

LilyPond

Das Notensatzprogramm

Neuerungen

Das LilyPond-Entwicklerteam

Dieses Dokument listet die Änderungen und neuen Funktionen in LilyPond für die Version 2.24.4 seit 2.22.

Zu mehr Information, wie dieses Handbuch unter den anderen Handbüchern positioniert, oder um dieses Handbuch in einem anderen Format zu lesen, besuchen Sie bitte Abschnitt “Manuals” in *Allgemeine Information*.

Wenn Ihnen Handbücher fehlen, finden Sie die gesamte Dokumentation unter <https://lilypond.org/>.

This document has been placed in the public domain.

Für LilyPond Version 2.24.4

Achtung: LilyPond-Veröffentlichungen können Änderungen der Syntax enthalten, die Modifikationen existierender Dateien, die für ältere Versionen geschrieben wurden, erfordern können, um in der neuen Version zu funktionieren. Um Dateien zu aktualisieren ist es **nachdrücklich empfohlen**, das Werkzeug `convert-ly` zu verwenden, welches mit LilyPond verteilt wird und in Abschnitt “Dateien mit `convert-ly` aktualisieren” in *Anwendungsbenutzung* beschrieben ist. `convert-ly` kann fast alle Änderungen der Syntax automatisch aktualisieren. Nutzer von Frescobaldi können `convert-ly` direct aus Frescobaldi aufrufen, indem sie „Werkzeuge > Mittels `convert-ly` aktualisieren...” verwenden. Andere Bearbeitungsprogramm mit Unterstützung für LilyPond können ebenfalls die Möglichkeit bieten, `convert-ly` über die graphische Oberfläche aufzurufen.

Bedeutende Änderungen in LilyPond

- LilyPond erfordert jetzt Guile 2.2. Selbst wenn Sie nicht selbst Code in Scheme schreiben, nutzen Sie vielleicht Bibliotheken, die eine signifikante Menge von Anpassungen enthält. Falls diese nicht mit LilyPond 2.24.4 funktionieren, melden Sie dies bitte an die Entwickler der Bibliothek. Falls Sie Entwickler einer Bibliothek sind, siehe [Hinweise zu Guile 2.2], Seite 22, am Ende dieses Dokuments.
- Die Infrastruktur zum Erstellen der offiziellen Pakete wurde komplett überarbeitet, zeitgleich mit dem Wechsel auf Guile 2.2. Ab dieser Veröffentlichung stellen wir 64-bit Programme für macOS und Windows bereit. Diese Pakete sind außerdem einfache Archive, die zur „Installation“ an einen beliebigen Ort entpackt werden können. Zum Deinstallieren entfernen Sie einfach das Verzeichnis. Wir haben außerdem das begrenzte Bearbeitungsprogramm aufgegeben, das auf macOS und Windows installiert wurde, LilyPad, und empfehlen stattdessen den Wechsel auf eine externe Lösung, wie zum Beispiel der beliebte Editor Frescobaldi (<https://frescobaldi.org>), oder eine andere Umgebung wie beschrieben in Abschnitt „Leichteres Editieren“ in *Allgemeine Information*. Für weitere Informationen beachten Sie bitte die ausführlichen Anweisungen in Abschnitt „Installing“ in *Handbuch zum Lernen*.

Hinweise zur Kompilierung des Quelltexts und für Paketersteller

Dieser Abschnitt richtet sich an Enthusiasten, die LilyPond aus dem Quelltext übersetzen, und Paketersteller, die LilyPond für Distributionen vorbereiten. Falls Sie nicht zu einer dieser Gruppen gehören, können Sie diesen Abschnitt überspringen.

- Wie bereits angemerkt benötigt LilyPond jetzt Guile 2.2. Falls aus anderen Gründen in der Distribution erforderlich, kann LilyPond auch mit Guile 3.0 übersetzt werden, durch Angabe von `GUILE_FLAVOR=guile-3.0` an das `configure` Skript. Dies ist jedoch zum aktuellen Zeitpunkt weder empfohlen noch offiziell unterstützt.
- Die Auswertung von Scheme Code in Guile 2.2 ist langsamer als in Guile 1.x. Um den Großteil der Leistungseinbußen auszugleichen, empfehlen wir die Übersetzung der `.scm` Dateien als Bytecode. Dies geschieht zuerst mit der Ausführung von `make bytecode` während der Übersetzung und anschließend `make install-bytecode` zusätzlich zu `make install`.
- Beginnend mit dieser stabilen Versionen installiert LilyPonds Build-System keine Textschrifarten mehr. Bitte installieren Sie diese als separate Pakete unter Beachtung der Lizenzen und und Notice-Dateien.

Neuerungen in der musikalischen Notation

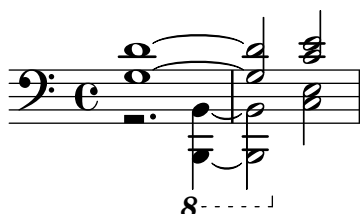
Verbesserungen bei der Anzeige von Tonhöhen

- Die Unterstützung für alternative Versetzungszeichen wurde verbessert. Durch die Eigenschaft `alterationGlyphs` in Kontexten wie `Staff` können Versetzungszeichen für alle Grobs auf einmal gesetzt werden (siehe auch Abschnitt “Alternate accidental glyphs” in *Notation-sreferenz*).



- Oktavierungsklammern können sich auf eine einzelne Stimme beziehen statt auf das gesamte Notensystem. Dies erforderte in der Vergangenheit einige Verrenkungen.

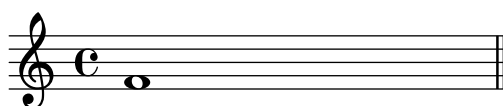
```
\layout {
  \context {
    \Staff
    \remove Ottava_spanner_engraver
  }
  \context {
    \Voice
    \consists Ottava_spanner_engraver
  }
}
```

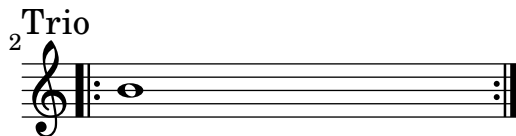


Verbesserungen bei Rhythmen

- Der neue Befehl `\section` fügt einen doppelten Taktstrich ein, der einwandfrei mit Wiederholungszeichen zusammenarbeitet. Ein Abschnitt kann mit dem neuen Befehl `\sectionLabel` benannt werden.

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





- `\numericTimeSignature` und `\defaultTimeSignature` beziehen sich jetzt auf alle Notensysteme auf einmal (genauer gesagt auf alle Notensysteme im gleichen Timing Kontext). Dies entspricht dem Verhalten von `\time`.
- Der Textbeschriftungsbefehl `\rhythm` wurde hinzugefügt. Er bietet einen einfachen Weg, um Rhythmen gemischt mit Text einzugeben, wie beispielsweise in „Swing“-Angaben.

```
\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
}
```



- Der Befehl `\enablePolymeter` bietet jetzt eine Abkürzung, um Engraver zu verschieben, wie es für die gleichzeitige Verwendung von unterschiedlichen Zeitsignaturen nötig ist. Der Code:

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

kann damit verkürzt werden zu:

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

Unabhängig davon wurde der `Default_bar_line_engraver` entfernt.

- Die neue Option `visible-over-note-heads` kann benutzt werden, um N-tolenklammern immer sichtbar zu machen, wenn deren Richtung über den Notenköpfen gesetzt ist. Die Option kann mit der üblichen Sichtbarkeit von N-tolenklammern genutzt werden oder mit `#'if-no-beam`.



- Taktzählungen beachten jetzt komprimierte mehrtaktige Pausen und Alternativen.



- Taktnummern können in ihrem Takt zentriert werden, wie es in Filmmusiken üblich ist.

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

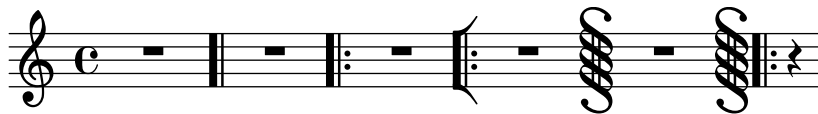
<<
{ \repeat unfold 3 { c'4 d' e' f' } }
{ \repeat unfold 3 { c'4 d' e' f' } }
>>
```



- Taktnummern in der Mitte oder am Ende eines Systems werden jetzt an ihrer linken Seite ausgerichtet. Dies folgt der Empfehlung von Elaine Gould (*Behind Bars*, S. 237), und war größtenteils der gefundene Konsens in einer Diskussion zwischen den Entwicklern. Die Ausrichtung der Taktnummern am Beginn eines Systems bleiben unverändert.
- `\bar ", "` erzeugt einen kurzen Taktstrich.



- Die folgenden vordefinierten Taktarten erscheinen nicht mehr als einfache Taktstriche am Ende einer Zeile. Annotierte Taktarten (zum Beispiel `\bar "S-|"`) wurden für diesen Zweck hinzugefügt.



- `\bar ""` ist nicht mehr nötig, um die erste Taktnummer darzustellen. Es reicht jetzt aus, `barNumberVisibility` auf `all-bar-numbers-visible` zu setzen oder eine der anderen Einstellungen, in der die erste Taktnummer sichtbar ist.

Es ist zu beachten, dass dies eine Änderung im Verhalten für Dokumente ist, die `barNumberVisibility` auf `all-bar-numbers-visible` oder ähnliches setzen und `BarNumber.break-visibility` auf `#t`, ohne ein `\bar ""`. Ab dieser Version wird dann eine Taktnummer am Anfang ausgegeben. Dies ist das erwartete Verhalten (*alle* Taktnummern sollen sichtbar sein), aber wegen leicht unklarer Dokumentation könnte diese Einstellung genutzt worden sein, um Taktnummern in der Mitte von Systemen außer der ersten Taktnummer darzustellen. In diesem Fall reicht es aus, `\set Score.barNumberVisibility = #all-bar-numbers-visible` zu entfernen, da `\override BarNumber.break-visibility = ##t` die relevanten Einstellungen alleine vornimmt.

- Der Befehl `\break` fügt jetzt immer einen Umbruch ein, ohne Rücksicht auf die standardmäßigen Entscheidungen zu Umbrüchen. Es ist zum Beispiel nicht mehr nötig, ein `\bar ""` einzufügen, um einen Umbruch in der Mitte eines Taktes zu erreichen.

Der neue Befehl `\allowBreak` fügt einen möglichen Umbruch ein, ohne ihn zu erzwingen, aber ebenfalls wie `\break` ohne Rücksicht auf die standardmäßigen Entscheidungen zu Umbrüchen.

- Der Taktstrich "-" wurde entfernt. `convert-ly` wandelt ihn in "" um. Dies führt zu geringfügigen Änderungen bei den horizontalen Abständen bei Zeilenumbrüchen.
- `automaticBars` wurde entfernt. `convert-ly` wandelt `automaticBars = ##f` in `measureBarType = #'()` um.
- `\defineBarLine` akzeptiert jetzt `#t` anstelle der Wiederholung des Zeichennamens in der Mitte einer Zeile.
- `Bar_engraver` hat früher Zeilenumbrüche zwischen Taktstrichen in allen Fällen verboten, aber tut dies jetzt nur noch, wenn die Kontexteigenschaft `forbidBreakBetweenBarLines` auf `#t` gesetzt ist, was der Standard ist. Die Eigenschaft `barAlways`, die in der Vergangenheit das Fehlen von `forbidBreakBetweenBarLines` umging, wurde entfernt.
- Wegen Änderungen in den Interna von `\bar` wird die Verwendung des Befehls nicht länger unterstützt, bevor die tieferen Kontexte mit `\new` erzeugt wurden. Diese Anwendungen werden jetzt ein zusätzliches Notensystem erzeugen. Diese ist ähnlich zu dem Fall, der mit Befehlen wie `\override Staff...` auftritt (siehe Abschnitt "An extra staff appears" in *Anwendungsbenutzung*).

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

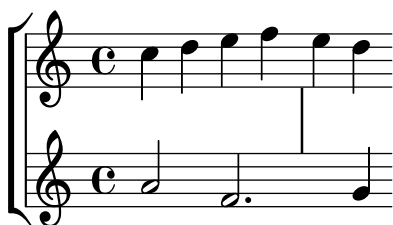


Die Lösung ist, `\bar` innerhalb der Musik für jedes Notensystem zu platzieren, wie üblich mit den meisten Befehlen.

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



- Die Taktart `"-span|"` erzeugt einen *mensurstrich*.



- Staff-Kontexte verwenden den neuen `Caesura_engraver`, um den Befehl `\caesura` zu notieren.

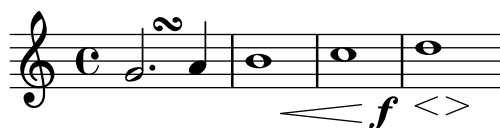


Verbesserungen bei Ausdrucksbezeichnungen

- Ereignisse, die mit Noten verbunden sind (zum Beispiel Dynamiken oder Artikulationen), können mit `\after` um eine beliebige Dauer verzögert werden. Das vereinfacht viele Situationen, die vorher die Verwendung von expliziter Polyphonie und leeren Pausen erforderten.

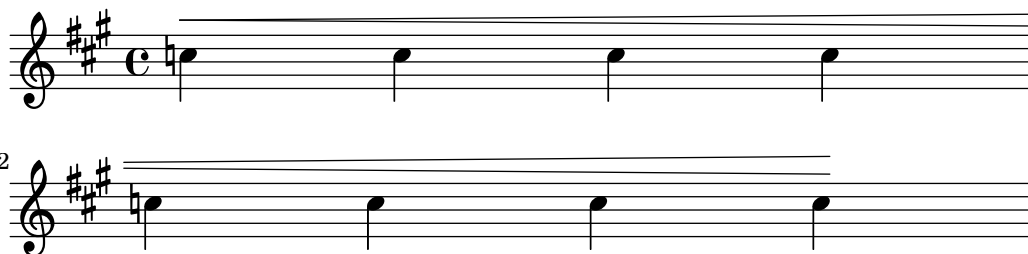
```
{
  \after 2 \turn g'2. a'4
  \after 2 \< b'1
  \after 2. \f c''
  <>\< \after 4 \> \after 2\! d''
```


}



- Umgebrochene Klammern haben jetzt standardmäßig einen linken Abstand. Das folgt dem Vorbild von veröffentlichten Notensätzen und behebt einige Fälle, in denen umgebrochene Klammern vertikal durch die Tonart verschoben wurden.

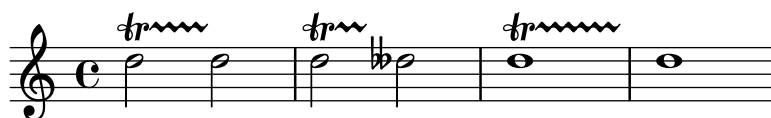
```
\relative {
  \key a \major
  c''4^\< c c c \break c c c c\! |
}
```



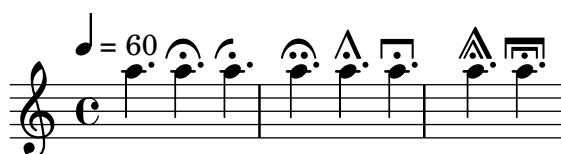
- Die Enden von Klammern können jetzt links, mittig oder rechts einer NoteColumn ausgerichtet werden, indem die Eigenschaft `endpoint-alignments` überschrieben wird.



- Die Richtung eines Trillers kann jetzt mit Richtungsanweisungen wie bei anderen Artikulationen gesetzt werden, also mit `_\startTrillSpan` oder `^\startTrillSpan`.
- Das Aussehen von Trillern wurde geändert, um besser dem klassischen Notensatz zu entsprechen. Sie enden jetzt vor der nächsten Note und nicht über ihr. Falls die nächste Note ein Vorzeichen hat, endet der Triller davor. Falls die nächste Note die erste Note eines Takts ist, endet der Triller stattdessen vor dem Taktstrich.



- Der Abstand von Fermaten ist jetzt größer. Dies verhindert einige Fälle, in denen Fermaten zu nah an Punkten und anderen Objekten platziert wurde.



- Das Flageolet-Symbol ist jetzt kleiner und etwas dicker. Das folgt dem Vorbild von veröffentlichten Notensätzen und macht die empfohlene Anpassung unnötig, um das Zeichen zu verkleinern (`\tweak font-size -3 \flageolet`).



- Das Akzentzeichen ist jetzt ein wenig kleiner. Das behebt einige Fälle, in denen ein Auflösungszeichen Akzente vertikal verschob.



- Das Kommazeichen, wie benutzt im Befehl `\breathe`, wurde zu einer üblicheren Form geändert.



Das alte Zeichen bleibt verfügbar unter dem Namen `,raltcomma`:

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



- Die neue Kontexteigenschaft `breathMarkType` wählt aus mehreren vordefinierten Typen das Symbol, das `\breathe` produziert.

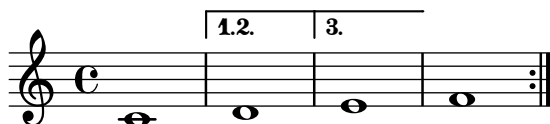
```
\fixed c' {
  \set breathMarkType = #'tickmark
  c2 \breathe d2
}
```



Verbesserungen bei Wiederholungen

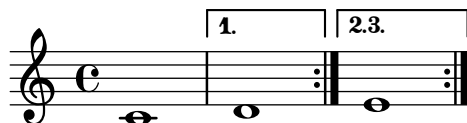
- Wiederholungsalternativen können innerhalb von wiederholten Abschnitten notiert werden.

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



- Die Zahlen in Wiederholungsklammern können mit dem Befehl `\volta` gesetzt werden.

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



- Der neue Befehl `\repeat segno` notiert automatisch eine Vielfalt von Formen mit *da-capo* und *dal-segno*.

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

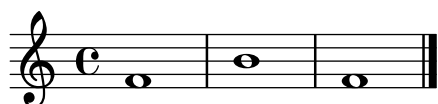
\score { \music }
\score { \unfoldRepeats \music }
```



- Der neue Befehl `\fine` fügt einen endgültigen Taktstrich ein, der einwandfrei mit Wiederholungszeichen zusammenarbeitet. Innerhalb eines `\repeat` gibt er außerdem *Fine* aus und beendet die Musik beim Entfalten.

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

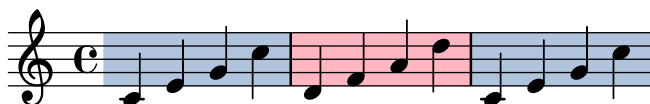
\score { \music }
\score { \unfoldRepeats \music }
```



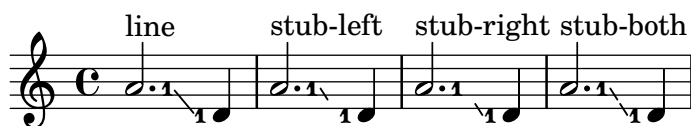
- Der Befehl `\volta` entfernt Musik, wenn eine Wiederholung entfaltet wird.
- Der Befehl `\unfolded` fügt Musik hinzu, wenn eine Wiederholung entfaltet wird.

Verbesserungen bei Anmerkungen

- Die neuen Befehle `\staffHighlight` und `\stopStaffHighlight` können benutzt werden, um einen musikalischen Abschnitt einzufärben.

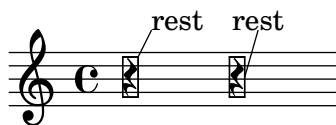


- Ein neuer Grob `FingerGlideSpanner` ist jetzt verfügbar, um das Rutschen von Fingern auf einer Saite von einer Position zu einer anderen anzuzeigen. Mehrere Aussehen sind möglich, abhängig von der Einstellung `style`. Im Bild sind die Stile `line`, `stub-left`, `stub-right` und `stub-both` zu sehen.

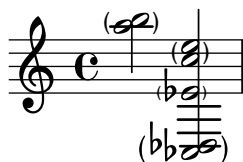


Außerdem möglich sind `dashed-line`, `dotted-line`, `zigzag`, `trill`, `bow` und `none`.

- Blasen haben jetzt änderbare Formattierungen.



- Einklammern von Akkorden wird jetzt unterstützt. Momentan muss die Schriftgröße der Klammern manuell angepasst werden.



- Einklammern von Streckern wird jetzt unterstützt.



- Eine „Zeit-basierte“ Version des Befehls `\parenthesize` wurde hinzugefügt. Der Befehl erwartet einen Pfad: `\parenthesize GrobName` oder `\parenthesize ContextName.GrobName`. Er verhält sich wie ein `\once \override`. Das Interface ergänzt die bereits vorhandene Form `\parenthesize event`, ähnlich zu `\footnote`.

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



- Das Hinzufügen von `Melody_engraver` zum `Voice`-Kontext funktioniert jetzt standardmäßig, um die Richtung des Halses der mittleren Note entsprechend der Melodie zu ändern. Vorher erforderte dies zusätzlichen Anpassungen von `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 |
}
```



Die Kontexteigenschaft `suspendMelodyDecisions` kann verwendet werden, um dieses Verhalten temporär auszuschalten, wie es auch `\override Stem.neutral-direction = #DOWN` vorher tat.

- Der neue `Mark_tracking_translator` übernimmt die Entscheidung von `Mark_engraver`, wann eine Markierung erzeugt wird. `Mark_engraver` kontrolliert weiterhin die Formatierung und die vertikale Platzierung.

Standardmäßig erzeugen `Mark_engraver` in mehreren Kontexten eine gemeinsame Abfolge von Markierungen. Falls unabhängige Sequenzen gewünscht werden, müssen mehrere `Mark_tracking_translator` verwendet werden.

Verbesserungen bei der Textformattierung

- Die neuen Befehle `\textMark` und `\textEndMark` sind verfügbar, um einen beliebiges Stück Text zwischen Noten einzufügen, genannt eine Textmarkierung. Diese Befehle verbessern die bisher verfügbare Syntax mit dem Befehl `\mark`, aufgerufen als `\mark markup (\mark "..."` oder `\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` und `\textEndMark` sind jetzt der empfohlene Weg, um Textmarkierungen zu erstellen. Die Benutzung von `\mark` zu diesem Zweck wird zwar weiter unterstützt, es wird aber davon abgeraten (das betrifft nicht den Befehl `\mark` selbst, nur der Aufruf mit einem Textargument; `\mark \default` oder `\mark number` sind weiterhin der empfohlene und einzige Weg, um eine Probenmarkierung zu erzeugen).

Die neuen Befehle haben einige Unterschiede zu `\mark markup`. Zu jedem Zeitpunkt kann es eine beliebige Anzahl geben, während es nur eine Benutzung von `\mark` geben kann. Sie erzeugen Grobs des dedizierten Typs `TextMark`, wohingegen `\mark` ein Grob des Typs `RehearsalMark` erzeugt, unabhängig davon ob es als Probenmarkierung oder Textmarkierung aufgerufen wurde; die Einführung dieser Unterscheidung erlaubt es Stylesheets, verschiedene Layouteinstellungen für Proben- und Textmarkierungen zu setzen. Die Ausrichtung der neuen Befehle ist unterschiedlich: `\textMark` erzeugt immer eine links-ausgerichtete Markierung, während `\textEndMark` eine rechts-ausgerichtete Markierung erzeugt; im Gegensatz dazu hängt die Ausrichtung eines `RehearsalMark` vom Ankerpunkt des Objekts ab, an dem aus ausgerichtet ist.

Siehe Abschnitt “Text marks” in *Notationsreferenz* für Details.

- Textvarianten für Kreuze, Bs, Auflösungszeichen, Doppelkreuze und Doppel-Bs sind jetzt in der Emmentaler-Schrift verfügbar. In Texten können sie einfach über ihre Unicode-Zeichen genutzt werden.

1 # 2 ♭ 3 ♮ 4 ♯ 5 × 6

- Es ist jetzt möglich, die Breite und die Form von (einigen) Emmentaler-Ziffern mit OpenType-Features zu kontrollieren.

0123456789 147 147 (time signatures)

0123456789 147 147 (alternatives)

0123456789 147 147 (fixed-width)

0123456789 147 147 (figured bass)

0123456789 147 147 (fingering)

- `\smallCaps` funktioniert jetzt für beliebige Textbeschriftungen, nicht nur für Zeichenketten.
- Die Syntax für Bedingungen in Textbeschriftungen wurde flexibler und nutzerfreundlicher gemacht. Es benutzt die neuen Textbeschriftungsbefehle `\if` und `\unless`. Hier sind beispielhaft einige Ersetzungen:

Sytanx für 2.22

```
\on-the-fly #first-page ...
\on-the-fly #not-part-first-page ...
\on-the-fly #(on-page n) ...
```

Syntax für 2.24

```
\if \on-first-page ...
\unless \on-first-page-of-part ...
\if \on-page #n ...
```

- Mit dem neuen Textbeschriftungslistenbefehle `string-lines` ist es jetzt möglich, eine Zeichenkette an einem gegebenen Zeichen aufzuteilen. Standardmäßig teilt der Befehl an einem Zeilenumbruch. Umgebende Leerzeichen werden entfernt. Die resultierende Liste von Textbeschriftungen kann weiter formatiert werden. Dies ist ein sehr praktischer Weg, um zusätzliche Strophen für Lieder einzugeben.

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!

- Der neue Textbeschriftungsbefehl `\align-on-other` übersetzt eine Textbeschriftung so, als wäre sie an einer anderen Textbeschriftung ausgerichtet.

1
12
12345
123

- Zwei neue Textbeschriftungsfunktionen `\with-dimension` und `\with-dimension-from` sind verfügbar. Sie sind ähnlich zu `\with-dimensions` und `\with-dimensions-from`, ändern aber nur eine Dimension (statt beider).
- Die neuen Textbeschriftungsfunktionen `\with-true-dimension` und `\with-true-dimensions` sind verfügbar. Sie geben der Textbeschriftung die tatsächliche Größe ihrer gedruckten Tinte, die von der standardmäßigen Größe für einige Symbole aufgrund von Einschränkungen der Textregularität abweichen kann.



- Textersetzungen können jetzt Zeichenketten mit beliebigem Textbeschriftungen ersetzen, nicht nur mit einer Zeichenkette.

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

2nd time

- Ein neuer Textbeschriftungsbefehl `\with-string-transformer` ist verfügbar. Er interpretiert eine Textbeschriftung mit einem installierten „string transformer“; dieser wird aufgerufen, wenn die Interpretation der Textbeschriftung die Interpretation einer Zeichenkette erfordert, und erlaubt es, Änderungen an dieser Zeichenkette vorzunehmen, wie das Ändern der Großschreibung.
- Die Funktion `markup->string` konvertiert eine Textbeschriftung in eine ungefähre Darstellung als Zeichenkette; sie wird für die Ausgabe von PDF-Metadaten sowie für Liedtexte und Markierungen in MIDI benutzt. Textbeschriftungsbefehle können jetzt eine spezielle Methode für die Konvertierung von Textbeschriftungen, die mit ihnen erzeugt wurden, in Zeichenketten definieren, für die Benutzung mit `markup->string`. Zum Beispiel:

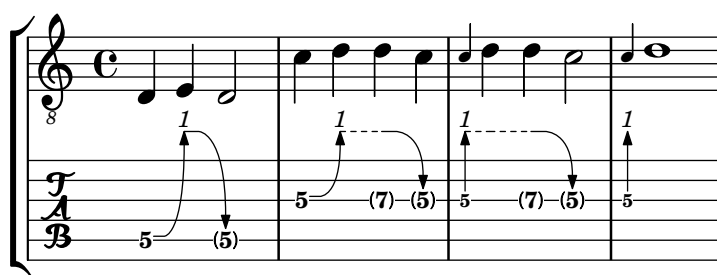
```
#(define-markup-command (upcase layout props arg) (string?)
```

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


Neuerungen für spezielle Notation

Verbesserungen für Saiteninstrumente mit Bünden

- Die neuen Saiteneinstellungen banjo-double-c und banjo-double-d wurden hinzugefügt.
- Ein neuer Grob BendSpanner ist jetzt für TabStaff verfügbar, um eine gekurvte Saite anzuzeigen. Abgesehen vom Standard sind drei Stile möglich: 'hold', 'pre-bend' und 'pre-bend-hold'.



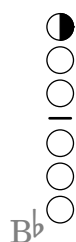
Neuerungen für Schlagzeug

- Der Notationsstil weinberg-drums-style für Schlaginstrumente wurde hinzugefügt. Er basiert auf der Standardisierungsarbeit von Norman Weinberg.

Verbesserungen für Blasinstrumente

- Zusätzliche Darstellungsdetails für Holzbläserdiagramme können jetzt angegeben werden, inklusive der Winkel von teilweise bedeckten Löchern und der Anzeige von nicht-graphischen Trillerklappen.

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



Verbesserungen für die Notation von Akkorden

- Unterstützung von Akkordgittern wurde hinzugefügt.

- KievanStaff, MensuralStaff, PetrucciStaff, und VaticanaStaff erlauben jetzt Zeilenumbrüche an jeder Position und erzeugen nicht länger leere Taktstriche ("").
- Im GregorianTranscriptionStaff werden Divisiones jetzt standardmäßig mit BarLine Grobs notiert. Um sie zu Divisio Grobs zu ändern, kann \EnableGregorianDivisiones genutzt werden.
- GregorianTranscriptionStaff erlaubt jetzt einen Zeilenumbruch nach einer Note und nutzt nicht länger Time_signature_engraver.
- GregorianTranscriptionVoice nutzt nicht länger Stem_engraver.

Verbesserungen für Weltmusik

- Unterstützung für klassische persische Musik ist jetzt verfügbar. Dafür wurden zwei neue Versetzungszeichen, *sori* und *koron*, zu LilyPond hinzugefügt.

```
\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 |
}
```



Verschiedene Verbesserungen

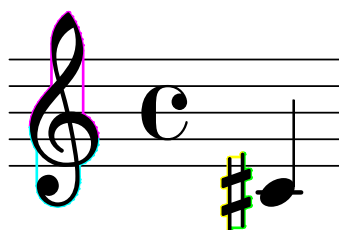
- In der Emmentalerschriftart wurden identisch-aussehende Notenköpfe, die sich nur in der Ausrichtung von Notenhälsen unterschieden, in ein einzelnes Symbol zusammengefasst. Zum Beispiel wurden die Zeichen `noteheads.u2triangle` und `noteheads.d2triangle` durch ein einzelnes Zeichen ersetzt, `noteheads.s2triangle`. Paare von Notenköpfen, die abhängig von der Ausrichtung anders aussehen, bleiben verschieden.

Zusätzlich gibt die Eigenschaft `stem-attachment` von `NoteHead` Grobs jetzt den wirklichen, ausrichtungsabhängigen Punkt für das Anbringen von Notenhälsen zurück statt eines hypothetischen Punktes für einen aufwärtsgerichteten Notenhals.

- Zwei überflüssige Zeichen in der Emmentalerschriftart wurden entfernt: `scripts.trillelement` (stattdessen ist `scripts.trill_element` zu benutzen) und `scripts.augmentum` (stattdessen ist `dots.dotvaticana` zu benutzen).
- Mit `\paper { bookpart-level-page-numbering = ##t }` ist es jetzt möglich, Buchteile unabhängig im Bezug auf die Seitennummerierung zu machen. Wenn dies für alle Buchteile genutzt wird, hat jeder Buchteil eine eigene Nummerierung, jeweils beginnend bei 1. Die Option kann auch für einen einzelnen Buchteil genutzt werden, was nützlich sein kann, um die Seiten einer analytischen Einführung unabhängig und mit römischen Ziffern zu nummerieren (letztes kann durch Benutzen von `page-number-type = #'roman-lower` erreicht werden).
- Eine neue Funktion `break-alignment-list` ist jetzt verfügbar, um unterschiedliche Werte abhängig von der Umbruchrichtung eines Grobs zurückzugeben. Dies kann beispielsweise genutzt werden, um verschiedene Ausrichtungen eines Grobs zu setzen, abhängig davon, ob er am Anfang, in der Mitte oder am Ende einer Zeile positioniert wird.

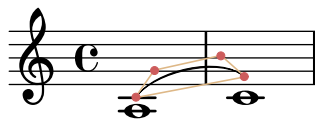


- Der neue `Mark_performer` erzeugt MIDI Markierungen in der gleichen Weise, wie `Mark_engraver` gedruckte Markierungen erzeugt.
- Eigenschaften von `PaperColumn` und `NonMusicalPaperColumn` (wie `NonMusicalPaperColumn.line-break-system-details`) können jetzt mitten in der Musik mit dem üblichen Befehl `\once \override` überschrieben werden. Vorher war dies ein Sonderfall und erforderte den Befehl `\overrideProperty`.
- Die neuen Eigenschaften `show-horizontal-skylines` und `show-vertical-skylines` erlauben das Anzeigen der Skyline eines Objekts. Dies ist flexibler als die bereits existierende Option `debug-skylines`, weil es für alle Grobs funktioniert. Obwohl die Eigenschaften primär für die Entwicklung von LilyPond gedacht sind, können sie nützlich sein, um Abstandsentscheidungen zu verstehen oder Stencils in Scheme zu überschreiben.




- Der neue Befehl `\vshape` verhält sich wie `\shape`, zeigt aber auch die Kontrollpunkte und Polygone für einfachere Anpassungen.

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



- `\markup \path` funktioniert jetzt auch in der Ausgabe von SVGs, selbst wenn der Pfad nicht mit einem der Befehle `moveto` oder `rmoveto` beginnt. Der Pfad akzeptiert außerdem die Buchstabenäquivalente für SVG (`moveto` = M, etc.).
- `set-default-paper-size` und `set-paper-size` akzeptieren jetzt eine benutzerdefinierte Papiergröße.

```
#(set-default-paper-size '(cons (* 100 mm) (* 50 mm)))
```
- `lilypond-book` unterstützt zwei neue Optionen für Musikfragmente, `paper-width` und `paper-height`, um die Papiergröße zu setzen.
- `lilypond-book` unterstützt eine neue Schnipsel-Option `inline` für die Anzeige von Musik, wie  innerhalb eines Textabsatzes.
- Das Skript `lilypond-book` erlaubt jetzt Klammern in den Argumenten für die Befehle `\lilypond` (für LaTeX) und `@lilypond` (für Texinfo).
- `lilypond-book` fügt jetzt das aktuelle Verzeichnis als letzten Eintrag für die Suche von eingebundenen Dateien an, statt es als ersten Eintrag vor alle angegebenen Pfade zu stellen. Dies erlaubt es eingebundenen Verzeichnissen, Dateien im aktuellen Verzeichnis zu überschreiben, und wird nur bemerkt werden, falls es Dateien gleichen Namens in beiden gibt.
- Die neue Scheme-Funktion `universal-color` beschreibt eine Farbpalette mit acht Elementen, die entworfen wurde, um für Personen mit Dichromatismus eindeutig zu sein.

black

orange

skyblue

bluegreen

yellow

blue

vermillion

redpurple

- Die Option `-dembed-source-code` bettet jetzt auch Bilder, die mit `\epsfile` hinzugefügt wurden, und Dateien, die mit `\verbatim-file` eingebunden wurden, ein.
- Der Standard der Programmoption `aux-files` wurde zu `#f` geändert. Falls LilyPond mit dem Argument `-dbackend=eps` aufgerufen wird und die Hilfsdateien mit den Endungen `.tex` und `.texi` benötigt werden, muss jetzt auch `-daux-files` angegeben werden. Die Formate der Bilder für `lilypond-book` kann separat für große Seitenbilder (typischerweise PNG für die Ausgabe als HTML) und für kleine Bilder von Systemen (typischerweise EPS oder PDF für Druckausgaben) gesetzt werden, mit den Optionen `-dtall-page-formats` und `-dseparate-page-formats`.
- Die Einheit ‚big point‘ (1bp = 1/72in) ist jetzt verfügbar durch Anfügen von `\bp` an Längenwerte.
- Scheme-definierte Translators, die sowohl in `‘\layout’` als auch `‘\midi’` nutzbar sind, können jetzt mit `make-translator` erzeugt werden. Scheme-definierte Performer, die nur in `‘\midi’` nutzbar sind, können mit `make-performer` erzeugt werden. Diese Makros funktionieren genau wie das existierende Makro `make-engraver` für die Erzeugung eines Engravers, der nur in `‘\layout’` nutzbar ist.

- Scheme Translators können jetzt einen neuen Slot definieren, genannt `pre-process-music`. Er wird für alle Translators nach allen Listenern aufgerufen, aber vor allen `process-music`. Dies kann für die Vorverarbeitung genutzt werden, die von allen Ereignissen abhängt, aber Kontexteigenschaften setzen muss, bevor diese von anderen Translators gelesen werden.
- Scheme Translators können jetzt Listener enthalten, geschrieben als

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

Diese werden niemals mehr als einmal pro Zeitschritt ausgelöst. Sie erzeugen eine Warnung, wenn sie zwei Ereignisse in einem Zeitschritt empfangen, außer wenn die beiden Ereignisse gleich sind.

- Die gleiche Definition von Grobs kann jetzt genutzt werden, um Grobs verschiedener Klassen zu erzeugen (`Item`, `Spanner`, `Paper_column` und `System`). Als Teil dieser Änderung würden die Typen `FootnoteItem` und `FootnoteSpanner` in einen einzigen Typ `Footnote` konsolidiert. Gleichfalls wurden `BalloonTextSpanner` und `BalloonTextItem` in `BalloonText` vereinigt.

Wenn die Definition keine Klasse vorschreibt, sollten die Engraver auswählen, mit welcher Klasse der Grob erstellt wird. Für Autoren von Scheme Engavern heißt das die Verwendung von entweder `ly:engraver-make-item` oder `ly:engraver-make-spanner`. Die Hilfsfunktion `ly:engraver-make-sticky` unterstützt den häufigen Fall von *sticky* Grobs, wie Fußnoten und Blasen. Es erzeugt einen Grob mit der gleichen Klasse wie ein anderer Grob und verwaltet Eltern und Grenzen.

- Die neue Option `-dcompile-scheme-code`, die auch direkt in der Eingabe an LilyPond gesetzt werden kann mit `#(ly:set-option 'compile-scheme-code)`, ermöglicht bessere Meldungen, wenn die Ausführung von Scheme-Code zu einem Fehler führt. Intern nutzt dies den Byte-Compiler von Guile statt des Interpreters.

Aufgrund einer Limitierung von Guile hat dies zur Zeit jedoch den Nachteil, dass es unmöglich ist, mehr als ein paar tausend Scheme-Ausdrücke laufen zu lassen. Es wird außerdem darauf hingewiesen, dass der Guile-Compiler einige Unterschiede zum Interpreter hat. Konstante Teile eines Quasiquotes werden beispielsweise aggressiver zu echten Konstanten und führen damit in Code wie `(let ((x 4)) (sort! `(,x 3 2 1)))` zu einem Fehler, weil der „cdr“ der Liste konstant ist und es ein Fehler in Scheme ist, eine Liste von Literalen zu verändern. (In diesem speziellen Fall könnte der Code das Problem durch die Nutzung der nicht-destruktiven Funktion `sort` vermeiden, oder jedes Mal eine neue Liste erzeugen mit `(list x 3 2 1)`).

Diese Option funktioniert zur Zeit nicht unter Windows.

Hinweise zu Guile 2.2

Diese Version von LilyPond wechselt von Guile 1.8 zu Guile 2.2. Dieser Abschnitt führt einige der häufigsten Inkompatibilitäten auf, mit denen man beim Upgrade seines Scheme-Codes zu tun hat.

Eine vollständige, detaillierte Liste der Änderungen in Guile findet sich in der NEWS Datei (<https://git.savannah.gnu.org/cgit/guile.git/tree/NEWS>) des Guile-Quellcodes.

- Die Funktion `format` erfordert jetzt einen boolschen Wert oder einen Port als erstes Argument. Dieses Argument war in Guile 1.8 optional. Um die formattierte Ausgabe als Zeichenkette zu erhalten, wie es `format` ohne dieses Argument in Guile 1.8 tat, muss `#f` als Argument übergeben werden, also `(format #f "string" arguments ...)` statt `(format "string" arguments ...)`.
- Die Regeln für interne Definitionen (also nicht auf oberster Ebene) sind strenger geworden. Definitionen sind nicht länger in verschiedenen Ausdruckskontexten erlaubt. Der folgende Code ist beispielsweise nicht länger gültig:

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

Die Lösung in diesem speziellen Beispiel ist:

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

- Zeichenketten unterstützen jetzt Unicode-Zeichen. Bisher wurde ein Unicode-Zeichen als mehrere Zeichen dargestellt, und verschiedene Funktionen waren nicht für die Unterstützung von Unicode ausgelegt.
- Einige numerische Funktionen geben jetzt in mehr Fällen exakte Ergebnisse zurück. Beispielsweise gibt `(sqrt 4)` in Guile 1.8 2.0 zurück, aber 2 (eine Ganzzahl) in Guile 2.2.