LilyPond

The music typesetter

Changes

The LilyPond development team

This document lists changes and new features in LilyPond version 2.24.3 since 2.22.

For more information about how this manual fits with the other documentation, or to read this manual in other formats, see Section “Manuals” in *General Information*.

If you are missing any manuals, the complete documentation can be found at `https://lilypond.org/`.

For LilyPond version 2.24.3

Note: LilyPond releases can contain syntax changes, which may require modifications in your existing files written for older versions so that they work in the new version. To upgrade files, it is **strongly recommended** to use the `convert-ly` tool distributed with LilyPond, which is described in Section "Updating files with convert-ly" in *Application Usage*. `convert-ly` can perform almost all syntax updates automatically. Frescobaldi users can run `convert-ly` directly from Frescobaldi using "Tools > Update with convert-ly...". Other editing environments with LilyPond support may provide a way to run `convert-ly` graphically.

Major changes in LilyPond

- LilyPond now requires Guile 2.2. Even if you are not writing Scheme code yourself, you may be using libraries that contain a non-trivial amount of customizations. If they do not work with LilyPond version 2.24.3, please report this to the library's developers. If you *are* a library developer, see [Notes on Guile 2.2], page 21, below.
- The infrastructure for creating the official binaries has been completely rewritten, incident with the switch to Guile 2.2. As of this release, we provide 64-bit binaries for macOS and Windows. Also, all packages are made available as simple archives that can be extracted to any "installation" location. To uninstall, simply delete that directory. We also discontinued the limited editor that was installed on macOS and Windows, LilyPad, and recommend switching to an external solution instead, such as the popular editor Frescobaldi (`https://frescobaldi.org`), or one of the others listed in Section "Easier editing" in *General Information*. For more information, please refer to the detailed instructions in Section "Installing" in *Learning Manual*.

Notes for source compilation and packagers

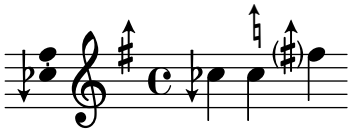
This section is aimed at enthusiasts compiling LilyPond from source and packagers preparing LilyPond for distribution. If you are not part of either group, you can skip over this section.

- As mentioned above, LilyPond now requires Guile 2.2. If needed for distribution reasons, it can also be compiled with Guile 3.0 by passing `GUILE_FLAVOR=guile-3.0` to the `configure` script. However, this is not yet recommended nor officially supported.
- The Scheme code evaluator in Guile 2.2 is slower than in Guile 1.*x*. To offset most of the performance penalty, we recommend compiling the `.scm` files into bytecode by first running `make bytecode` during compilation and then `make install-bytecode` in addition to `make install`.
- Starting with this stable release, LilyPond's build system does not install text fonts anymore. Please provide them as separate packages while paying attention to the fonts' license and notice files.

New for musical notation

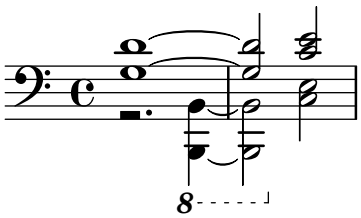
Pitches improvements

- Support for alternate accidentals was improved. Through the `alterationGlyphs` property of staff-like contexts, accidental glyphs may be set for all grobs at once (refer to Section "Alternate accidental glyphs" in *Notation Reference*).



- Ottava brackets may apply to a single voice instead of the entire staff. This used to require workarounds.

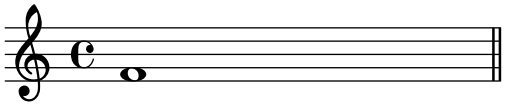
```
\layout {
  \context {
    \Staff
    \remove Ottava_spanner_engraver
  }
  \context {
    \Voice
    \consists Ottava_spanner_engraver
  }
}
```

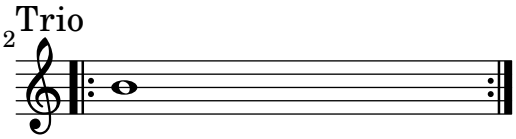


Rhythm improvements

- The new `\section` command inserts a double bar line that interacts gracefully with repeat bar lines. A passage can be named with the new `\sectionLabel` command.

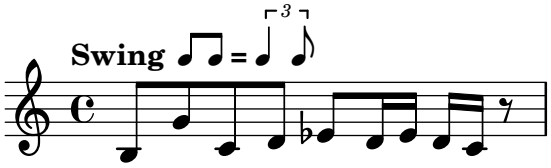
```
\fixed c' {
  f1
  \break
  \section
  \sectionLabel "Trio"
  \repeat volta 2 {
    b1
  }
}
```





- \numericTimeSignature and \defaultTimeSignature now act on all staves at once (more precisely, on all staves in the same Timing context), thus matching the behavior of \time.
- The \rhythm markup command has been added. It is a simple way to enter rhythms mixed with text, such as in "swing" indications.

```
\relative {
  \tempo \markup {
    Swing
    \hspace #0.4
    \rhythm { 8[ 8] } = \rhythm { \tuplet 3/2 { 4 8 } }
  }
  b8 g' c, d ees d16 ees d c r8
}
```



- The \enablePolymeter command is now provided as an input shorthand for moving engravers as is necessary to allow different time signatures in parallel. The code:

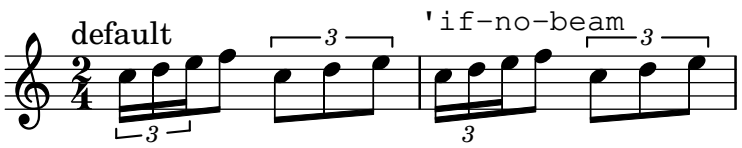
```
\layout {
  \context {
    \Score
    \remove Timing_translator
    \remove Default_bar_line_engraver
  }
  \context {
    \Staff
    \consists Timing_translator
    \consists Default_bar_line_engraver
  }
}
```

  can thus be shortened as:

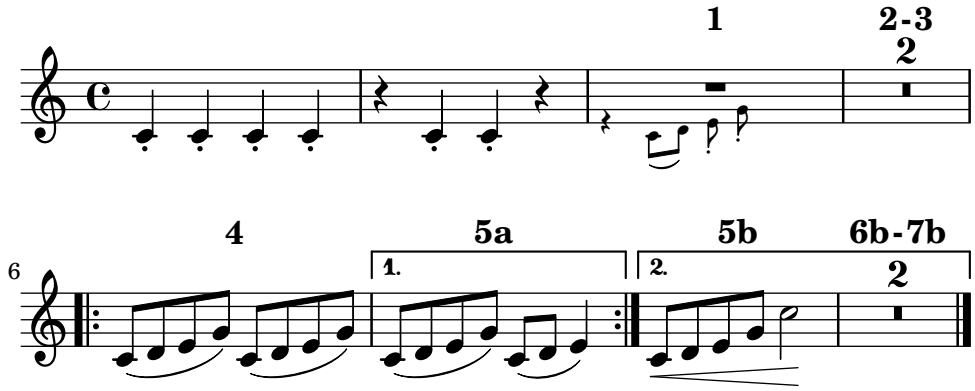
```
\layout {
  \enablePolymeter
}
```

  Independent of this, Default_bar_line_engraver has been removed.

- The new option visible-over-note-heads can be used to make tuplet brackets always appear when their direction is set to be over the note heads. It can be used with the default tuplet bracket visibility style or with #'if-no-beam.

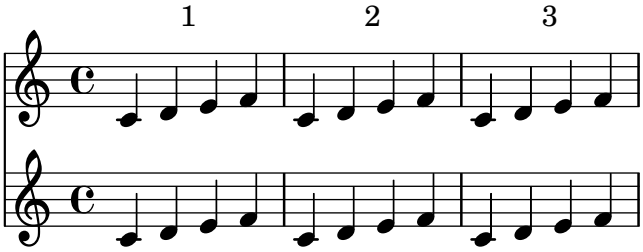


- Measure counts now take compressed multi-measure rests and alternatives into account.

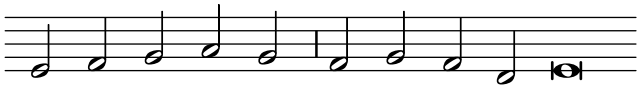


- Bar numbers may be centered in their measure, as is common in film scores.

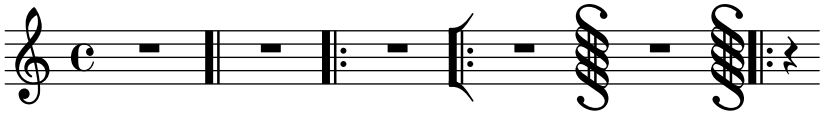
```
\layout {
  \context {
    \Score
    centerBarNumbers = ##t
    barNumberVisibility = #all-bar-numbers-visible
  }
}

<<
  { \repeat unfold 3 { c'4 d' e' f' } }
  { \repeat unfold 3 { c'4 d' e' f' } }
>>
```



- The alignment of bar numbers appearing in the middle or end of a system has been changed to align them on their left edge. This is in keeping with the advice of Elaine Gould (*Behind Bars*, p. 237), and was mostly the consensus reached in a discussion of the issue by developers. The alignment of bars at the beginning of a system remains unchanged.
- `\bar ","` creates a short bar line.



- The following predefined bar types no longer appear as a single bar line at the end of a line. Annotated bar types (e.g., `\bar "S-|"`) have been added for that purpose.

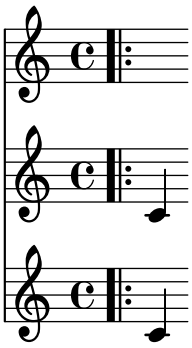


- `\bar ""` is no longer necessary to print the first bar number. It now suffices to set `barNumberVisibility` to `all-bar-numbers-visible`, or one of the other visibility settings where the first bar number is visible.

  Note that this is a change in behavior for scores that set `barNumberVisibility` to `all-bar-numbers-visible` or such and `BarNumber.break-visibility` to `#t` without having `\bar ""`. Now, a bar number is printed at the beginning. This is just the expected behavior (*all* bar numbers should be visible), but due to slightly unclear documentation, users may have used these settings to print bar numbers in the middle of systems except for the first bar number. In such cases, simply remove `\set Score.barNumberVisibility = #all-bar-numbers-visible` since `\override BarNumber.break-visibility = ##t` does the relevant setting alone.
- The `\break` command now always inserts a break, bypassing all default decisions about break points. For example, it is no longer necessary to insert `\bar ""` to obtain a mid-measure break.

  The new `\allowBreak` command inserts a possible break point, without forcing it, but bypassing default decisions like `\break` does.
- The bar line type `"-"` has been removed. `convert-ly` converts it to `""`. There is a slight difference in horizontal spacing at line breaks.
- `automaticBars` has been removed. `convert-ly` converts `automaticBars = ##f` to `measureBarType = #'()`.
- `\defineBarLine` now accepts `#t` in lieu of repeating the mid-line glyph name.
- `Bar_engraver` used to forbid line breaks between bar lines in all cases, but now it only does so when the `forbidBreakBetweenBarLines` context property is set to `#t`, which is the default. The `barAlways` context property, which previously worked around the lack of `forbidBreakBetweenBarLines`, has been removed.
- Due to changes in the internals of `\bar`, it is no longer supported to use it before creating lower contexts with `\new`. Such uses will now create an extra staff. This is similar to what happens with commands such as `\override Staff...` (see Section "An extra staff appears" in *Application Usage*).

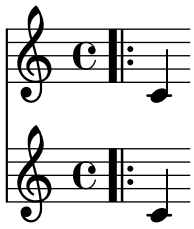
```
{
  \bar ".|:"
  <<
    \new Staff { c' }
    \new Staff { c' }
  >>
}
```



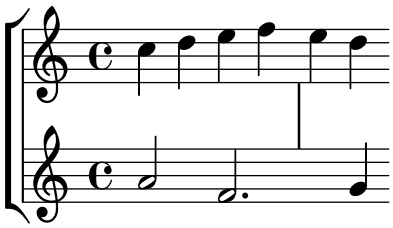
  The solution is to place `\bar` inside the music for each staff, as is usual with most commands.

```
<<
```

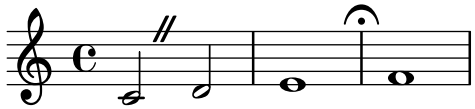
```
  \new Staff { \bar ".|:" c' }
  \new Staff { \bar ".|:" c' }
>>
```



- The bar type `"-span|"` creates a *mensurstrich*.



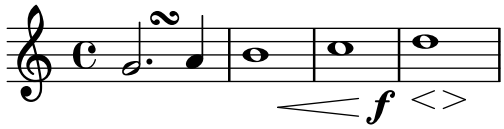
- `Staff` contexts use the new `Caesura_engraver` to notate the `\caesura` command.



Expressive mark improvements

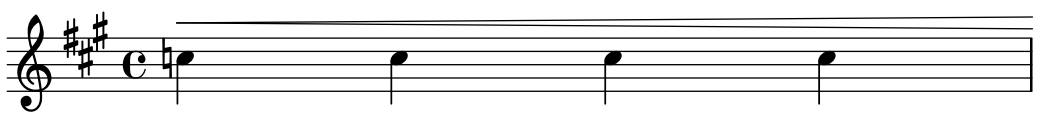
- Events attached to notes (e.g., dynamics or articulations) can be delayed by an arbitrary duration using `\after`. This simplifies many situations that previously required the use of explicit polyphony and spacer rests.

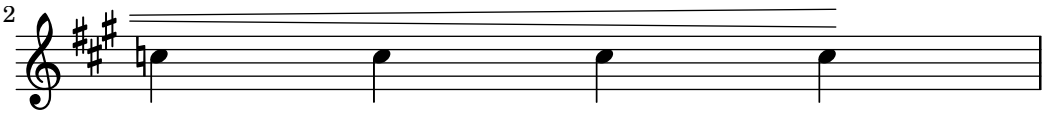
```
{
  \after 2 \turn g'2. a'4
  \after 2 \< b'1
  \after 2. \f c''
  <>\< \after 4 \> \after 2\! d''
}
```



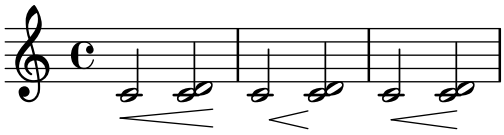
- Broken hairpins now have some left padding by default. This is in line with published scores and it fixes some cases where broken hairpins were vertically displaced by the key signature.

```
\relative {
  \key a \major
  c''4^\< c c c \break c c c c\! |
}
```



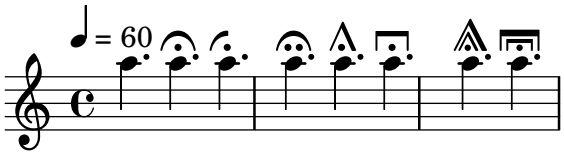


- The ends of hairpins may now be aligned to the `LEFT`, `CENTER` or `RIGHT` of `NoteColumn` grobs by overriding the property `endpoint-alignments`.

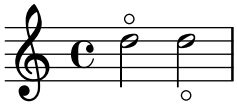


- The direction of a trill spanner can now be set with direction indicators like other articulations, i.e. with `_\startTrillSpan` or `^\startTrillSpan`.
- The default appearance of trill spanners has changed to better match classical engraving conventions. They now end before the next note, not over it. If the next note has an accidental, they stop before it. If the next note is the first note of a measure, they stop over the bar line instead.

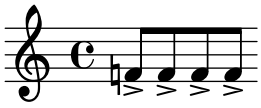
- The padding by default for fermatas is now larger. This avoids some cases where the fermata was placed too close to dots and other objects.



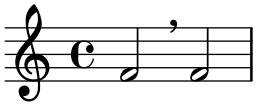
- The flageolet symbol is now smaller and slightly thicker. This is in line with published scores and makes the recommended workaround to make it smaller (`\tweak font-size -3 \flageolet`) unnecessary.



- The accent glyph is now a bit smaller. This fixes some cases where a natural sign would vertically displace accents.

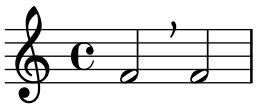


- The comma glyph shape, as used in the `\breathe` command, has been changed to a more common form.



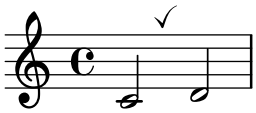
The old glyph remains available under the name ‘raltcomma’:

```
{
  \override BreathingSign.text =
    \markup { \musicglyph "scripts.raltcomma" }
  f'2 \breathe f' |
}
```



- The new context property `breathMarkType` selects the mark that `\breathe` produces from several predefined types.

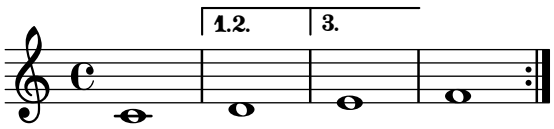
```
\fixed c' {
  \set breathMarkType = #'tickmark
  c2 \breathe d2
}
```



Repeat improvements

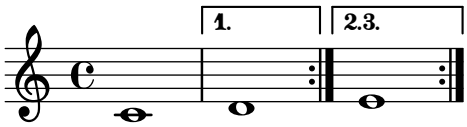
- Repeat alternatives may appear within the repeated section.

```
\repeat volta 3 { c'1 \alternative { d' e' } f' }
```



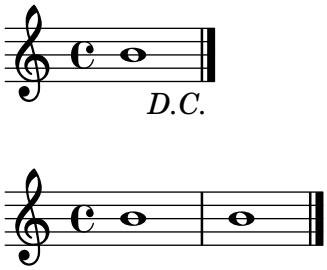
- The volta numbers for repeat alternatives may be set with the `\volta` command.

```
\repeat volta 3 c'1 \alternative { \volta 1 d' \volta 2,3 e' }
```



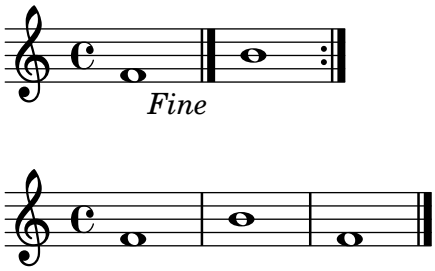
- The new `\repeat segno` command automatically notates a variety of *da-capo* and *dal-segno* forms.

```
music = \fixed c' {
  \repeat segno 2 {
    b1
  }
  \fine
}

\score { \music }
\score { \unfoldRepeats \music }
```



- The new `\fine` command inserts a final bar line that interacts gracefully with repeat bar lines. Used inside `\repeat`, it also prints *Fine* and ends the music after unfolding.

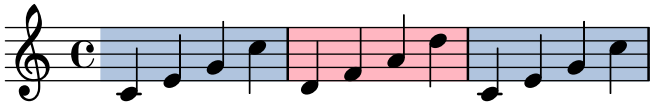
```
music = \fixed c' {
  \repeat volta 2 {
    f1
    \volta 2 \fine
    \volta 1 b1
  }
}

\score { \music }
\score { \unfoldRepeats \music }
```



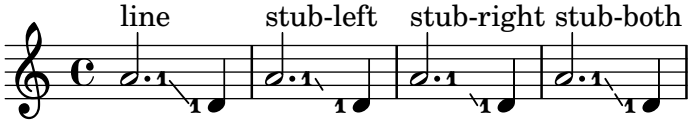
- The `\volta` command removes music when a repeat is unfolded.
- The `\unfolded` command adds music when a repeat is unfolded.

Editorial annotation improvements

- The new `\staffHighlight` and `\stopStaffHighlight` commands can be used to color a musical passage.

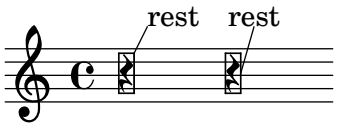


- A new grob `FingerGlideSpanner` is now available, indicating a finger gliding on a string from one to another position. Several appearances are possible, depending on the setting of `style`. Shown in the image are `line`, `stub-left`, `stub-right` and `stub-both`.

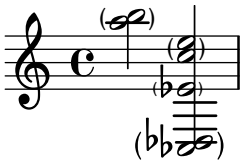


Also possible is `dashed-line`, `dotted-line`, `zigzag`, `trill`, `bow` and `none`.

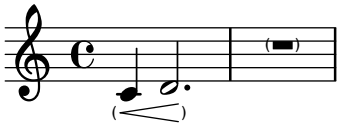
- Balloons now have changeable formatting.



- Parenthesizing chords is supported. Currently, the font size of the parentheses has to be adjusted manually.

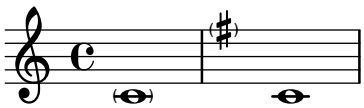


- Parenthesizing spanners is supported.



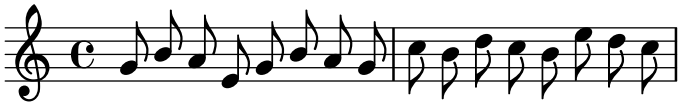
- A "time-based" version of the `\parenthesize` command was added. It takes a grob path: `\parenthesize` *`GrobName`* or `\parenthesize` *`ContextName.GrobName`*. It acts like a `\once \override`. This interface complements the already existing form `\parenthesize` *`event`*, in a fashion similar to `\footnote`.

```
{
  \parenthesize NoteHead
  c'1
  \parenthesize Staff.KeySignature
  \key g \major
  c'1
}
```



- Adding the `Melody_engraver` to the `Voice` context now works out of the box to change the stem direction of the middle note according to the melody. Previously, this required an additional override to `Stem.neutral-direction`.

```
\new Voice \with {
  \consists Melody_engraver
}
\relative c'' {
  \autoBeamOff
  g8 b a e g b a g |
  c b d c b e d c |
}
```



The suspendMelodyDecisions context property may be used to turn off this behavior temporarily, as \override Stem.neutral-direction = #DOWN used to do.

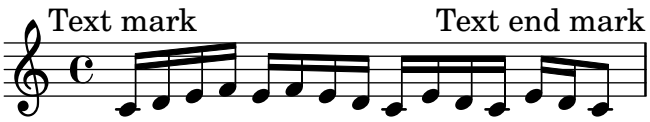
- The new Mark_tracking_translator takes over from Mark_engraver the decision of when to create a mark. Mark_engraver continues to control formatting and vertical placement.

  By default, Mark_engravers in multiple contexts create a common sequence of marks. If independent sequences are desired, multiple Mark_tracking_translators must be used.

Text improvements

- New commands \textMark and \textEndMark are available to add an arbitrary piece of text between notes, called a text mark. These commands improve over the previously existing syntax with the \mark command called as \mark *markup* (i.e., \mark "..." or \mark \markup ...).

```
\fixed c' {
  \textMark "Text mark"
  c16 d e f e f e d c e d c e d c8
  \textEndMark "Text end mark"
}
```

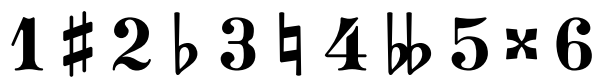


  \textMark and \textEndMark are now the recommended way to create textual marks. The use of \mark for this purpose is still supported, but discouraged (note that the \mark command itself is not discouraged, only calling it on a markup argument; \mark \default or \mark *number* is still the recommended and only way to create a rehearsal mark).

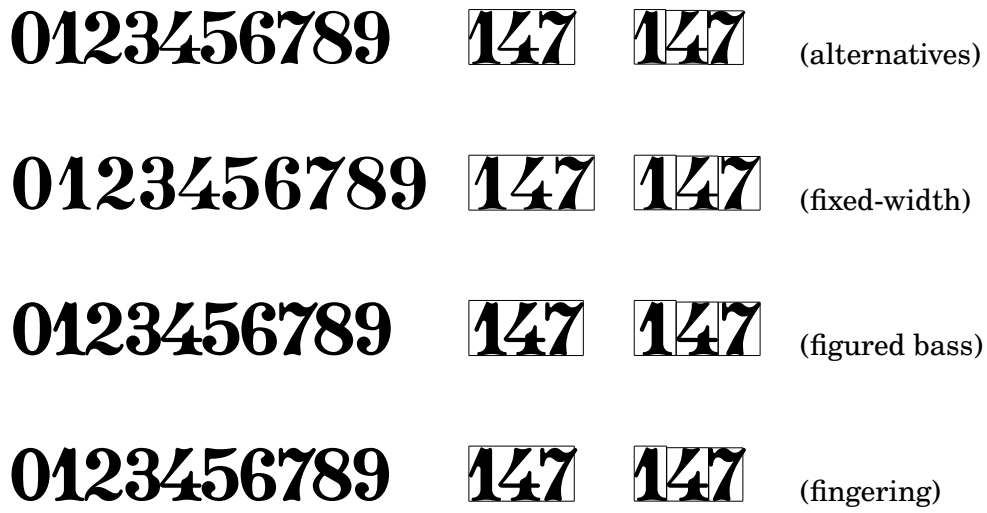
  The new commands have several differences to \mark *markup*. There can be an arbitrary number of them at a given moment, while there can only be one use of \mark. They output grobs of the dedicated TextMark type, whereas \mark creates a RehearsalMark grob regardless of whether it is called for a rehearsal mark or a textual mark; introducing this distinction allows stylesheets to set different layout settings for rehearsal marks and text marks. The alignment set by the new commands is different: \textMark always creates a left-aligned mark, and \textEndMark creates a right-aligned mark; in contrast, the alignment of a RehearsalMark depends on the anchor point of the object it aligns to.

  See Section "Text marks" in *Notation Reference* for full details.

- Text variant glyphs for sharp, flat, natural, double sharp, and double flat are now available in the Emmentaler fonts. In markup, they can be easily accessed with standard Unicode values.



- It is now possible to control the width and the shape of (some) Emmentaler digits using OpenType features.



- `\smallCaps` now works on any markup, not just on a bare string.
- The syntax for conditions in markups was made more flexible and user-friendly. It uses the new markup commands `\if` and `\unless`. Here are example replacements:

2.22 syntax	**2.24 syntax**
`\on-the-fly #first-page ...`	`\if \on-first-page ...`
`\on-the-fly #not-part-first-page ...`	`\unless \on-first-page-of-part ...`
`\on-the-fly #(on-page n) ...`	`\if \on-page #n ...`

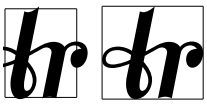
- With the new markup list command `string-lines` it is now possible to split a string at a given character. The default is to split at line break. Surrounding white space gets dropped. The resulting list of markups may be further formatted. This is a very convenient way of entering additional stanzas for songs.

  Twinkle, twinkle, little star,
  How I wonder what you are!
  Up above the world so high,
  Like a diamond in the sky.
  Twinkle, twinkle, little star,
  How I wonder what you are!

- The new markup command `\align-on-other` translates a markup as if it was aligned to another markup.

  1
  12
  12345
  123

- Two new markup functions `\with-dimension` and `\with-dimension-from` are available. They are similar to `\with-dimensions` and `\with-dimensions-from`, respectively, modifying only a single dimension (instead of both).
- New markup commands `\with-true-dimension` and `\with-true-dimensions` are available. They give the markup the actual extent(s) of its printed ink, which may differ from the default extents for some font glyphs due to text regularity constraints.



- Text replacements can now replace strings with any markup, not just with a string.

```
\markup
  \replace #`(("2nd" . ,#{ \markup \concat { 2 \super nd } #}))
  "2nd time"
```

2nd time

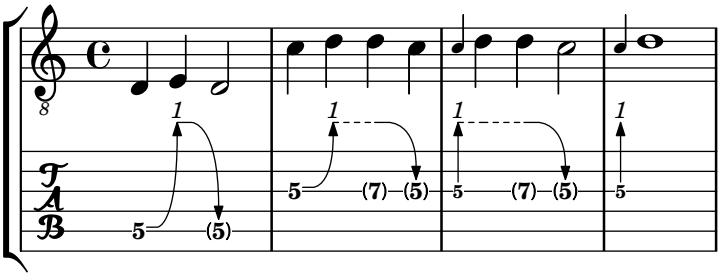
- A new markup command `\with-string-transformer` is available. It interprets a markup with a "string transformer" installed; the transformer is called when the interpretation of the markup requires interpreting a string, and allows to perform modifications on this string, such as changing case.
- The `markup->string` function converts a markup into an approximate string representation; it is used for outputting PDF metadata as well as MIDI lyrics and markers. Markup commands can now define a custom method to convert markups created using them into strings, for use by `markup->string`. For example:

```
#(define-markup-command (upcase layout props arg) (string?)
   #:as-string (string-upcase arg)
   (interpret-markup layout props (string-upcase arg)))
```

New for specialist notation

Fretted string instrument improvements

- The string tunings `banjo-double-c` and `banjo-double-d` were added.
- A new grob `BendSpanner` is now available for `TabStaff`, indicating a bent string. Apart from the default three styles are possible: `'hold`, `'pre-bend` and `'pre-bend-hold`.



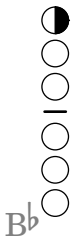
Percussion improvements

- The drum notation style `weinberg-drums-style` was added. It is based on Norman Weinberg's standardization work.

Wind instrument improvements

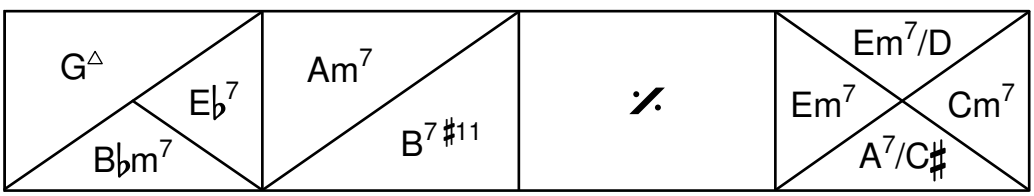
- Additional display details of a `\woodwind-diagram` can now be specified including the angle of partially-covered-keys and the display of non-graphical trill keys.

```
\markup {
  \override #'(graphical . #f)
  \override #'(woodwind-diagram-details . ((fill-angle . 90)
                                           (text-trill-circled . #f)))
  \woodwind-diagram #'flute #'((cc . (one1h))
                               (lh . ())
                               (rh . (besT)))
}
```



Chord notation improvements

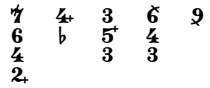
- Support for chord grids has been added.



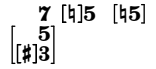
- In `ChordNames`, multi-measure rests now also cause the "N.C." symbol to be printed, just like normal rests.
- In figured bass, _ now creates an empty figure that still takes up space.

```
\figures {
  <8 _ 4]> <_ 5+ 3>
}
```

- Formatting of figured bass has been improved. In particular, the default size is reduced to a value used by many Urtext editions of Baroque music.
- In figured bass, specially designed glyphs for `6\\`, `7\\`, and `9\\` are now used by default. Similarly, specially designed glyphs for symbols `2\+`, `4\+`, and `5\+` are used by default if plus signs appear after the number.



  Use the new command `\figured-bass` to access these glyphs in markup.
- In figured bass, brackets can now also be added around accidentals.

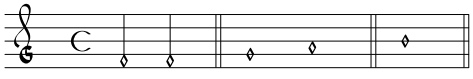


Ancient notation improvements

- A new context `VaticanaLyrics` is available. It is similar to `Lyrics`, providing a hyphenation style (a single, flush-left hyphen between two syllables) as used in the notational style of Editio Vaticana.
- The predefined commands for Gregorian divisiones are no longer variations on `\breathe`. `\divisioMinima`, `\divisioMaior`, `\divisioMaxima`, and `\virgula` are variations on the basic `\caesura`. `\finalis` is equivalent to `\section`.

  `MensuralStaff` and `VaticanaStaff` use `Divisio_engraver` to interpret the above commands as well as `\repeat volta` and `\fine`.

```
\new MensuralStaff \fixed c' {
  \repeat volta 2 { f2 f }
  g1
  a1 \section
  b1 \fine
}
```

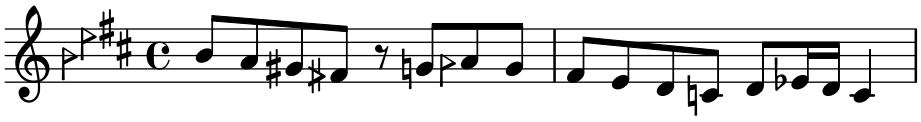


- `KievanStaff`, `MensuralStaff`, `PetrucciStaff`, and `VaticanaStaff` now allow line breaks anywhere, and they no longer create `""` measure bar lines.
- In `GregorianTranscriptionStaff`, divisiones are now engraved as `BarLine` grobs by default. To change them to `Divisio` grobs, use `\EnableGregorianDivisiones`.
- `GregorianTranscriptionStaff` allows a line break after any note and no longer uses `Time_signature_engraver`.
- `GregorianTranscriptionVoice` no longer uses `Stem_engraver`.

World music improvements

- Support for Persian classical music is now available. For this, two accidental glyphs, *sori* and *koron*, have been added to LilyPond.

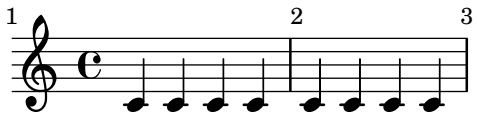
```
\include "persian.ly"

\relative c' {
  \key d \chahargah
  bk'8 a gs fo r g ak g |
  fs ek d c d ef16 d c4 |
}
```



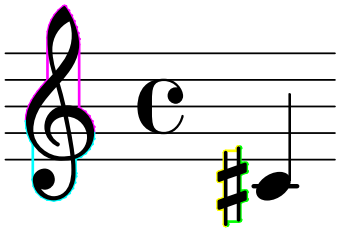
Miscellaneous improvements

- In the Emmentaler font, identical-looking noteheads whose only difference was stem direction have been consolidated into a single glyph. For instance, the glyphs `noteheads.u2triangle` and `noteheads.d2triangle` have been replaced by a single glyph, `noteheads.s2triangle`. Notehead pairs that look visually different depending on the direction remain distinct.

  In addition, the `stem-attachment` property of `NoteHead` grobs now returns its actual, direction-dependent stem attachment point instead of a hypothetical upwards-stem attachment point.
- Two redundant glyphs in the Emmentaler font have been removed: `scripts.trillelement` (use `scripts.trill_element` instead) and `scripts.augmentum` (use `dots.dotvaticana` instead).
- Using `\paper { bookpart-level-page-numbering = ##t}`, it is now possible to make bookparts independent with respect to page numbering. If this is used for all bookparts, each bookpart has its own numbering sequence, starting at 1 by default. It can also be used in an individual bookpart, which is useful to achieve the standard practice of numbers pages in an analytical introduction independently and in roman numerals (the latter is achieved using `page-number-type = #'roman-lower`).
- A new grob callback function `break-alignment-list` is now available for returning different values depending on a grob's break direction. As an example, use it to provide different alignments of a grob depending on whether it is positioned at the beginning, the middle, or the end of a line.

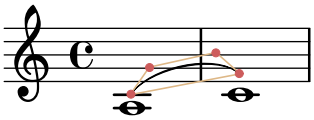


- The new `Mark_performer` creates MIDI Marker events like `Mark_engraver` creates printed marks.
- Properties of `PaperColumn` and `NonMusicalPaperColumn` (such as `NonMusicalPaperColumn.line-break-system-details`) can now be overridden mid-music with the usual command `\once \override`. They used to be a special case requiring the `\overrideProperty` command.
- The new `show-horizontal-skylines` and `show-vertical-skylines` properties allow to display an object's skylines. This is more flexible than the already existing `debug-skylines` option because it works for all grobs. While primarily meant for debugging LilyPond, this can be useful when trying to understand spacing decisions or overriding stencils in Scheme.



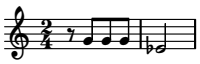
- The new command `\vshape` is like `\shape`, but also shows the control points and polygon for easier tweaking.

```
{ a1\vshape #'((0 . 0) (0 . 0.5) (0 . 0.9) (0 . 0.4))^( c'1) }
```



- `\markup \path` now also works in SVG output even if the path does not begin with a `moveto` or `rmoveto` command. Also, it now accepts single-letter SVG equivalents (`moveto` = `M`, etc.).
- `set-default-paper-size` and `set-paper-size` now accept a custom paper size.

```
#(set-default-paper-size '(cons (* 100 mm) (* 50 mm)))
```

- `lilypond-book` supports two new music fragment options `paper-width` and `paper-height` to set a custom paper size.
- `lilypond-book` supports a new snippet option `inline` for inline music, that is, music snippets like  that appear within a paragraph of text.
- The `lilypond-book` script now allows braces in the argument of the commands `\lilypond` (for LaTeX) and `@lilypond` (for Texinfo).
- `lilypond-book` now appends the current directory as the last entry to search for included files, instead of prepending it to the list of specified include paths. This allows include directories to shadow files from the current directory, and will only be noticed if there are files with the same name in both.
- The new Scheme function `universal-color` provides an eight-element color palette designed to be unambiguous to people with dichromatism.

black
orange
skyblue
bluegreen
yellow
blue
vermillion
redpurple

- The `-dembed-source-code` option now also embeds images added with `\epsfile` and files included with `\verbatim-file`.
- The default of the `aux-files` program option changed to `#f`. If you are calling LilyPond with the `-dbackend=eps` argument and need the auxiliary `.tex` and `.texi` files, you now have to specify `-daux-files` explicitly. The formats for `lilypond-book` images can be set separately for the tall page image (typically PNG for HTML output) and per-system images (typically, EPS or PDF for printed output) with the `-d` sub-options `-dtall-page-formats` and `-dseparate-page-formats` respectively.
- The 'big point' unit (1 bp = 1/72 in) is now available by appending `\bp` to length values.
- Scheme-defined translators usable in both '`\layout`' and '`\midi`' can now be created with `make-translator`. Scheme-defined performers usable only in '`\midi`' can now be created with `make-performer`. Those macros work strictly like the previously existing macro `make-engraver` for creating engravers only usable in '`\layout`'.
- Scheme translators can now define a new slot called `pre-process-music`. It is called on all translators, after all listeners but before all `process-music` slots. This can be used for processing that depends on all events heard but needs to set context properties before other translators read them.

- Scheme translators can now contain listeners written as

```
(listeners
 ((event-class engraver event #:once)
  ...))
```

  These are never triggered more than once per time step. They emit a warning if they receive two events in the same time step, except if the events are equal.

- The same grob definition can now be used to create grobs of different classes (`Item`, `Spanner`, `Paper_column`, `System`). As part of this change, the grob types `FootnoteItem` and `FootnoteSpanner` were consolidated into a single type `Footnote`. Similarly, `BalloonTextSpanner` and `BalloonTextItem` are unified into `BalloonText`.

  When the grob definition does not mandate a class, engravers should choose what class to create a grob with. For authors of Scheme engravers, this means using either `ly:engraver-make-item` or `ly:engraver-make-spanner`. The utility function `ly:engraver-make-sticky` is provided to support the frequent case of *sticky* grobs, such as footnotes and balloons. It creates a grob with the same class as another grob and administrates parents and bounds.

- The new command-line option `-dcompile-scheme-code`, also settable in the LilyPond input with `#(ly:set-option 'compile-scheme-code)`, provides with better diagnostics when running Scheme code leads to an error. Internally, this uses the byte-compiler provided by Guile, instead of the interpreter.

  However, due to a limitation in Guile, this currently has the disadvantage of making it impossible to run more than a few thousand Scheme expressions. Also, be aware that the Guile compiler has a few differences to the interpreter. For example, constant parts of quasiquotes are made actual constants more aggressively, making code such as `(let ((x 4)) (sort! `(,x 3 2 1)))` produce an error because the "cdr" of the quasiquoted list is constant, and it is an error in Scheme to mutate literal data. (In this specific case, the code could avoid the issue by using the non-destructive `sort`, or by creating a fresh list each time with `(list x 3 2 1)`.)

  Furthermore, this option does not currently work on Windows.

Notes on Guile 2.2

This version of LilyPond switches from Guile 1.8 to Guile 2.2. This section lists some of the most common incompatibilities that you could have to deal with in order to upgrade your Scheme code.

A full, detailed log of changes in Guile can be found in the `NEWS` file (`https://git.savannah.gnu.org/cgit/guile.git/tree/NEWS`) of the Guile source.

- The `format` function now requires a boolean or port as the first argument. This argument was optional in Guile 1.8. In order to make the function return the formatted output as a string, like `format` does without this argument in Guile 1.8, pass `#f` for this argument, i.e., `(format #f "`*`string`*`" `*`arguments`*` ...)` instead of `(format "`*`string`*`" `*`arguments`*` ...)`.
- The rules for internal (i.e., non-toplevel) definitions have become stricter. Definitions are no longer allowed in various expression contexts. This is no longer valid, for example:

  ```
  (if (not (defined? 'variable))
      (define variable 'value))
  ```

  The solution in this particular example is:

  ```
  (define variable
    (if (not (defined? 'variable))
        'value
        variable))
  ```

- Strings now support Unicode characters. Previously, a Unicode character was represented by several characters, and various functions were not tailored for Unicode support.
- Some numeric functions now return exact results in more cases. For instance, `(sqrt 4)` returns `2.0` in Guile 1.8, but `2` (an integer) in Guile 2.2.