

## New features in 2.20 since 2.18

- Accidental rules can now be defined across `ChoirStaff` contexts. Two new rules `choral` and `choral-cautionary` are available that combine the characteristics of `modern-voice` and `piano` or their equivalents with cautionary accidentals.
- The music function `\\unfoldRepeats` can now take an optional argument-list specifying which type(s) of repeated music should be unfolded. Possible entries are `percent`, `tremolo`, `volta`. If the optional argument-list is unspecified, `repeated-music` will be used, unfolding all.
- A new `output-attributes` grob property is now used for svg output instead of the `id` grob property. It allows multiple attributes to be defined as an association list. For example, `#'((id . 123) (class . foo) (data-whatever . \bar"))` will produce the following group tag in an SVG file: `<g id=\123" class=\foo" data-whatever=\bar"> ... </g>`.
- Slurs and phrasing slurs may now be started from individual notes in a chord. Several simultaneous slurs per `Voice` need to be distinguished by `spanner-id` setting.
- The music and grob property `spanner-id` for distinguishing simultaneous slurs and phrasing slurs has been changed from a string to a ‘key’, a non-negative integer or symbol.
- There is a new command `\=` for specifying the `spanner-id` for simultaneous slurs and phrasing slurs.

```
\fixed c' {  
  <c~ f\=1( g\=2( >2 <c e\=1) a\=2) >  
}
```



- Blocks introduced with `\header` can be stored in variables and used as arguments to music and scheme functions and as the body of `#{...#}` constructs. They are represented as a Guile module.

While `\book`, `\bookpart`, `\score`, `\with`, `\layout`, `\midi`, `\paper` blocks can be passed around in similar manner, they are represented by different data types.

- Dot-separated symbol lists like `FretBoard.stencil` were already supported as of version 2.18. They may now also contain unsigned integers, and may alternatively be separated by commas. This allows usage such as

```
{ \time 2,2,1 5/8 g'8 8 8 8 8 }
```



and

```
\tagGroup violin,oboe,bassoon
```

- Such lists may also be used in expressions for assignments, sets, and overrides. This allows usage such as

```
{ \unset Timing.beamExceptions  
  \set Timing.beatStructure = 1,2,1  
  g'8 8 8 8 8 8 8 }
```



- Association list elements could previously be assigned values individually (for example, paper variables like `system-system-spacing.basic-distance`). They may now be also referenced in this manner, as with

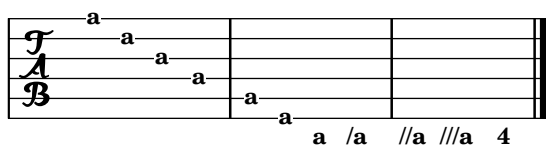
```
\paper {
  \void \displayScheme \system-system-spacing.basic-distance
}
```

In combination with the previously mentioned changes, this allows setting and referencing pseudovariables like `violin.1`.

- LilyPond source files may now be embedded inside the generated PDF files. This experimental feature is disabled by default and may be regarded as unsafe, as PDF documents with hidden content tend to present a security risk. Please note that not all PDF viewers have the ability to handle embedded documents (if not, the PDF output will appear normally and source files will remain invisible). This feature only works with the PDF backend.
- French note names are now defined specifically instead of being aliased to Italian note names: in addition to the generic Italian-derived syntax, the *d* pitch may be entered as *ré*. Double sharps may also be entered using the `-x` suffix.
- Additional bass strings (for lute tablature) are supported.

```
m = { f'4 d' a f d a, g, fis, e, d, c, \bar "|." }

\score {
  \new TabStaff \m
  \layout {
    \context {
      \Score
      tablatureFormat = #fret-letter-tablature-format
    }
    \context {
      \TabStaff
      stringTunings = \stringTuning <a, d f a d' f'>
      additionalBassStrings = \stringTuning <c, d, e, fis, g,>
      fretLabels = #("a" "b" "r" "d" "e" "f" "g" "h" "i" "k")
    }
  }
}
```



- The markup-list-command `\table` is now available. Each column may be aligned differently.

```
\markuplist {
  \override #'(padding . 2)
  \table
  #'(0 1 0 -1)
  {
    \underline { center-aligned right-aligned center-aligned left-aligned }
    one "1" thousandth "0.001"
```

```

eleven "11" hundredth "0.01"
twenty "20" tenth "0.1"
thousand "1000" one "1.0"
}
}

```

center-aligned right-aligned center-aligned left-aligned

one	1	thousandth	0.001
eleven	11	hundredth	0.01
twenty	20	tenth	0.1
thousand	1000	one	1.0

- A new markup-command, `\with-dimensions-from`, makes `\with-dimensions` easier to use by taking the new dimensions from a markup object, given as first argument.

```

\markup {
  \pattern #5 #Y #0 "x"
  \pattern #5 #Y #0 \with-dimensions-from "x" "f"
  \pattern #5 #Y #0 \with-dimensions-from "x" "g"
  \override #'(baseline-skip . 2)
  \column {
    \pattern #5 #X #0 "n"
    \pattern #5 #X #0 \with-dimensions-from "n" "m"
    \pattern #5 #X #0 \with-dimensions-from "n" "!"
  }
}

```

```

x f g nnnnn
x f g mmmmm
x f g !!!!!

```

- There are two new page breaking functions. `ly:one-page-breaking` automatically adjusts the height of the page to fit the music, so that everything fits on one page. `ly:one-line-auto-height-breaking` is like `ly:one-line-breaking`, placing the music on a single line and adjusting the page width accordingly, however it also automatically adjusts the page height to fit the music.
- Markup-command `\draw-squiggle-line` is now available. Customizing is possible with overrides of thickness, angularity, height and orientation

```

\markup
\overlay {
  \draw-squiggle-line #0.5 #'(3 . 3) ##t

  \translate #'(3 . 3)
  \override #'(thickness . 4)
  \draw-squiggle-line #0.5 #'(3 . -3) ##t

  \translate #'(6 . 0)

```

```

\override #'(angularity . -5)
\draw-squiggle-line #0.5 #'(-3 . -3) ##t

\translate #'(3 . -3)
\override #'(angularity . 2)
\override #'(height . 0.3)
\override #'(orientation . -1)
\draw-squiggle-line #0.2 #'(-3 . 3) ##t
}

```



- A new command, `\RemoveAllEmptyStaves`, has been made available, which acts exactly like `\RemoveEmptyStaves`, except for also removing empty staves on the first system in a score.
- Markup-commands `\undertie` and `\overtie` are now available, as well as the generic markup-command `\tie`.

```

\markup {
  \undertie "undertied"
  \overtie "overtied"
}

```

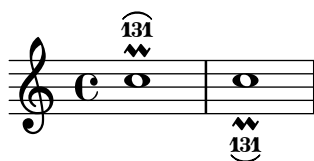
```

m = {
  c''1 \prall -\tweak text \markup \tie "131" -1
}

```

```
{ \voiceOne \m \voiceTwo \m }
```

undertied overtied



- `TabStaff` is now able to print micro-tones for bendings etc.

```

\layout {
  \context {
    \Score
    supportNonIntegerFret = ##t
  }
}

```

```
mus = \relative { c'4 cih d dih }
```

```

<<
  \new Staff << \clef "G_8" \mus >>
  \new TabStaff \mus
>>

```



- Two new styles of whiteout are now available. The **outline** style approximates the contours of a glyph's outline, and its shape is produced from multiple displaced copies of the glyph. The **rounded-box** style produces a rounded rectangle shape. For all three styles, including the default box style, the whiteout shape's **thickness**, as a multiple of staff-line thickness, can be customized.

```
\markup {
  \combine
    \filled-box #'(-1 . 15) #'(-3 . 4) #1
    \override #'(thickness . 3)
    \whiteout whiteout-box
}
\markup {
  \combine
    \filled-box #'(-1 . 24) #'(-3 . 4) #1
    \override #'(style . rounded-box)
    \override #'(thickness . 3)
    \whiteout whiteout-rounded-box
}
\markup {
  \combine
    \filled-box #'(-1 . 18) #'(-3 . 4) #1
    \override #'(style . outline)
    \override #'(thickness . 3)
    \whiteout whiteout-outline
}
\relative {
  \override Staff.Clef.whiteout-style = #'outline
  \override Staff.Clef.whiteout = 3
  g'1
}
```

whiteout-box

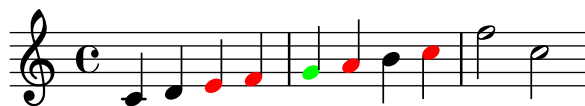
whiteout-rounded-box

whiteout-outline



- All of `\override`, `\revert`, `\set`, and `\unset` now work with the `\once` prefix for making one-time settings.

```
\relative {
  c'4 d
  \override NoteHead.color = #red
  e4 f |
  \once \override NoteHead.color = #green
  g4 a
  \once \revert NoteHead.color
  b c |
  \revert NoteHead.color
  f2 c |
}
```



- When outputting MIDI, LilyPond will now store the `title` defined in a score's `\header` block (or, if there is no such definition on the `\score` level, the first such definition found in a `\header` block of the score's enclosing `\bookpart`, `\book`, or top-level scope) as the name of the MIDI sequence in the MIDI file. Optionally, the name of the MIDI sequence can be overridden using the new `midititle` `\header` field independently of `title` (for example, in case `title` contains markup code which does not render as plain text in a satisfactory way automatically).
- Music (and scheme and void) functions and markup commands that just supply the final parameters to a chain of overrides, music function and markup command calls can now be defined in the form of just writing the expression cut short with `\etc`.

```
bold-red-markup = \markup \bold \with-color #red \etc
highlight = \tweak font-size 3 \tweak color #red \etc
```

```
\markup \bold-red "text"
\markuplist \column-lines \bold-red { One Two }
```

```
{ c' \highlight d' e'2-\highlight -! }
```

**text**

**One**

**Two**



- LilyPond functions defined with `define-music-function`, `define-event-function`, `define-scheme-function` and `define-void-function` can now be directly called from Scheme as if they were genuine Scheme procedures. Argument checking and matching will still be performed in the same manner as when calling the function through LilyPond

input. This includes the insertion of defaults for optional arguments not matching their predicates. Instead of using `\default` in the actual argument list for explicitly skipping a sequence of optional arguments, `*unspecified*` can be employed.

- Current input location and parser are now stored in GUILE fluids and can be referenced via the function calls `(*location*)` and `(*parser*)`. Consequently, a lot of functions previously taking an explicit `parser` argument no longer do so.

Functions defined with `define-music-function`, `define-event-function`, `define-scheme-function` and `define-void-function` no longer use `parser` and `location` arguments.

With those particular definitions, LilyPond will try to recognize legacy use of `parser` and `location` arguments, providing backwards-compatible semantics for some time.

- In the "english" notename language, the long notenames for pitches with accidentals now contain a hyphen for better readability. You now have to write

```
\key a-flat \major
```

instead of the previous

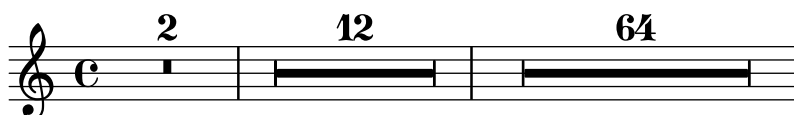
```
\key aflat \major
```

Double accidentals do not get another hyphen, so the Dutch `cisis` has the long English notename `c-sharpsharp`.

- The visual style of tremolo slashes (shape, style and slope) is now more finely controlled.

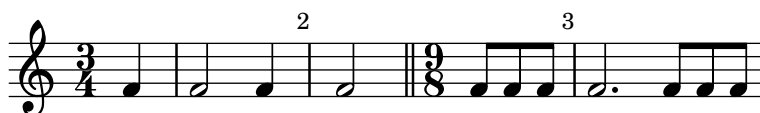


- Multi-measure rests have length according to their total duration, under the control of `MultiMeasureRest.space-increment`.



- Page numbers may now be printed in roman numerals, by setting the `page-number-type` paper variable.
- It is now possible to use `\time` and `\partial` together to change the time signature in mid measure.

```
\override Score.BarNumber.break-visibility = #end-of-line-invisible
\partial 4 \time 3/4 f4 | 2 4 | 2 \bar "||"
\time 9/8 \partial 4. f8 8 8 | 2. 8 8 8 |
```



- It is now possible to override the `text` property of chord names.

```
<<
\new ChordNames \chordmode {
  a' b c:7
  \once \override ChordName.text = #"foo"
  d
```

```
}
>>
```

A B C<sup>7</sup> foo

- Improved horizontal alignment when using `TextScript`, with `DynamicText` or `LyricText`.
- A new command `\magnifyStaff` has been added which scales staff sizes, staff lines, bar lines, beamlets and horizontal spacing generally at the `Staff` context level. Staff lines are prevented from being scaled smaller than the default since the thickness of stems, slurs, and the like are all based on the staff line thickness.
- `InstrumentName` now supports `text-interface`.
- There is now support for controlling the ‘expression level’ of MIDI channels using the `Staff.midiExpression` context property. This can be used to alter the perceived volume of even sustained notes (albeit in a very ‘low-level’ way) and accepts a number value between 0.0 and 1.0.

```
\score {
  \new Staff \with {
    midiExpression = #0.6
    midiInstrument = #"clarinet"
  }
  <<
  { a'1~ a'1 }
  {
    \set Staff.midiExpression = #0.7 s4\f\<
    \set Staff.midiExpression = #0.8 s4
    \set Staff.midiExpression = #0.9 s4
    \set Staff.midiExpression = #1.0 s4

    \set Staff.midiExpression = #0.9 s4\>
    \set Staff.midiExpression = #0.8 s4
    \set Staff.midiExpression = #0.7 s4
    \set Staff.midiExpression = #0.6 s4\!
  }
  >>
  \midi { }
}
```

- Support for making it easier to use alternative ‘music’ fonts other than the default Emmentaler in LilyPond has been added. See <http://fonts.openlilylib.org/> for more information.
- Grobs and their parents can now be aligned separately allowing more flexibility for grob positions. For example the ‘left’ edge of a grob can now be aligned on the ‘center’ of its parent.
- Improvements to the `\partial` command have been made to avoid problems when using multiple, parallel contexts.
- `\chordmode` can now use `< >` and `<< >>` constructs.
- A new command `\tagGroup` has now been added. This complements the existing `\keepWithTag` and `\removeWithTag` commands. For Example:

```
\tagGroup #'(violinI violinII viola cello)
```

declares a list of ‘tags’ that belong to a single ‘tag group’.

```
\keepWithTag #'violinI
```



Is now only concerned with ‘tags’ from ‘violinI’'s tag group.

Any element of the included music tagged with one or more tags from the group, but *not* with *violinI*, will be removed.

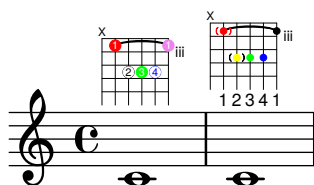
- The `\addlyrics` function now works with arbitrary contexts including **Staff**.
- String numbers can now also be used to print roman numerals (e.g. for unfretted string instruments).

```
c2\2
\romanStringNumbers
c\2
\arabicStringNumbers
c1\3
```



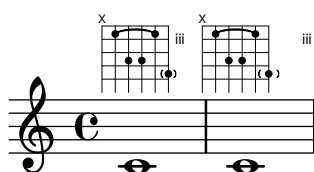
- The `thin-kern` property of the `BarLine` grob has been renamed to `segno-kern`.
- `KeyCancellation` grobs now ignore cue clefs (like `KeySignature` grobs do).
- Add support for `\once` `\unset`
- It is now possible to individually color both the dots and parentheses in fret diagrams when using the `\fret-diagram-verbose` markup command.

```
\new Voice {
  c1^\markup {
    \override #'(fret-diagram-details . (
      (finger-code . in-dot))) {
      \fret-diagram-verbose #'((mute 6)
        (place-fret 5 3 1 red)
        (place-fret 4 5 2 inverted)
        (place-fret 3 5 3 green)
        (place-fret 2 5 4 blue inverted)
        (place-fret 1 3 1 violet)
        (barre 5 1 3 ))
      )
    }
  }
  c1^\markup {
    \override #'(fret-diagram-details . (
      (finger-code . below-string))) {
      \fret-diagram-verbose #'((mute 6)
        (place-fret 5 3 1 red parenthesized)
        (place-fret 4 5 2 yellow
          default-paren-color
          parenthesized)
        (place-fret 3 5 3 green)
        (place-fret 2 5 4 blue )
        (place-fret 1 3 1)
        (barre 5 1 3))
      )
    }
  }
}
```



- Two new properties have been added for use in `fret-diagram-details` when using the `\fret-diagram-verbose` markup command; `fret-label-horizontal-offset` which affects the `fret-label-indication` and `paren-padding` which controls the space between the dot and the parentheses surrounding it.

```
\new Voice {
  c1^\markup {
    \fret-diagram-verbose #'((mute 6)
                        (place-fret 5 3 1)
                        (place-fret 4 5 2)
                        (place-fret 3 5 3)
                        (place-fret 1 6 4 parenthesized)
                        (place-fret 2 3 1)
                        (barre 5 2 3))
  }
  c1^\markup {
    \override #'(fret-diagram-details . (
      (fret-label-horizontal-offset . 2)
      (paren-padding . 0.25))) {
      \fret-diagram-verbose #'((mute 6)
                          (place-fret 5 3 1)
                          (place-fret 4 5 2)
                          (place-fret 3 5 3)
                          (place-fret 1 6 4 parenthesized)
                          (place-fret 2 3 1)
                          (barre 5 2 3))
    }
  }
}
```



- A new markup command `\justify-line` has been added. Similar to the `\fill-line` markup command except that instead of setting *words* in columns, the `\justify-line` command balances the whitespace between them ensuring that when there are three or more words in a markup, the whitespace is always consistent.

```
\markup \fill-line {ooooooo oooooo oooooo oooooo}
\markup \fill-line {ooooooooo ooooooooo oo ooo}

ooooooo      oooooo      oooooo      oooooo

oooooooooooo  oooooooooo      oo      ooo

\markup \justify-line {ooooooo oooooo oooooo oooooo}
\markup \justify-line {ooooooooo ooooooooo oo ooo}

ooooooo      oooooo      oooooo      oooooo
```

000000000

00000000

00

000

- A new command `\magnifyMusic` has been added, which allows the notation size to be changed without changing the staff size, while automatically scaling stems, beams, and horizontal spacing.

```

\new Staff <<
  \new Voice \relative {
    \voiceOne
    <e' e'>4 <f f'>8. <g g'>16 <f f'>8 <e e'>4 r8
  }
  \new Voice \relative {
    \voiceTwo
    \magnifyMusic 0.63 {
      \override Score.SpacingSpanner.spacing-increment = #(* 1.2 0.63)
      r32 c'' a c a c a c r c a c a c a c
      r c a c a c a c a c a c a c a c
    }
  }
>>

```



- A new flexible template suitable for a range of choral music, is now provided. This may be used to create simple choral music, with or without piano accompaniment, in two or four staves. Unlike other templates, this template is ‘built-in’, which means it does not need to be copied and edited: instead it is simply `\include`’d in the input file. For details, see Section “Built-in templates” in *Learning Manual*.
- The positioning of tuplet numbers for kneed beams has been significantly improved. Previously, tuplet numbers were placed according to the position of the tuplet bracket, even if it was not printed. This could lead to stranded tuplet numbers. Now they are now positioned closer to the kneed-beam when an appropriate beam segment exists for its placement and when the bracket is not drawn.

Collision detection is also added, offsetting horizontally if too close to an adjoining note column but preserving the number’s vertical distance from the kneed beam. If the number itself is too large to fit in the available space the original, bracket-based, positioning system is used instead; and in the event of a collision (e.g. with an accidental) the tuplet number is moved vertically away instead.

```

\time 3/4
\override Beam.auto-knee-gap = 3
\tuplet 3/2 4 {
  g8 c'' e,
  c'8 g,, e''
  g,,8 e''' c,,
}

```



The original kneed-beam tuplet behavior is still available through an `\override` via a new, `knee-to-beam` property.

```
\time 3/4
\override Beam.auto-knee-gap = 3
\override TupletNumber.knee-to-beam = ##f
\tuplet 3/2 4 {
  g8 c'' e,
  c'8 g,, e''
  g,,8 e''' c,,
}
```



- `\lyricsto` and `\addLyrics` have been ‘harmonized’. Both now accept the same kind of delimited argument list that `\lyrics` and `\chords` accept. Backward compatibility has been added so music identifiers (i.e. `\mus`) are permitted as arguments. A `convert-ly` rule has been added that removes redundant uses of `\lyricmode` and rearranges combinations with context starters such that `\lyricsto` in general is applied last (i.e. like `\lyricmode` would be).
- Scheme functions and identifiers can now be used as output definitions.
- Scheme expressions can now be used as chord constituents.
- Improved visual spacing of small and regular ‘MI’ Funk and Walker noteheads so they are now the same width as other shaped notes in their respective sets. SOL noteheads are also now visually improved when used with both the normal Aiken and Sacred Harp heads, as well as with the thin variants.
- `LeftEdge` now has a definable Y-extent (i.e. vertical). See Section “LeftEdge” in *Internals Reference*.
- Added a new `make-path-stencil` function that supports all `path` commands both relative and absolute:  
`lineto`, `rlineto`, `curveto`, `rcurveto`, `moveto`, `rmoveto`, `closepath`. The function also supports ‘single-letter’ syntax used in standard SVG path commands:  
`L`, `l`, `C`, `c`, `M`, `m`, `Z` and `z`. The new command is also backward-compatible with the original `make-connected-path-stencil` function. Also see `scm/stencil.scm`.
- Context properties named in the ‘`alternativeRestores`’ property are restored to their value at the start of the *first* alternative in all subsequent alternatives.

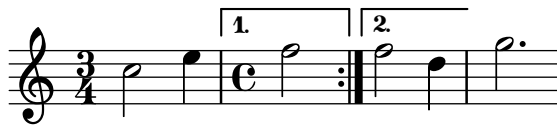
Currently the default set restores ‘current meter’;

```
\time 3/4
\repeat volta 2 { c2 e4 | }
\alternative {
  { \time 4/4 f2 d | }
  { f2 d4 | }
}
g2. |
```



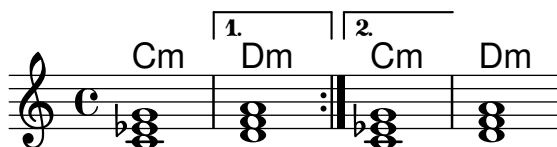
‘measure position’;

```
\time 3/4
\repeat volta 2 { c2 e4 | }
\alternative {
  { \time 4/4
    \set Timing.measurePosition = #(ly:make-moment -1/2)
    f2 | }
  { f2 d4 | }
}
g2. |
```



and ‘chord changes’;

```
<<
\new ChordNames {
  \set chordChanges = ##t
  \chordmode { c1:m d:m c:m d:m }
}
\new Staff {
  \repeat volta 2 { \chordmode { c1:m } }
  \alternative {
    { \chordmode { d:m } }
    { \chordmode { c:m } }
  }
}
\chordmode { d:m }
}
>>
```



- Improved MIDI output for breathe marks. After tied notes, breaths take time *only* from the last note of the tie; e.g. { c4~ c8 \breathe } performs as { c4~ c16 r } instead of { c4 r8 }. This is more consistent with articulations and how humans interpret breaths after ties. It now also makes it easier to align simultaneous breathe marks over multiple parts, all with different note lengths.
- A new note head style for Tabulature has been added; `TabNoteHead.style = #'slash`.
- Four new Clef glyphs have been added *Double G*, *Tenor G*, *Varpercussion* and *varC* and their related tessitura.

```
\override Staff.Clef.full-size-change = ##t

\clef "GG" c c c c
\clef "tenorG" c c c c
\clef "varC" c c c c
```

```

\clef "altovarC" c c c c
\clef "tenorvarC" c c c c
\clef "baritonevarC" c c c c
\clef "varpercussion" c c c c

\break
\override Staff.Clef.full-size-change = ##f

\clef "GG" c c c c
\clef "tenorG" c c c c
\clef "varC" c c c c
\clef "altovarC" c c c c
\clef "tenorvarC" c c c c
\clef "baritonevarC" c c c c
\clef "varpercussion" c c c c

```

The image displays a musical score with four staves. The first staff uses a C-clef (soprano), the second a C-clef (alto), the third a C-clef (tenor), and the fourth a C-clef (bass). Each staff contains a sequence of notes with varying durations, including quarter, eighth, and sixteenth notes, as well as rests. The notes are represented by black dots on the staff lines.

- Isolated durations in music sequences now stand for unpitched notes. This may be useful for specifying rhythms to music or scheme functions. When encountered in the final score, the pitches are provided by the preceding note or chord. Here are two examples where this makes for readable input:

```

\new DrumStaff \with { \override StaffSymbol.line-count = 1 }
\drummode {
  \time 3/4
  tambourine 8 \tuplet 3/2 { 16 16 16 }
              8 \tuplet 3/2 { 16 16 16 } 8 8 |
}

```

The image shows a musical score for a drum staff in 3/4 time. It features a series of notes and rests, with some notes grouped in triplets. The notes are represented by black dots on the staff lines.

```

\new Staff { r16 c'16 ~ 8 ~ 4 ~ 2 | }

```

The image shows a musical score for a staff in C major. It features a series of notes and rests, with some notes grouped in triplets. The notes are represented by black dots on the staff lines.

- `\displayLilyMusic` and its underlying Scheme functions no longer omit redundant note durations. This makes it easier to reliably recognize and format standalone durations in expressions like

```
{ c4 d4 8 }
```

- Beaming exceptions can now be constructed using the `\beamExceptions` scheme function. One can now write

```
\time #'(2 1) 3/16
\set Timing.beamExceptions =
  \beamExceptions { 32[ 32] 32[ 32] 32[ 32] }
c16 c c |
\repeat unfold 6 { c32 } |
```



with multiple exceptions separated with `|` bar checks (writing the exception pattern without pitches is convenient but not mandatory). Previously, setting the beam exceptions would have required writing

```
\set Timing.beamExceptions =
#'(
  (end . ;start of alist
    ( ;entry for end of beams
      ((1 . 32) . (2 2 2)) ;start of alist of end points
    )) ;rule for 1/32 beams -- end each 1/16
)
```

- The most common articulations are now reflected in MIDI output. Accent and marcato make notes louder; staccato, staccatissimo and portato make them shorter. Breath marks shorten the previous note.

This behavior is customizable through the `midiLength` and `midiExtraVelocity` properties on `ArticulationEvent`. See `script-init.ly` for examples.

- The PostScript functionality of stroke adjustment is no longer applied automatically but left to the discretion of the PostScript device (by default, Ghostscript uses it for resolutions up to 150dpi when generating raster images). When it is enabled, a more complex drawing algorithm designed to benefit from stroke adjustment is employed mostly for stems and bar lines.

Stroke adjustment can be forced by specifying the command line option `'-dstrokeadjust'` to LilyPond. When generating PDF files, this will usually result in markedly better looking PDF previews but significantly larger file size. Print quality at high resolutions will be unaffected.