isn't intended to be used with `\insert=fullpage`, and won't give a good result with it.

`gutter, leftgutter, rightgutter, exampleindent` *($0.4\,in$)*

`leftgutter` control the supplementary left margin of a "quoted" score, `rightgutter` the right margin. If not set, they're automatically set to `gutter` value; `exampleindent` is an alias for `gutter` (for compatibility with `lilypond-book`).

Fullpage There are several options that can change the basic page layout of full-page scores. However, by default all these options inherit their values from the .tex document, and there should very rarely be the need to explicitly change the values for these options.

`papersize` *(default)*

By default the LilyPond score will have the same paper size as the text document, but it is possible to override this with the `papersize` option. It accepts any paper sizes that are predefined in LilyPond[6], it is not possible to use custom paper sizes.

`paperwidth` (`\paperwidth`)

`paperheight` (`\paperheight`)

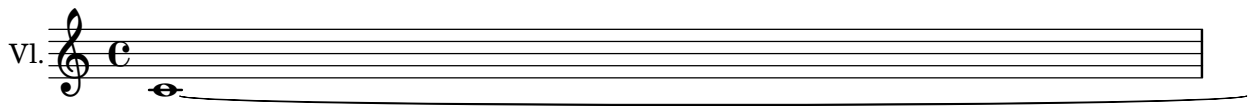
`twoside` *(default)*

Note:

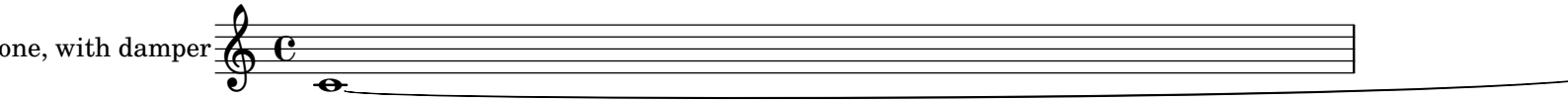
If `papersize` is set, any values of `paperheight` and `paperwidth` are ignored.

Alignment

Protrusion (system-by-system Scores) The reference for the horizontal alignment of scores included system-by-system is the *staff symbol*. By default *ly*LuaTEX aligns the two ends of the staff symbol with the current `\linewidth`, and any score items that exceed the staff lines to the left or right will protrude into the page margin(s):



This is also how LilyPond handles margins (and the only option when including scores with `insert=fullpage`). However, *ly*LuaTEX provides a configurable limit to guard against excessive protrusion. By default this is effectively "disabled" by being set to the length `\maxdimen`, so protruding elements may be cut off at the page border:



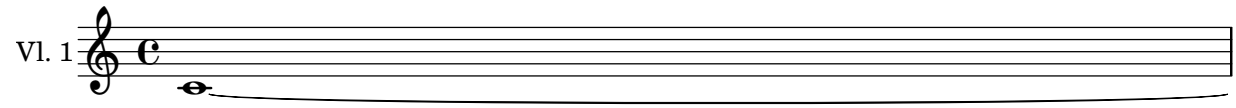
[6] see the manual page at http://lilypond.org/doc/v2.18/Documentation/notation/predefined-paper-sizes

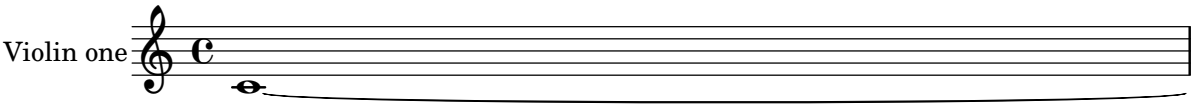
`max-protrusion` *(`\maxdimen`)*

`max-left-protrusion` *(default)*

`max-right-protrusion` *(default)*

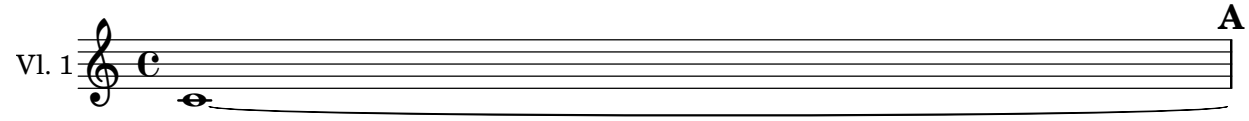
These options set the protrusion limit. If either of the `-left-` or `-right-` options is unset then the value will be taken from `max-dimension`. Note that this is not a fixed value for the protrusion but a *limit*, so it will only have an effect when the actual protrusion of the score exceeds the limit. In a way it can be understood as a dynamic variant of the `quote` option, something like a "fence". The following two scores have the same `max-left-protrusion=1cm`, but only the second is modified.



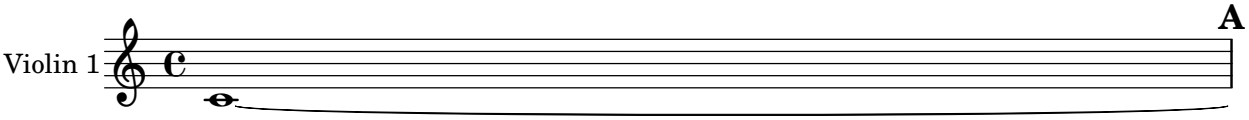


When the protrusion limit kicks in the score will be offset to the right by the appropriate amount, and if necessary it will be shortened to accomodate the right edge with its individual protrusion limit. *ly*LuaTEX will automatically ensure both that the staff line doesn't exceed the text and that the protruding elements don't exceed the limit. The following three scores demonstrate that behaviour with `max-protrusion=1cm`.

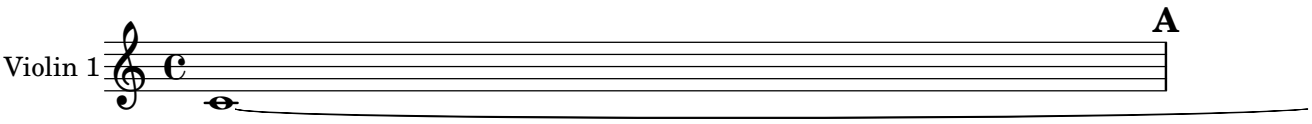
The first score has elements that protrude less than the limit, so nothing is modified:



In the second score the longer instrument name makes the system shift to the right. The rehearsal mark still protrudes into the margin but is below the threshold. The score will automatically be recompiled with a narrower line width to ensure the staff lines don't protrude into the right margin.

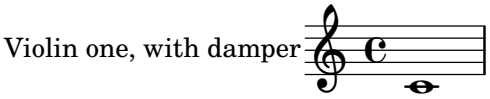


In a third score the tie has been tweaked to protrude into the margin and exceed the limit. As a result the score is narrowed even further, also shifting the *right* margin.



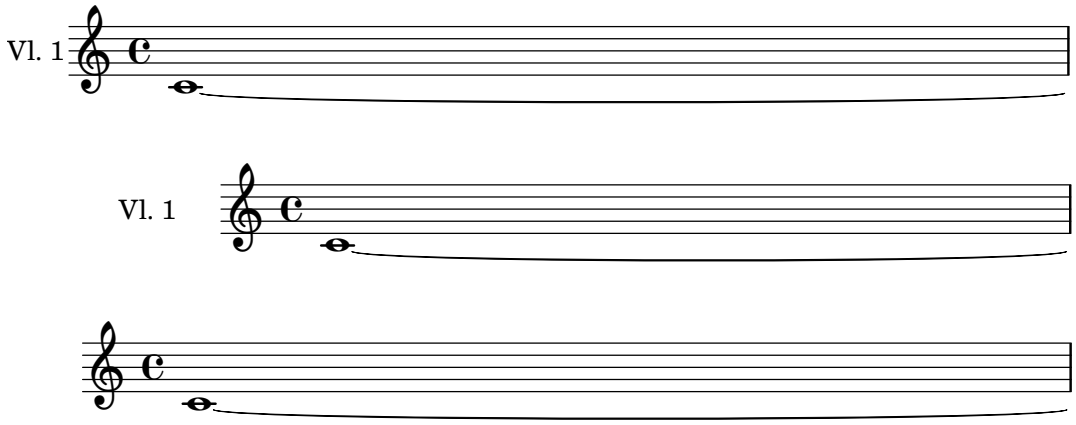
Note that this is not achieved by *scaling* the PDF file but by actually *recompiling* the score with modified `line-width`, thus keeping the correct staffsize. A warning message will inform about that fact on the console and in the log file.

Note further that in the final example the score is short enough to fit on the line even with the horizontal offset, so in this case there is no need to recompile a shortened version:



Negative max-protrusion

The protrusion limits can also be set to *negative* lengths, which makes them behave similar to using the `quote` option. However, there is a substantial difference between the two: using `quote` will apply a fixed indent, and the reference will again be the staff lines. Any protrusion will be considered from that reference point, so protruding elements will protrude into the margins, starting from the indent. Using a negative protrusion limit instead will prevent *any* part of the score to exceed that value. Twe following three scores demonstrate the difference: the first has `quote, gutter=0.4in` while the second has `max-protrusion=-0.4in` set. The third has the same protrusion limit as the second but no protruding elements.



Managing indentation`indent` *(false)*

As mentioned above `indent` controls the indentation of the first system in a score. However, *ly*LuaTEX provides smart dynamic indent handling for `insert=systems` that goes beyond simply setting the `indent` in the LilyPond score.

Deactivating indent

The indent is deactivated if one of the following condition is true:

- The score consists of a single system
- Only the first system of a score is printed using `print-only=true`
- `print-only` is set so the first system of a score is printed but not in the first position.

In the first case the score is simply shifted left, but in the other cases the score is recompiled to avoid a "hole" at the right edge.

`autoindent` *(true)*

When `autoindent` is active protrusion handling will be modified. If a given protrusion limit is exceeded *ly*LuaTEX will not reduce the `line-width` of the *whole* score but add an indent. This is because in many cases it is the *first* system of a score that contains significant protruding elements. If after application of the indent the protrusion limit is still exceeded due to other systems the line width is only reduced by the necessary amount and the indent adjusted accordingly.

`autoindent` is active by default but will be deactivated if `indent` is set.

`autoindent` will also be applied when a given indent is deactivated as described in the previous paragraph. This is done in order to avoid the whole score to be narrowed because of the deactivated indent.

Note:

Handling automatic indent requires up to three recompilations of a score, but it will only be applied when a protrusion limit is given and exceeded. Intermediate scores are cached and won't be unnecessarily recompiled.

NOTE:

Cacluations regarding automatic indent rely on the High Resolution Bounding Box retrieved from the final PDF file of the score. This is done using Ghostscript with the gs invocation. If this should not be available a low resolution bounding box is used instead, which can lead to rounding errors. Note that under certain circumstandes these rounding errors may not only lead to less accurate alignment but to wrong decisions in the alignment process. If you encounter wrong results please try to create a Minimal Working Example and submit it to *ly*LuaTEX's issue tracker[7].

Examples:

A comprehensive set of examples demonstrating the dynamic indent behaviour is available in Dynamic Indent.

[7] https://github.com/jperon/lyluatex/issues

Vertical Alignment of Fullpage Scores`fullpagealign` *(crop|staffline)*

Controls how the top and bottom margins of a score are calculated. With `crop` LilyPond's `margin` paper variables are simply set to those of the LaTeX document, while `staffline` pursues a different approach that makes the outermost *stafflines* align with the margin of the text's type area.

With `crop` the pages may look somewhat uneven because the top and bottom systems are often pushed inside the page because the *extremal* score items are aligned to the text. With `staffline` on the other hand it may happen that score items protrude too much into the vertical margins.

NOTE:

The `fullpagealign=staffline` option is highly experimental and has to be used with care. While positioning the extremal staves works perfectly the approach may confuse LilyPond's overall spacing algorithms. The `stretchability` parameters of `top-system-spacing`, `top-markup-spacing`, and `last-bottom-spacing` are forced to `0`, which seems to "unbalance" the mutual stretches of vertical spacing. When scores appear compressed it is possible to experiment with (a combination of) explicitly setting `max-systems-per-page`, `page-count`, or – if everything else fails – by including manual page breaks in the score.

Another issue with `fullpagealign=staffline` is that it doesn't work properly with `print-page-number`. If these two options are set LilyPond will print the page numbers at the top of the paper, without a margin. But when aligning the stafflines to the type area one will usually want to have LaTeX print the page headers and footers anyway.

`extra-bottom-margin` *(0)*

`extra-top-margin` *(0)*

These options may be used to add (or remove) some space to the vertical margins of fullpage scores. This can be used to create a vertical "indent" or to adjust for scores with unusually large vertical protrusion. *Note:* This setting affects a whole score and can't be applied to individual pages (which is a limitaion with LilyPond).

Score Options

Automatic Wrapping of Music Expressions

`fragment` *(true)*

With this option set to `true`, the input code is wrapped between `{ }`, so that you can directly enter simple code, for example:

```
\lilypond{a' b' c'}
```

This option defaults to `true` with `\lilypond` and `lilypond` environment, to `false` with `\lilypondfile`. It will be automatically disabled if a `\book`, `\header`, `\layout`, `\paper` or `\score` block is found within the input code; but in some cases, it will be necessary to explicitly disable it with `fragment=false` or its equivalent `nofragment`.

`nofragment` and `relative` are mutually exclusive; the locally-defined option will take precedence over the globally-defined one, and if both are defined at the same level, the result will be random.

Font Handling

`pass-fonts` *(false)*

Use the text document's fonts in the LilyPond score.

The choice of fonts is arguably the most obvious factor in the appearance of any document, be it text or music. In text documents with interspersed scores the text fonts should be consistent between text and music sections. *ly*LuaTEX can handle this automatically by passing the used text fonts to LilyPond, so the user doesn't have to worry about keeping the scores' fonts in sync with the text document.

The following steps are taken when `pass-fonts` is `true`: Before generating any score *ly*LuaTEX retrieves the currently defined fonts for `\rmfamily`, `\sffamily`, and `\ttfamily`, as well as the font that is currently in use for typesetting. These fonts are included in the score compiled by LilyPond, but if the LilyPond input explicitly defines fonts in a `\paper {}` block this takes precedence over the automatically transferred fonts.

Note:

So far only the font *family* is used by LilyPond, but it is intended to add support for OpenType features in the future.

Note:

LilyPond handles font selection differently from LuaTEX and can only look up fonts that are installed as system fonts. For any font that is installed in the `texmf` tree LilyPond will use an arbitrary fallback font. Therefore `pass-fonts` defaults to `false`.

`current-font-as-main` *(false)*

Use the font family *currently* used for typesetting as LilyPond's main font.

By default `pass-fonts` matches, roman, sans, and mono fonts, but with `current-font-as-main=false` the font that is *currently* used for typesetting is passed to LilyPond as its "main" roman font. This ensures that the score's main font is consistent with the surrounding text. However, this behaviour may not be desirable because it effectively removes the roman font from the LilyPond score, and it may make the *scores* look inconsistent with each other. Therefore *ly*LuaTEX by default passes the text document's three font families to their directy LilyPond counterparts.

`rmfamily` *()*

`sffamily` *()*

`ttfamily` *()*

The roman, sans, and mono fonts can also be specified explicitly to be passed into the LilyPond document independently from the text document's fonts. If *any* of these options is set `pass-fonts` is implicitly set to `true`. Note that in this case for families that are *not* set explicitly the current text document fonts are used.

If `rmfamily` is set explicitly then `current-font-as-main` is implicitly disabled.

Examples:

Demonstrations of the different font handling features are available in Font Handling.

Staff Display

There are a number of options that directly affect how staves are displayed, basically removing parts of the staff elements. The options can be freely combined, and a few presets have been prepared.

`noclef` *(false)*

Don't print clefs.

`notimesig` *(false)*

Don't print time signatures.

`nostaffsymbol` *(false)*

Don't print staff lines.

`notiming` *(false)*

Don't use any timing information, e.g. don't print automatic barlines.

`notime` *(false)*

Preset: don't print time signatures and don't use timing (`lilypond-book` option).

`nostaff` *(false)*

Preset: suppress staff lines, clefs, and time signatures (but do use timing).

Note that there is no option to suppress key signatures since a key signature is not *implicitly* printed. *If* there should be the need to *have* a key signature and at the same time suppress it, it's reasonable to expect this to be explicitly done in the LilyPond code.

Relative or Absolute Pitches

By default LilyPond input is parsed as-is with regard to pitches. That means pitches are treated as absolute pitches except if the music is wrapped in a `\relative` clause.

`relative` *(0)*

With the `relative` option set the LilyPond input is parsed in *relative* mode, with the option value specifying the starting pitch. Zero (or an empty value) takes the "middle C" as the origin, positive integers refer to the number of octaves upwards, negative integers to downward octaves.

Note:

This deviates from LilyPond's usual behaviour: in LilyPond the "natural" `c` corresponds to C3 in MIDI terminology, while `relative=0` refers to C4 instead. This is in accordance with the use in lilypond-book.

Note:

`relative` is only allowed when the content is automatically wrapped in a music expression (as described in Automatic Wrapping).

Input Language

`language` *()*

Specify the language for LilyPond input, defaulting to LilyPond's default language Dutch.

Labels

`label` *()*

If the `label` option is set a `\label` is inserted directly before the image. The label name is prepended with the value of the `labelprefix` option, so any references to the score have to take that into account.

Note:

It should be obvious but `label` can only be used as a *local* option since multiple identical labels will trigger LaTeX errors.

`labelprefix` *(ly_)*

Sets the prefix to be prepended to each label.

Printing the Filename

For scores included by `\lilypondfile` it is possible to print the filename before the score. This is activated by the

`printfilename` *(false)*

option. It will print the actual filename only, without any path information.

By default the filename is printed in its own unindented paragraph, including `\bigskip` between the text and the score. However, the appearance can be modified by renewing the command

`\lyFilename`

The following redefinition removes any indent and prints the text in monospace:

```
\renewcommand{\lyFilename}[1]{%
\noindent \texttt{#1}\par\bigskip%
}
```

Printing LilyPond Code

`verbatim` *(false)*

Depending on the use case it may be desired to not only include the score into the document but to also print the LilyPond input verbatim. This can be achieved by setting the `verbatim` option to `true`. In this case first the input code will be printed in a `verbatim` environment, followed by the score.

Note:

Please note that input from LilyPond fragments entered with the `\lilypond` command will be printed on a single line. But as such fragments are intended to contain short snippets anyway this shouldn't be an issue.

Partial printing

If the LilyPond input contains a comment with the character sequence `% begin verbatim` then everything up to and including this comment will *not* be printed verbatim (but still used for engraving the score). If after that `% end verbatim` is found then the remainder of the input will be skipped too, otherwise the code is printed to the end.

`addversion` *(false)*

If `addversion` is set the LilyPond version used to compile the current score is printed before the verbatim input code.

`intertext` *()*

If `intertext` is set to a string its value will be printed between the verbatim code and the score.

`\lyIntertext`

By default the intertext will be printed in its own paragraph, with a `\bigskip` glue space between it and the score. The appearance is controlled by the macro `\lyIntertext`, and by renewing this macro the appearance can be modified. The following redefinition removes any indent and prints the text blue:

```
\renewcommand{\lyIntertext}[1]{%
\noindent \textcolor{blue}{#1}\par\bigskip%
}
```

Syntax Highlighting

By default printed LilyPond code will be wrapped in a `verbatim` environment. It is possible to change the way how the code is wrapped through the command

`\lysetverbenv`

which works very much like `\newenvironment` and expects the code to be inserted before and after the LilyPond code as its two arguments. Typical use cases would be to enable some syntax highlighting, although it may also be of interest to wrap the `verbatim` environment into a `quote` environment.

So far no proper syntax highlighting for LilyPond is available in LaTeX (which is why it is not switched on by default), and the closest match today is to use the `TeX` highlighting of the `minted` package.

```
% In the document header:
\usepackage{minted}

% anywhere in the header or the body:
\lysetverbenv{\begin{minted}{TeX}}{\end{minted}}
```

Miscellaneous Options

Include Paths

When referencing external files with `\lilypondfile` *ly*LuaTeX understands absolute and relative paths. By default, relative paths are considered relative not to the current `.tex` document's directory, but to the *current working directory*, which is one reason why it's strongly recommended to launch LuaLaTeX from the document's directory. Additionally, *ly*LuaTeX will find any file that is visible to LaTeX itself, i.e. all files in the `texmf` tree. A special case are paths that start with a tilde (~). This tilde (which has to be input as `\string~` in LaTeX) will be expanded to the user's `HOME` directory, which should work equally in UNIX/Linux and Windows.

`includepaths` *(./)*

With the `includepaths` option a comma-separated list (enclosed in curly brackets) of search paths can be specified. These paths will be used by *ly*LuaTeX to locate external files, and relative paths are searched for in the following order:

- relative to the current `.tex` file's directory (i. e. the file from which the score is included)
- relative to each `includepath`, in the order of their definition in the list
- using LaTeX's search mechanism

Additionally the list of include paths is passed to LilyPond's include path, so they can be used for including files from within the LilyPond code. Paths starting with the tilde will implicitly be expanded to absolute paths in that process.

```
\setluaoption{ly}{includepaths}{\string~/lilypond-lib}

\lilypondfile[includepaths={\string~/lilypond-lib,/home/johndoe/project-lib}]
```

LilyPond Executable

By default *ly*LuaTeX will invoke LilyPond through the `lilypond` command, which will work in many situations for default installations. However, in order to accomodate specific installations (Windows?) or to use specific versions of LilyPond the command to be used can be specified with the

`program` *(lilypond)*

option. If given this must point to a valid LilyPond *executable* (and not, say, to the installation directory). If LilyPond can be started the version string will be printed to the console for every score, otherwise an error is raised, as is described in Handling LilyPond Failures.

`ly-version` *(2.18.2)* The LilyPond version to be written to the generated LilyPond code. This option is partially redundant with the `program` option but may serve as a guard against using outdated LilyPond versions. This can for example be relevant when sharing documents and `program` is set to its default `lilypond`, which may be something different on another computer.

Temporary Directory for Scores

*ly*LuaTeX uses a temporary directory to store LilyPond scores. For each score a unique name will be created using its *content* and the state of all options. LilyPond will only be invoked to compile a score when no corresponding file is present in the temporary directory, an approach that avoids unnecessary recompilation while ensuring that any updates to the content or the parameters of a score will trigger a new score.

`tmpdir` *(tmp-ly)*

The directory that is used for this purpose can be set with the `tmpdir` option. Its value is a relative path starting from the *current working directory*, i.e. the directory from which LuaLaTeX has been started, not necessarily that of the `.tex` document. Note that for several reasons it is strongly suggested to always compile documents from their own directory.

`cleantmp` *(false)*

While the caching mechanism is great for avoiding redundant LilyPond compilations it can quickly produce a significant number of unused score files since *any* change will cause a new set of image files to be generated. Therefore the `cleantmp` option can be used to trigger some garbage collection after the LaTeX document has been completed.

*ly*LuaTeX writes a `<documentname>.list` log file to the temporary directory, listing the hashed filenames of all scores produced in the document. If the score has been given a `label` (see Labels) or if it is generated from an external file this information is added to the list entry for use in any later inspection.

With the `cleantmp` option in place *ly*LuaTeX will remove *all* files that have not been generated from the current document. Note that this will also remove scores that may become useful again in the future if changes to the document will be reverted (for example if a document is created for different output formats). But of course these will simply be regenerated when necessary.

When the temporary directory is shared by several documents purging files might remove scores needed by *other* documents. Therefore *ly*LuaTeX will read *all* `<documentname>.list` files and only remove scores that are not referenced by *any* list file.

Writing headers to include file

`write-headers` *(false)*

When using `\lilypondfile` it is possible to write a copy of the LilyPond headers defining the layout and appearance of the score to an include file. When working on the score in an external editor this makes it possible to include this file to see the score in the layout it will have in the final LaTeX document. Using this option together with non-filebased scores makes no sense, therefore it is ignored while a warning is issued.

NOTE: Of course this will produce conflicts if a LilyPond file is used in multiple LaTeX documents.

If set to a *path* the LilyPond headers defining the layout and appearance of the score will be exported to a file `<path>/<input-file-basename>-lyluatex-headers.ily`. The target directory will be created if necessary.

If set to a *filename* (i.e. a path with a file extension) the headers will be written to this specific file. This is useful because in most cases the headers will be consistent throughout a LaTeX document, so it should be unncecessary to copy them for all input files. A typical use case might be to specify one header file as a package option while overriding the option for specific scores that require different headers (e.g. in combination with a different `staffsize`)

PDF optimization

optimize-pdf *(false)*

If set to true, each included pdf will be optimized by ghostscript before inclusion. It's set to false by default, because it's time consuming, and it loses information about the fonts.

Handling LilyPond Failures

Compiling a score with LilyPond can produce several types of problems which will be detected and handled (if possible) by *ly*LuaTEX. The most basic problem is when LilyPond can't be started at all. *ly*LuaTEX will correctly determine and report an error if LuaLATEX has been started without the --shell-escape option or if the program option doesn't point to a valid LilyPond executable. However, if the showfailed option is also set then only a *warning* is issued while instead of a score an information box is created in the document, informing about the problem.

Two other situations that are correctly recognized are when LilyPond *reports* a compilation failure but still produces a (potentially useful) score, and when LilyPond actually fails to engrave a score. How this is handled is controlled by the debug and showfailed options.

debug *(false)*

If LilyPond reports an error and debug is set to true then *ly*LuaTEX will save both the generated LilyPond code and the complete log output to a .ly and a .log file in the temporary directory. The file names are printed to the console for easy reference. Otherwise only a general warning will be issued. This will happen regardless of whether a score file is produced or not. In addition *ly*LuaTEX will usually delete intermediate files that are not useful for later compilations but keep them all when debug is active.

showfailed *(false)*

If LilyPond failed to produce a score and showfailed is set to false then the LATEX compilation will stop with an error. This error can be skipped, but nothing will be included in the document. If on the other hand showfailed is set to true only a warning is issued and a box with an informative text is typeset into the resulting document.

Forcing (Re-)Compilation

force-compilation *(false)*

In some cases *ly*LuaTEX's heuristics to determine the need for recompilation may fail, especially when not all relevant code is included through LilyPond's \include command, in which cases *ly*LuaTEX may consider the content unchanged. In such cases the force-compilation option skips the checks and unconditionally recompiles the score, which may be a better solution than to (selectively) delete the scores from the tmpdir directory.

Bug workaround

`fix_badly_cropped_staffgroup_brackets` *(false)*

This option is a dirty workaround for a known bug of LilyPond. It's disabled by default; should you enable it globally, you may cancel it locally with `nofix_badly_cropped_staffgroup_brackets`.

MusicXML options

`xml2ly` *(musicxml2ly)*

This option does the same for `\musicxmlfile` as `program` for `\lilypondfile`.

`language` *()*

`absolute, lxml, verbose` *(false)*

`no-articulation-directions, no-beaming, no-page-layout, no-rest-positions` *(true)*

All those options control the corresponding `musicxml2ly` switches; please refer to `musicxml2ly` documentation for more information.

Using *ly*LuaTEX in Classes or Style Files

Wrapping *ly*LuaTEX commands

`\lilypond` and `lilypond` are aliases for a command and an environment that *ly*LuaTEX defines internally, respectively `\lily` and `ly`.

`\lily` can be wrapped within another command in an usual way; but `ly` is quite a special environments, which makes it a bit unusual to wrap. You'll find more about this point in Wrapping Commands

Providing Raw filenames

*ly*LuaTEX's default mode of operations is to directly insert scores into the document. For this the generated PDF files of the scores are transparently wrapped in `\includegraphics` or `\includepdf` commands and given appropriate layout.

`raw-pdf` *(false)*

However, for more control over the placement and handling of the scores, especially for package developers, `raw-pdf` provides the option to make available the raw file name(s) to be processed and wrapped at will. When `raw-pdf` is set *ly*LuaTEX will implicitly and temporarily define a command

\lyscore

taking one mandatory argument, which may be empty. In this case \lyscore{} expands to the filename of the first system of the score while \lyscore{N} will return the filename of the N-th system. The special keywords \lyscore{nsystems} and \lyscore{hoffset} return the number of systems in the score and <hoffset>pt as a distance to be applied to handle protrusion.

Additionally any *ly*LuaTEX option can be used to retrieve the corresponding given or calculated value. For example \lyscore{valign} will return top, center, or bottom. By accessing these options it is possible to make use of information that is not part of the actual generated score but that would otherwise be used by *ly*LuaTEX's LATEX wrapping.

Examples:

Examples on how raw filenames can be wrapped in secondary commands can be found in Wrapping Raw PDF Filenames.

Index

\betweenLilyPondSystem, 10
\lilypond, 7
\lilypondfile, 7
\lyFilename, 22
\lyIntertext, 23
\lyscore, 28
\lysetverbenv, 23
\musicxmlfile, 7
\preLilyPondExample, \postLilyPondExample, 10
\setluaoptionly, 9

no-articulation-directions, no-beaming, no-page-layout, no-rest-positions , 27

absolute, lxml, verbose, 27
addversion, 22
autoindent, 17

cleantmp, 25
clip-regions, 12
current-font-as-main, 19

debug, 26

extra-bottom-margin, 18
extra-top-margin, 18

first-page-number, 10
fix_badly_cropped_staffgroup_brackets, 27
force-compilation, 26
fragment, 18
fullpagealign, 18
fullpagestyle, 10

gutter, leftgutter, rightgutter, exampleindent, 13

hpadding, 11

includepaths, 23
indent, 13, 16
inline-staffsize, 11
insert, 9
intertext, 22

label, 21
labelprefix, 21
language, 21, 27
lilypond, 6
line-width, 13
Local Options, 8
ly-version, 24

max-left-protrusion, 15
max-protrusion, 15
max-right-protrusion, 15

noclef, 20
nostaff, 20
nostaffsymbol, 20
notime, 20
notimesig, 20
notiming, 20

optimize-pdf, 26

Package Options, 8
paperheight, 14
papersize, 14
paperwidth, 14
pass-fonts, 19
print-only, 11
print-page-number, 10
printfilename, 22
program, 24

quote, 13

ragged-right, 13
raw-pdf, 27
relative, 21
rmfamily, 20

sffamily, 20
showfailed, 26
staffsize, 13
system-count, 13

tmpdir, 24

ttfamily, 20
twoside, 14

valign, 11
verbatim, 22
voffset, 11

write-headers, 25

xml2ly, 27

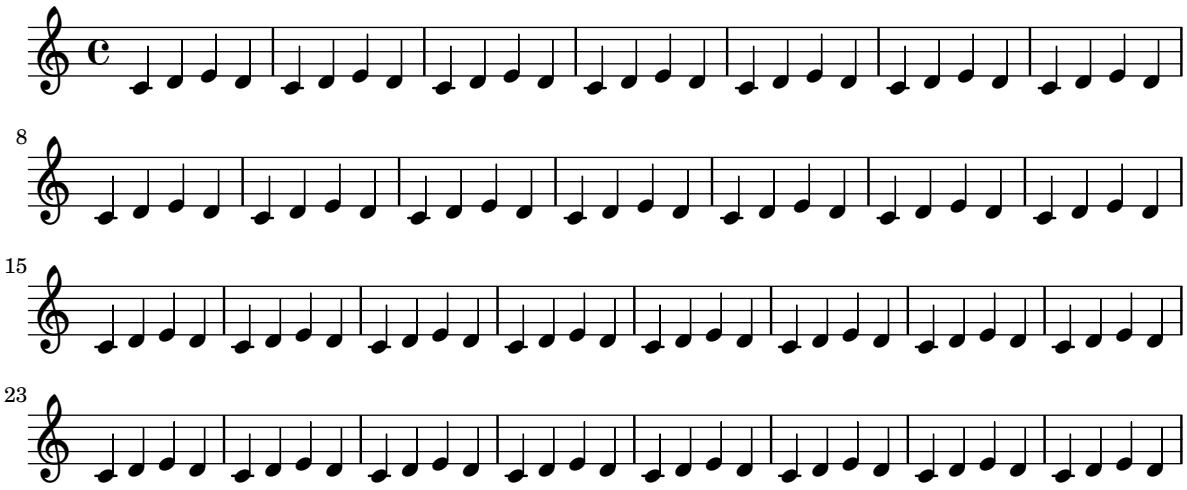
Examples

Those examples and others may be found in the package repository.

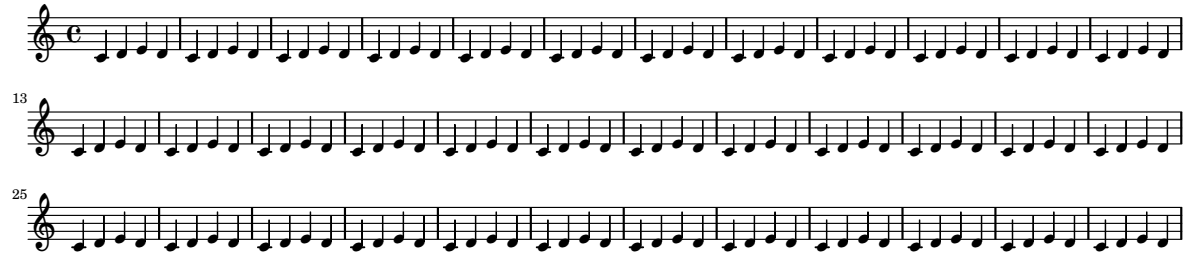
Insert System-by-System

By default scores defined by the `lilypond` environment or the `\lilypondfile` command are inserted as a sequence of systems.

*ly*LuaTEX determines the vertical space between the systems as a flexible length calculated from the *staff size* of the score (as opposed to from the font size) to produce an regular-looking vertical spacing:



The following score has a significantly smaller staff size, and consequently the inter-system space is reduced:



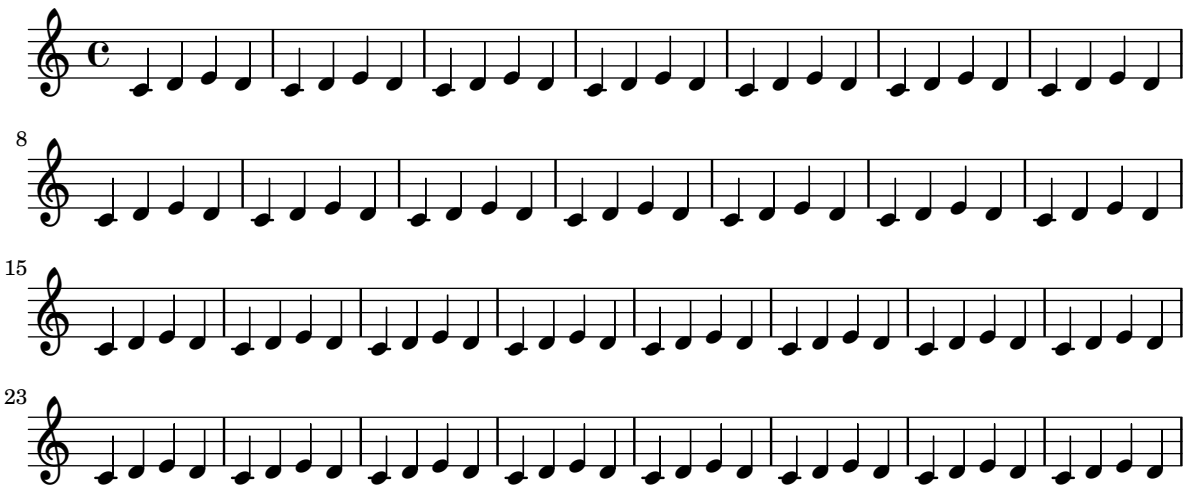
Before and After the Score

\preLilyPondExample and \postLilyPondExample allow some code to be printed before and after the score. This may for example be used to wrap the resulting score in an environment. In the following example rules are printed:

```
\newcommand{\preLilyPondExample}{%
\par\bigskip
\noindent Before the score:
\par\medskip\hrule\par\medskip}

\newcommand{\postLilyPondExample}{%
\par\bigskip
\hrule\par\medskip\noindent After the score
\par\bigskip}
```

Before the score:

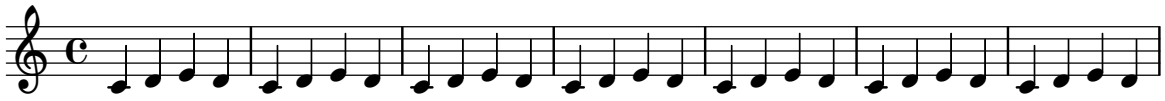


After the score

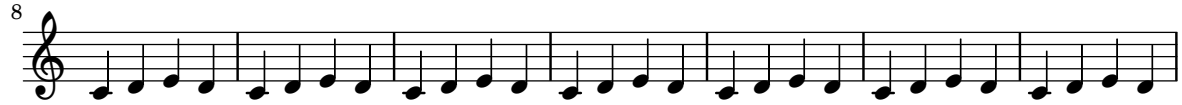
Configuring the Inter-System Content

Using `\betweenLilyPondSystem` it is possible to define a macro that is expanded between each system pair. It is given the index of the previous system as an argument to work with. The following example simply prints that index between the systems, but with some programming more complicated and useful things could be done, for example printing a rule after every third system or conditionally insert a page break.

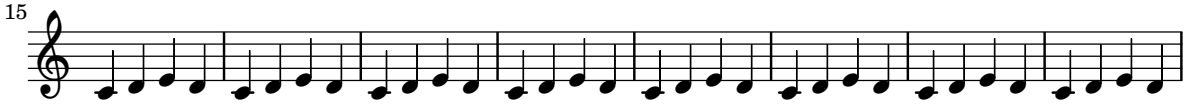
```
\newcommand{\betweenLilyPondSystem}[1]{%
\begin{center}
System #1
\end{center}
}
```



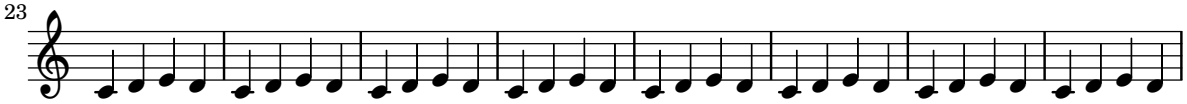
System 1



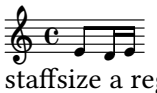
System 2

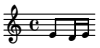


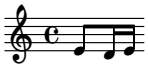
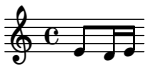
System 3

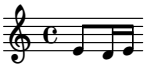


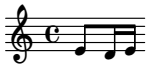
Insert Scores Inline

With the `insert=inline` option it is simple to insert arbitrary notational fragments in the  continuous text of a document. By default the staffsize is scaled to be 2/3 of the staffsize a regular score would have at this point. This means if the `staffsize` option is modified globally or locally then the staffsize of the inline score is changed too.

In order to make the size of inline scores independent from the regular staffsize the option `inline-staffsize` can be used the same way as `staffsize`.  has the inline staffsize manually set to 8.

Alignment and padding By default inline scores are vertically centered to a line 1/2em above the text's baseline.  but the score can also be aligned  to the top or the baseline of the text.

Unfortunately this can only consider the borders of the *image* and not those of the *score* or the staff lines. To alleviate this situation a specific vertical offset can be given with `voffset=-3pt` (or any other TEX lengths). This offset is calculated after the alignment.  is inserted with `valign=bottom,voffset=-4pt`.

Horizontally inline scores are padded by `hpadding=0.75ex` – except if they happen to appear at the beginning or end of a line, as can be seen in the last score in the previous paragraph.  Increasing the `hpadding` will ensure more space around the score.

Bare Inline scores `insert=bare-inline` will remove all the staff elements (staff symbol, time signature, clef) by implicitly applying `nostaff`, which is most useful for including notational symbols like characters in the paragraph.  This actually works like the `lilyglyphs` package[8] but with the possibility of inserting arbitrary LilyPond material without having to prepare precompiled PDF images.

Print only Selected Systems or Pages

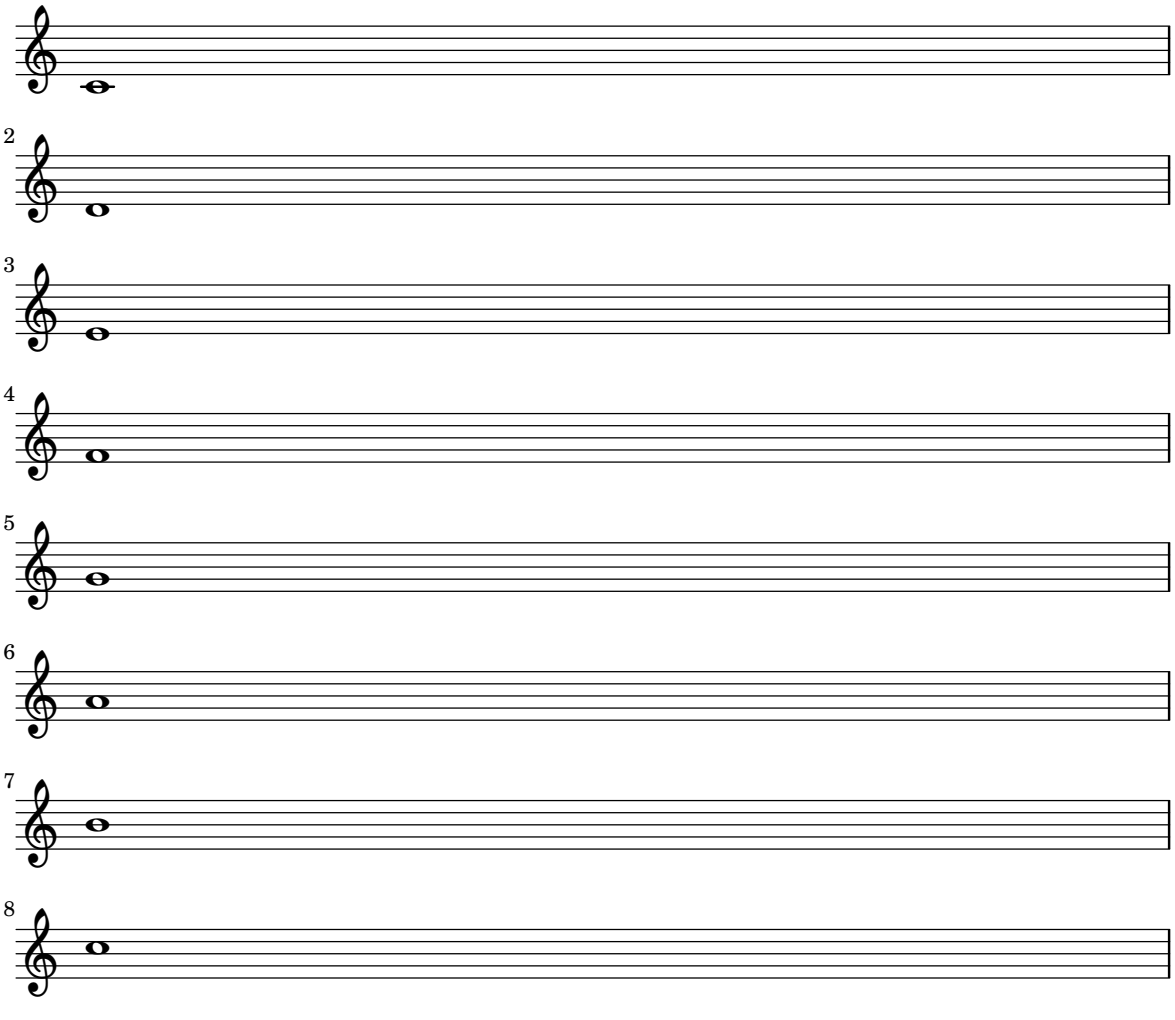
The `print-only` and `do-not-print` options allow to limit the printed systems or pages from a score. A typical use case is to print a score interspersed with comments. The advantage of this approach is that the score is compiled only once while the individual systems are simply reused by LATEX.

Throughout this document we'll demonstrate the different options to select systems from the following score:

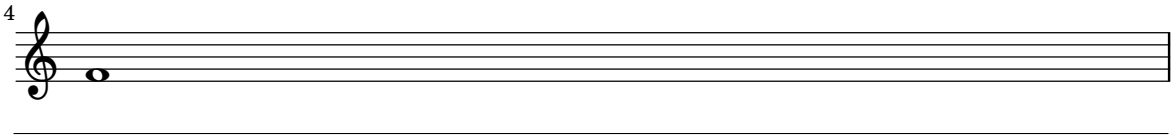
[8] https://github.com/uliska/lilyglyphs

```
\version "2.18"

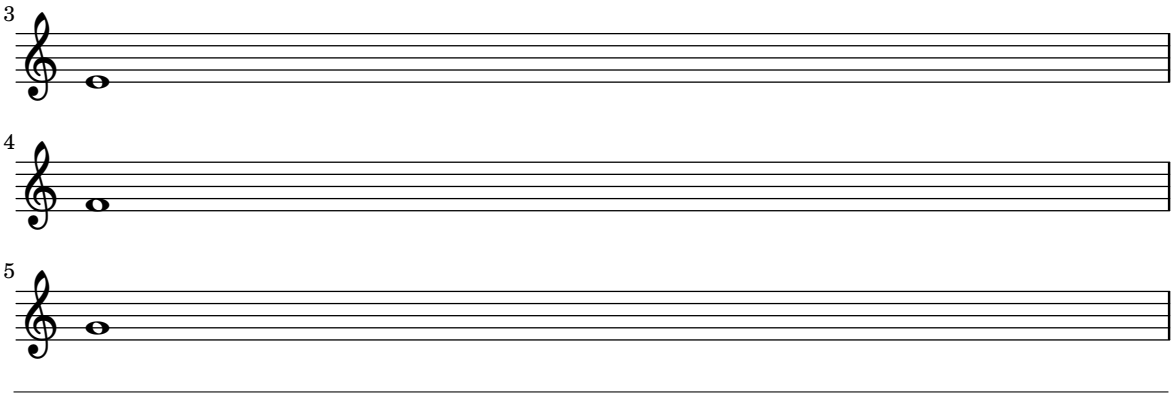
\relative c' {
  \omit Staff.TimeSignature
  c1 \break
  d1 \break
  e1 \break
  f1 \break
  g1 \break
  a1 \break
  b1 \break
  c1 \break
}
```

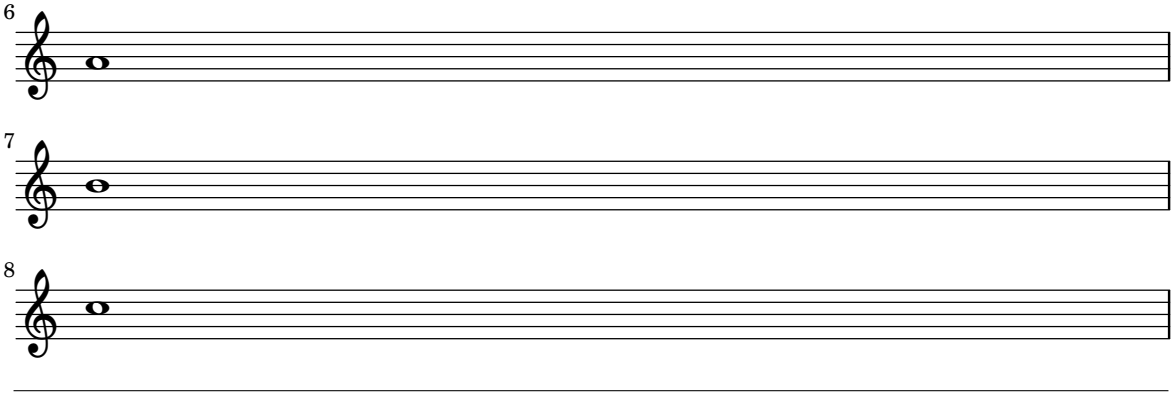


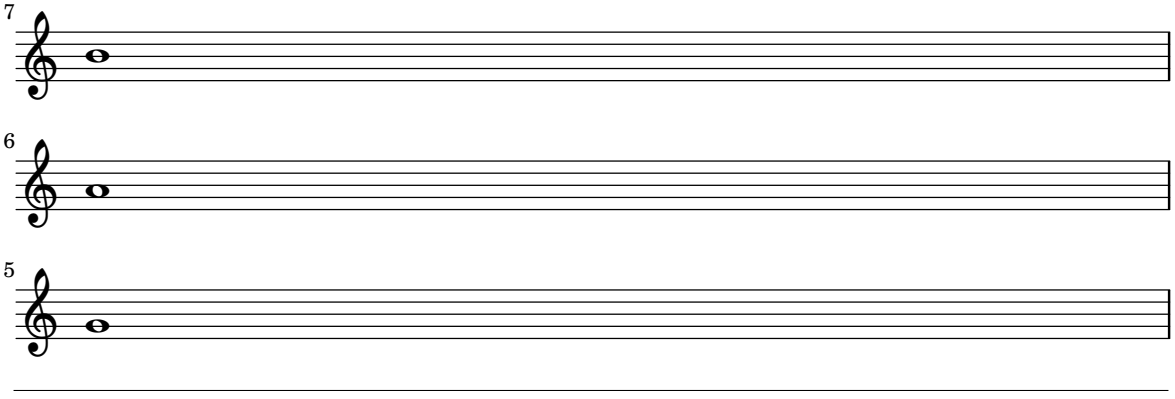
The simplest selection is a single system: `print-only=4`



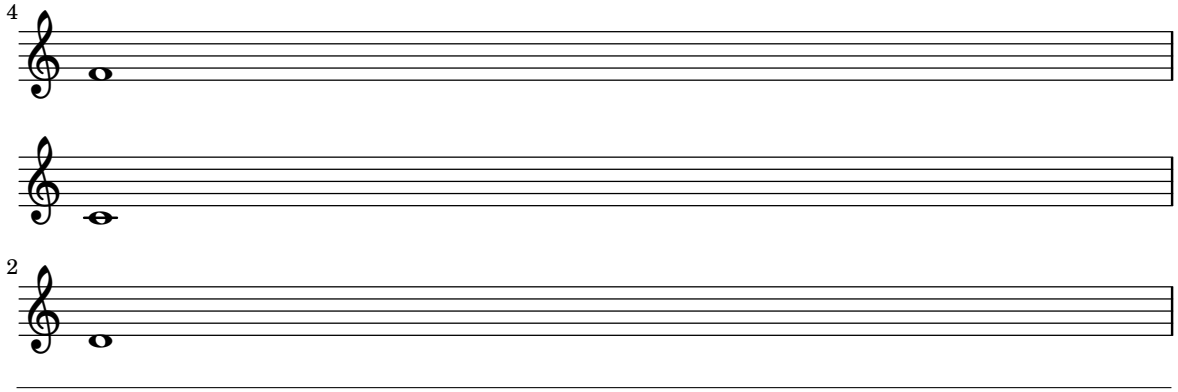
Ranges are also possible: `print-only=3-5`, with the special form of `print-only=6-` which prints from the given system throughout the end of the score. Negative ranges can be given with `print-only=7-5`







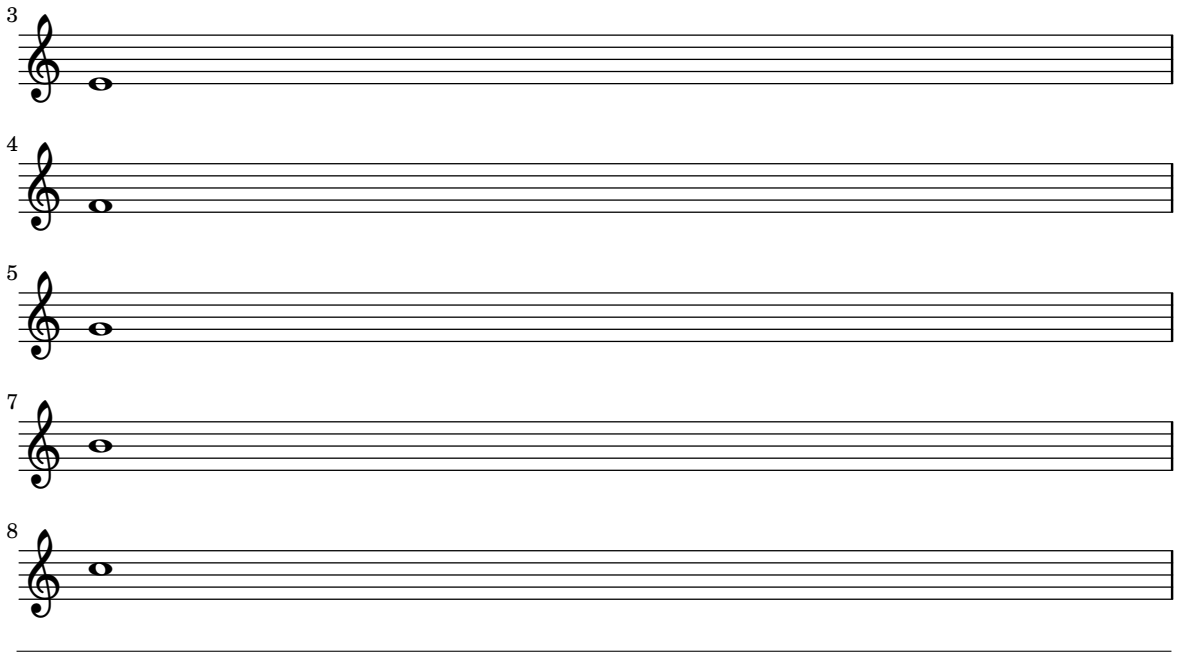
With a comma-separated list an arbitrary sequence of systems can be specified. The list has to be enclosed in curly brackets: `print-only=4,1,2`



Each element of the list can include any of the forms described above:
`print-only=3,5-7,4,7-`



`do-not-print` does the opposite: it prevents the list of systems from being printed. It might be used alone, or in combination with `print-only`:
`print-only=3-,do-not-print=6`



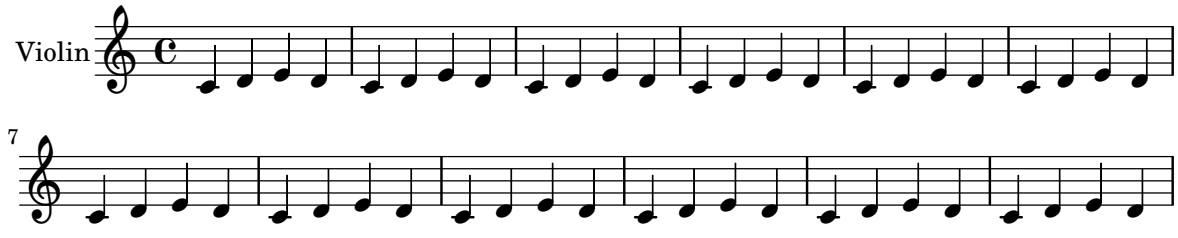
The functionality is identical with fullpage scores where the selection applies to *pages* instead. This can for example be used when the "score" file contains a number of individual pieces (e.g. songs for a song book), and individual selections are to be printed.

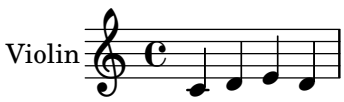
Systems have some specific behaviour with regard to *indent*, but this is demonstrated in its own file `dynamic-indent.tex`.

Dynamic Indent

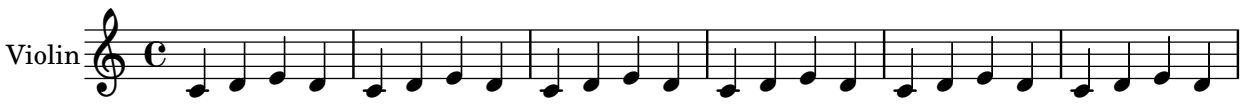
This document demonstrates the use of `indent` and `autoindent`, partially in combination with `print-only`.

`indent=1cm` indents the first line, but if the resulting score contains only one system this indent is suppressed (issuing a warning on the console):

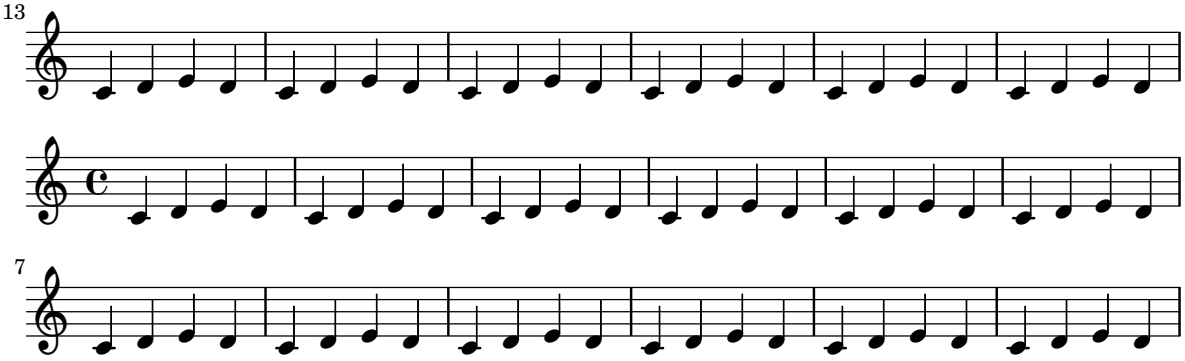




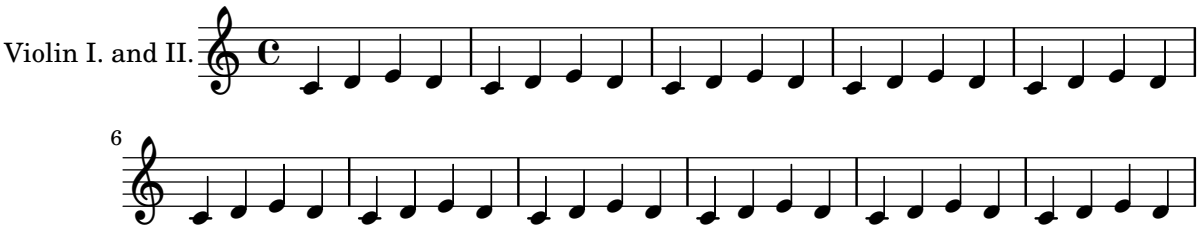
If the output of a score which contains more than one system is limited to the first system using `print-only=1` then the indent is removed but the score is recompiled to ensure a full-length system. The following score shows the two-system score from above (with `indent=1cm`), limited to its first system:



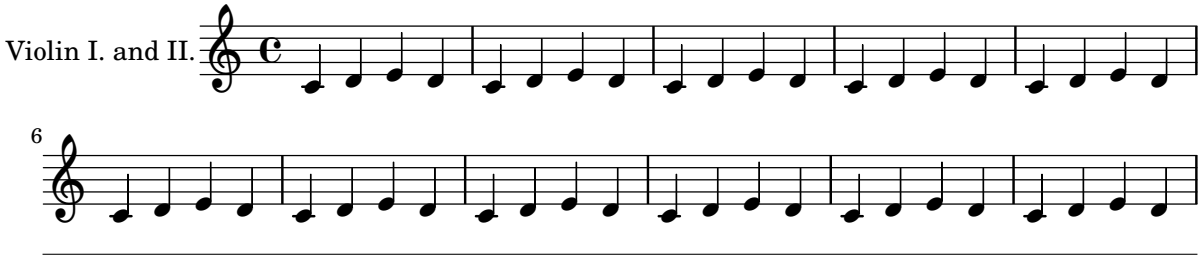
Note that this behaviour also applies when `print-only` causes the first system to be printed at another position, e.g. with `print-only=3,1,2`. In this case the indent of the first system is suppressed in order to avoid a "hole". Of course this is a corner case, but might be useful when a score consists of separate entities (examples, exercises) per system.



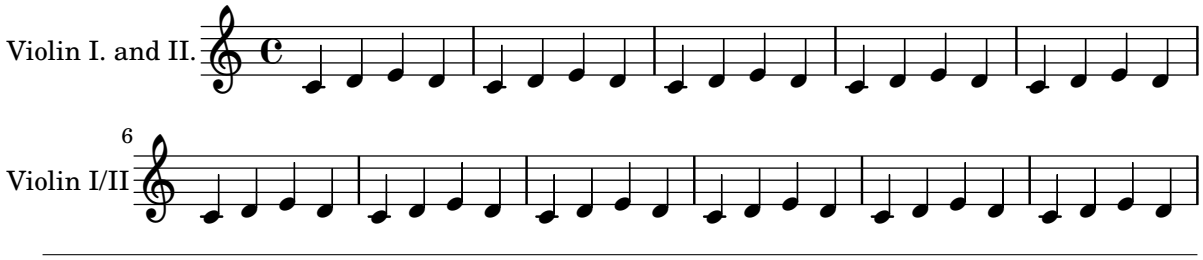
If a protrusion limit has been set with `max-protrusion=0.5cm` and the score exceeds that limit in spite of `indent=1cm` then the whole score will appropriately be narrowed:



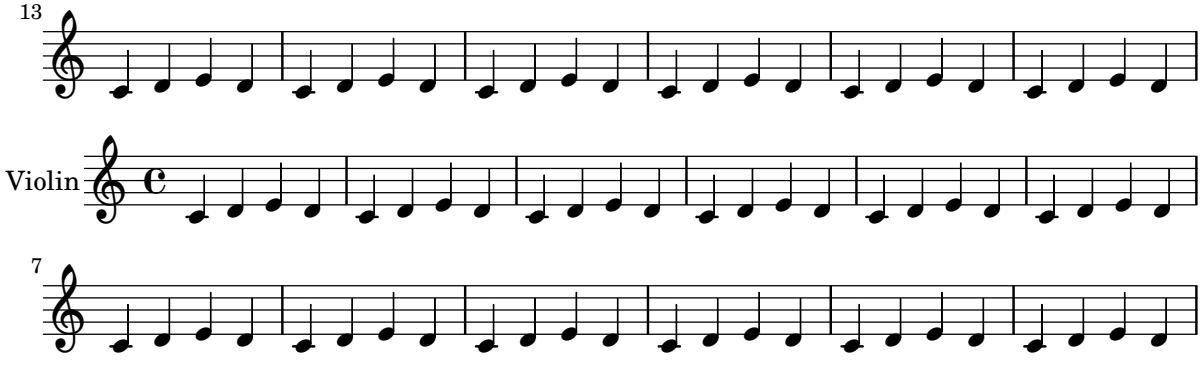
This doesn't really look good because the indentation of the second system wouldn't have been necessary since only the first system exceeds the protrusion limit. The solution to this situation is the option `autoindent` which handles the indentation *automatically* and set the indent to a value that will make the *first* system fit into the protrusion limit and leave the remaining systems unchanged:



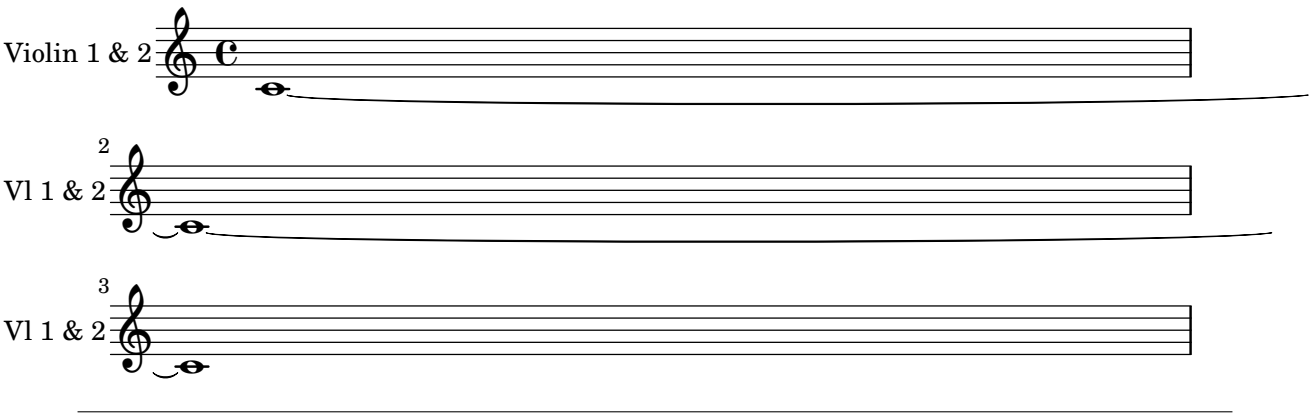
However, if the protrusion limit is not only exceeded by the *first* system (which should be the typical case due to the instrument name) `lyluatex` will deal with the situation by narrowing the *whole* score by the appropriate amount and adjusting the indent of the first system so all systems will just fit into the protrusion limit:



There is one special case to be mentioned. As described above the indent is deactivated if the first system of a score is printed at a later position. However, if this score will exceed the left protrusion limit `autoindent` will be automatically activated to avoid having the *whole* score narrowed:



Right protrusion The dynamic handling of (automatic) indent also works correctly when there is protrusion handling to the right. The following score has the ties manually shaped to exceed the staff symbol by 10, and 7 staff spaces, and `max-protrusion=1cm` .



Performance considerations The handling of indent suppression may require up to four compilations of the score, but these are handled automatically, and the resulting intermediate stages of the score are cached just like the scores actually used in the document.

The `autoindent` option is active by default but will be deactivated if `indent` is set explicitly. It has to be noted that this option will add more LilyPond compilations and therefore compilation time. But it will only apply and be executed if the score exceeds the protrusion limit, so it can only occur in circumstances where multiple LilyPond runs are expected anyway.

Font Handling

To demonstrate the font handling features of *ly*LuaTEX we will repeatedly include the following score from an external file. It includes roman (lyrics, instrument name), sans (rehearsal mark), and mono (tempo) text, first using LilyPond's built-in default fonts.

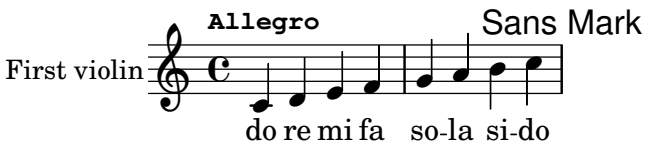
```
\version "2.18"

mel = {
  \set Staff.instrumentName = "First violin"
  \tempo \markup \typewriter Allegro
  c' d' e' f'
  g' a' b' c''
```

```
  \mark \markup \sans "Sans Mark"
}

lyr = \lyricmode {
  do re mi fa so -- la si -- do
}

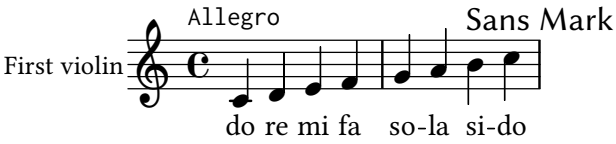
\score {
  <<
  \new Staff \new Voice = "mel" \mel
  \new Lyrics \lyricsto "mel" \lyr
  >>
}
```



The current document uses `fontspec` to set roman font to *Linux Libertine O*, sans font to *Linux Biolinum O*, and mono font to *Inconsolata*. So if you compile this document yourself and don't have these fonts installed you will receive unexpected results.

Passing Document Fonts to Score

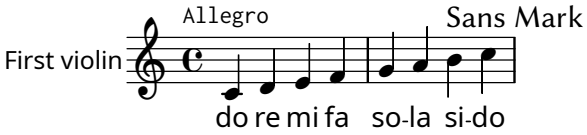
With `pass-fonts` the currently active font families for roman, sans, and mono fonts are passed to LilyPond in order to achieve the most coherent appearance between text and music.



Note that LilyPond loads fonts differently than LaTeX and can only make use of fonts installed as system fonts, fonts that are only installed through a LaTeX distribution are not accessible to it. That means that if the document fonts are not installed system-wide (e. g. the default fonts) LilyPond will use rather ugly fallback fonts. This can't be demonstrated here but the section about explicitly setting font families will include an example.

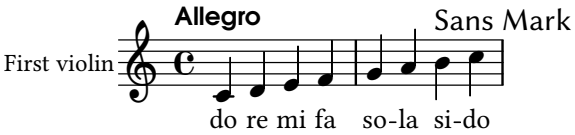
The inherent problem of fallback fonts, especially with LaTeX's default settings, is the reason `pass-fonts` is inactive by default. But the general recommendation is to set `pass-fonts` as package option if the text document uses fonts that are available to LilyPond.

`current-font-as-main` will use the font that is *currently* used for typesetting as LilyPond's main (roman) font. This can make sure that the score's main font (and roman is usually the font used most in scores) matches the surrounding text. Note that this might produce surprising behaviour if it is not clear that the current font has changed, and it will effectively suppress the original roman font from the score if the current font is one of the two others. Additionally this *may* introduce an inconsistency not between the score and the surrounding text but between different scores in a document. For all these reasons the option is by default set to `false`.

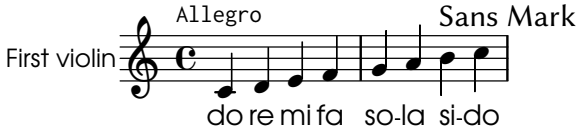


Setting Score Fonts Explicitly

With `rmfamily`, `sffamily`, and `ttfamily` specific families can be set to arbitrary fonts, independently from the text document. For the following score `ttfamily={TeXGyre Adventor}` is used.[9] Note that this implicitly sets `pass-fonts=true`, and *Linux Libertine O* and *Linux Biolinum O* are used from the text document.

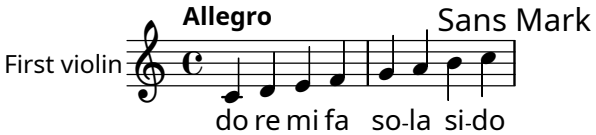


NOTE: when `rmfamily` is set explicitly `current-font-as-main` is forced to `false` to ensure that the roman font is actually used. The next score sets `rmfamily={TeXGyre Adventor}` and `current-font-as-main`, and despite the current font still being `\sffamily` *Adventor* is used as the score's main font:



LilyPond's Font Fallback

If unavailable fonts are set in a LilyPond document they will *silently* be replaced with fallback fonts that tend to cause ugly results. This will be shown by setting `rmfamily=FantasyFontOne`, `sffamily=FantasyFontTwo`, and `tfamily=FantasyFontThree`:



[9] Note that this font (which is included in TeXLive) has to be installed if you want to successfully compile this document.

This can happen in several contexts: apart from compiling the document on a different computer where the used fonts are missing it is most likely to occur with the pass-fonts option, when the text document uses internal LaTeX fonts. Note in particular that this may happen implicitly when only one family is specified explicitly with an option and the other families are passed from the text document.

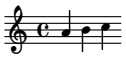
Wrapping Commands

Command within commands

\lily can be wrapped within another command as usual:

```
\newcommand\mylily[2][1]{\lily[inline-staffsize=10, #1]{#2}}

This is \mylily[voffset=10pt]{a' b' c''} an example.
```

This is  an example.

Environment within environments

It isn't possible to wrap ly environment within a command.

It's possible to wrap ly within and environment, but there are several drawbacks[10].

To avoid those drawbacks, *ly*LuaTeX defines a special command, \lynewenvironment, that behaves as you'd expect from \newenvironment.

```
\lynewenvironment{myly}{%
  This is \emph{my} lilypond environment.
  \begin{ly}%
}{%
  \end{ly}
}

\begin{myly}
```

[10] Those drawbacks are:

- this custom environment cannot have optional parameters. To be more precise, if it has only optional parameters, it will be necessary to add [] after \begin{MY_ENV} if no parameter is specified ; so they're not optional any more...
- to call ly, you'll have to:
  - either write \begin{ly}[] \end{ly} (which works with \begin{lilypond}[] \end{lilypond} too) ;
  - or use the TeX primitives \ly \endly (not only for ly, but also for other environments).

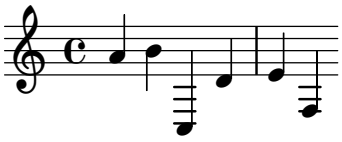
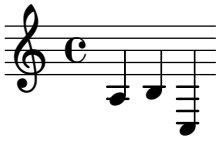


Figure 1: This is a caption

```
  a b c
\end{myly}
```

This is *my* lilypond environment.



```
\lynewenvironment{lyfigure}[2][]{%
\edef\mycaption{#2}
\begin{figure}
\begin{center}
  \begin{lilypond}[#1]%
}{%
  \end{lilypond}
  \caption{\mycaption}
\end{center}
\end{figure}
}

\begin{lyfigure}{This is a caption}
a' b' c
d' e' f
\end{lyfigure}


\lynewenvironment{lyotherfigure}[1][]{%
\edef\option{#1}
\figure
\center
  \ly
}{%
  \endly%
  \def\empty{}\ifx\option\empty\else\caption{\option}\fi
```

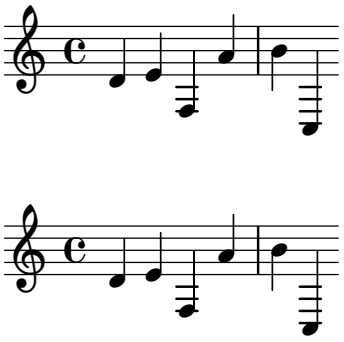


Figure 2: This time with a caption

```
\endcenter
\endfigure
}

\begin{lyotherfigure}
d' e' f
a' b' c
\end{lyotherfigure}


\begin{lyotherfigure}[This time with a caption]
d' e' f
a' b' c
\end{lyotherfigure}
```

Important note: `\lynewenvironment` is intended to insert LaTeX code before and after the scores; due to the special behavior of `ly` environment, it isn't possible to insert *LilyPond* code that way. So this won't work:

```
\lynewenvironment{myly}{%
  \begin{ly}
    a b c
}{%
  \end{ly}
}
```

To do such a thing, *ly*LuaTeX defines a command and four options:

- `\lysavefrag` lets one save a LilyPond fragment to be re-used afterward;
- `include_header`, `include_footer`, `include_before_body` and `include_after_body` options let one insert such fragments at designed places within inserted score.

So this works:

```
\begin{lysavefrag}{head}
a b c
\end{lysavefrag}

\begin{lysavefrag}{foot}
g a' b
\end{lysavefrag}

\begin{lysavefrag}{mymark}
\mark \default
\end{lysavefrag}

\begin{lysavefrag}{mymark}
\mark \default
\end{lysavefrag}

begin{ly}[
  include_before_body={head,mymark,head},
  include_after_body=foot,
]
d e f
\end{ly}
```

It's also possible to use `\lynewenvironment` to wrap such a command:

```
\begin{lysavefrag}{head}
a b c
\end{lysavefrag}

\begin{lysavefrag}{foot}
g a' b
\end{lysavefrag}

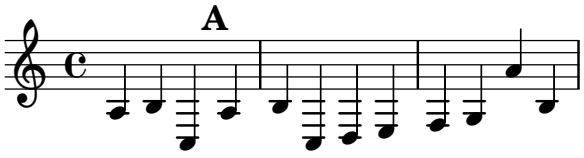
\begin{lysavefrag}{mymark}
\mark \default
\end{lysavefrag}

\lynewenvironment{yourly}[1][]{%
  {\centering test \par}
  \begin{ly}[
    include_before_body={head,mymark,head},
    include_after_body=foot,
  ]
```

```
}{
  \end{ly}
}

\begin{yourly}
d e f
\end{yourly}
```

test



Wrapping Raw PDF Filenames

With the raw-pdf option it is possible to create wrapping commands that circumvent *ly*LuaTEX's layout considerations by working with the raw PDF filename of the generated score. This is especially useful for developing packages or personal class and style files. For this scores generated with raw-pdf define a command \lyscore that can be used in the wrapping commands or environments.

All examples in this document could also be realized using "default" *ly*LuaTEX without raw-pdf, but they are intended to show how this low-level access can be used to retrieve the information from the generated score in order to build custom versions of commands that don't have to adhere to *ly*LuaTEX's pre-built strategies of including the score in the document

The easiest way to use a "raw" score is to simply access \lyscore in a command and pass it to an \includegraphics macro:

```
\newcommand\lilyinline[2][]{%
\lily[raw-pdf,%
insert=bare-inline,%
inline-staffsize=8,%
hpadding=0.25ex,#1]{
\omit Stem
#2}%
\includegraphics{\lyscore{}}%
}
```

This basically is a way to provide pre-configured commands. In this case it is used to pre-configure an inline type, entered as `\lilyinline{ c'8 d' c' d'}`.

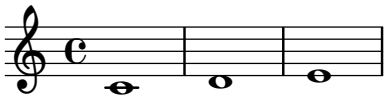


Figure 3: tmp-ly/147b904dccad7e63a757ff9de07ba982-1.pdf

`\lyscore` takes one mandatory argument which can be empty – as in the example above –, receive a number, one of the keywords `nsystems` and `hoffset`, or any of the score's options. If passed a number it will return the filename of the N-th system. With `nsystems` the number of systems in the generated score will be returned, while `hoffset` generates the code that shifts the score to the left to accommodate protrusion.

The following example takes an optional argument with options that are passed to *ly*LuaTEX, and one mandatory argument which expects the system to be used. It prints the given system centered in a figure and uses the file name as the caption and makes use of the score's `label`. Figure 3 shows the centering of a short fragment, figure 4 the selection of the fifth system from a larger score.

```
\newenvironment{centeredlilypondsystem}[2][]{%
\def\usesystem{#2}
\begin{figure}
\begin{center}
  \begin{lilypond}[raw-pdf,#1]%
}{%
  \end{lilypond}
  \includegraphics{\lyscore{\usesystem}}
  \caption{\lyscore{\usesystem}.pdf}
  \label{\lyscore{label}}
\end{center}
\end{figure}
}

\begin{centeredlilypondsystem}[label=centered]{1}
c'1 d' e'
\end{centeredlilypondsystem}

\begin{centeredlilypondsystem}[label=fifth]{5}
\repeat unfold 8 { c'1 \break }
\end{centeredlilypondsystem}
```

Finally there's an example showing how to iterate over the systems of a score using `\foreach` from the `pgffor` package. It iterates over all the systems in the given score, prints them using the protrusion adjustment seen before, and if the system is the third it prints this information, otherwise just a line break:

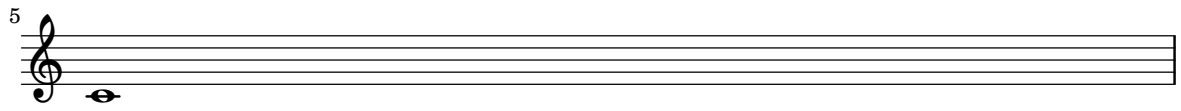
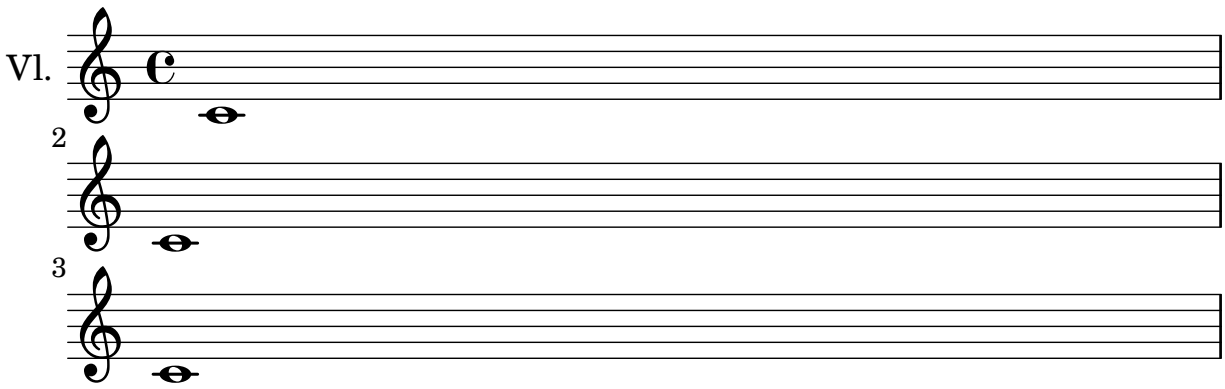


Figure 4: tmp-ly/fc76b417546d92d6ff5e7e00a7f332e4-5.pdf

```
\newcommand\myforlily[2][]{%
\lily[insert=systems,raw-pdf,#1]{#2}%
\foreach \n in {1,...,\lyscore{nsystems}}%
    {\noindent\hspace*{\lyscore{hoffset}}\includegraphics{\lyscore{\n}}%
    \ifthenelse{\equal{\n}{3}}{\par Third system\par}{\\}
    }%
}

\myforlily[staffsize=24]{
\set Staff.instrumentName = "Vl. "
\repeat unfold 4 { c'1 \break } }
```



Third system

