LilyPond

Das Notensatzprogramm

Handbuch zum Lernen

Das LilyPond-Entwicklerteam

Copyright © 1999–2008 bei den Autoren

The translation of the following copyright notice is provided for courtesy to non-English speakers, but only the notice in English legally counts.

Die Übersetzung der folgenden Lizenzanmerkung ist zur Orientierung für Leser, die nicht Englisch sprechen. Im rechtlichen Sinne ist aber nur die englische Version gültig.

> Es ist erlaubt, dieses Dokument unter den Bedingungen der GNU Free Documentation Lizenz (Version 1.1 oder spätere, von der Free Software Foundation publizierte Versionen, ohne Invariante Abschnitte), zu kopieren, verbreiten und/oder zu verändern. Eine Kopie der Lizenz ist im Abschnitt "GNU Free Documentation License" angefügt.
>
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections. A copy of the license is included in the section entitled "GNU Free Documentation License".

Für LilyPond Version 2.12.1

Inhaltsverzeichnis

Vorwort **1**

1 Einleitung **2**

1.1 Hintergrund 2
Notensatz 2
Automatisierter Notensatz 3
Welche Symbole? 5
Die Darstellung der Musik 6
Beispielanwendung 8
1.2 Über die Dokumentation 9
Über das Handbuch zum Lernen (LM) 9
Über das Glossar (MG) 9
Über die Notationsreferenz (NR) 9
Über die Anwendungsbenutzung (AU) 10
Über die Schnipselliste 10
Über die Referenz der Interna (IR) 11
Andere Dokumentation 11

2 Übung **12**

2.1 Erste Schritte 12
2.1.1 Eine Quelldatei übersetzen 12
2.1.2 Einfache Notation 14
2.1.3 Arbeiten an Eingabe-Dateien 18
2.1.4 Wie soll das Handbuch gelesen werden 19
2.2 Notation auf einem System 20
2.2.1 Versetzungszeichen und Tonartbezeichnung (Vorzeichen) 20
2.2.2 Bindebögen und Legatobögen 21
2.2.3 Artikulationszeichen und Lautstärke 23
2.2.4 Text hinzufügen 24
2.2.5 Automatische und manuelle Balken 24
2.2.6 Zusätzliche rhythmische Befehle 25
2.3 Mehrere Noten auf einmal 26
2.3.1 Musikalische Ausdrücke erklärt 26
2.3.2 Mehrere Notensysteme 28
2.3.3 Notensysteme gruppieren 29
2.3.4 Noten zu Akkorden verbinden 30
2.3.5 Mehrstimmigkeit in einem System 30
2.4 Lieder 31
2.4.1 Einfache Lieder setzen 31
2.4.2 Text an einer Melodie ausrichten 32
2.4.3 Text zu mehreren Systemen 36
2.5 Letzter Schliff 36
2.5.1 Stücke durch Bezeichner organisieren 36
2.5.2 Versionsnummer 38
2.5.3 Titel hinzufügen 38
2.5.4 Absolute Notenbezeichnungen 38
2.5.5 Nach der Übung 40

3 Grundbegriffe ... 41

3.1 Wie eine LilyPond-Datei funktioniert ... 41
3.1.1 Einführung in die Dateistruktur von LilyPond ... 41
3.1.2 Score ist ein (einziger) zusammengesetzter musikalischer Ausdruck ... 44
3.1.3 Musikalische Ausdrücke ineinander verschachteln ... 46
3.1.4 Über die Nicht-Schachtelung von Klammern und Bindebögen ... 47
3.2 Voice enthält Noten ... 49
3.2.1 Ich höre Stimmen ... 49
3.2.2 Stimmen explizit beginnen ... 54
3.2.3 Stimmen und Text ... 57
3.3 Kontexte und Engraver ... 64
3.3.1 Was sind Umgebungen? ... 64
3.3.2 Umgebungen erstellen ... 65
3.3.3 Was sind Engraver? ... 67
3.3.4 Kontexteigenschaften verändern ... 68
Kontexteigenschaften mit `\with` setzen ... 71
Kontexteigenschaften mit `\context` setzen ... 71
3.3.5 Engraver hinzufügen und entfernen ... 72
3.4 Erweiterung der Beispiele ... 75
3.4.1 Sopran und Cello ... 75
3.4.2 Vierstimmige SATB-Partitur ... 78
3.4.3 Eine Partitur von Grund auf erstellen ... 82

4 Die Ausgabe verändern ... 86

4.1 Grundlagen für die Optimierung ... 86
4.1.1 Grundlagen zur Optimierung ... 86
4.1.2 Objekte und Schnittstellen ... 86
4.1.3 Regeln zur Benennung von Objekten und Eigenschaften ... 87
4.1.4 Optimierungsmethoden ... 87
4.2 Die Referenz der Programminterna ... 91
4.2.1 Eigenschaften von Layoutobjekten ... 91
4.2.2 Eigenschaften, die Schnittstellen besitzen können ... 95
4.2.3 Typen von Eigenschaften ... 96
4.3 Erscheinung von Objekten ... 97
4.3.1 Sichtbarkeit und Farbe von Objekten ... 97
4.3.2 Größe von Objekten ... 102
4.3.3 Länge und Dicke von Objekten ... 105
4.4 Positionierung von Objekten ... 106
4.4.1 Automatisches Verhalten ... 106
4.4.2 within-staff (Objekte innerhalb des Notensystems) ... 107
4.4.3 Objekte außerhalb des Notensystems ... 110
4.5 Kollision von Objekten ... 116
4.5.1 Verschieben von Objekten ... 116
4.5.2 Überlappende Notation in Ordnung bringen ... 119
4.5.3 Beispiele aus dem Leben ... 124
4.6 Weitere Optimierungen ... 131
4.6.1 Andere Benutzung von Optimierungen ... 131
4.6.2 Variablen für Optimierungen einsetzen ... 134
4.6.3 Mehr Information ... 135
4.6.4 Vermeiden von Optimierungen durch langsamere Übersetzung ... 136
4.6.5 Fortgeschrittene Optimierungen mit Scheme ... 137

5 An LilyPond-Projekten arbeiten ... 139
5.1 Vorschläge, wie LilyPond-Eingabe-Dateien geschrieben werden sollen ... 139
5.1.1 Allgemeine Vorschläge ... 139
5.1.2 Das Kopieren von existierender Musik ... 140
5.1.3 Große Projekte ... 140
5.1.4 Tipparbeit sparen durch Bezeichner und Funktionen ... 141
5.1.5 Stil-Dateien ... 142
5.2 Wenn etwas nicht funktioniert ... 146
5.2.1 Alte Dateien aktualisieren ... 146
5.2.2 Fehlersuche (alles auseinandernehmen) ... 146
5.2.3 Minimalbeispiele ... 147
5.3 Partituren und Stimmen ... 147

Anhang A Vorlagen ... 150
A.1 Ein einzelnes System ... 150
A.1.1 Nur Noten ... 150
A.1.2 Noten und Text ... 150
A.1.3 Noten und Akkordbezeichnungen ... 151
A.1.4 Noten, Text und Akkordbezeichnungen ... 152
A.2 Klaviervorlagen ... 152
A.2.1 Piano Solo ... 152
A.2.2 Klavier und Gesangstimme ... 153
A.2.3 Klavier mit zentriertem Text ... 154
A.2.4 Klavier mit zentrierten Lautstärkebezeichnungen ... 155
A.3 Streichquartett ... 157
A.3.1 Streichquartett ... 157
A.3.2 Streichquartettstimmen ... 158
A.4 Vokalensemble ... 160
A.4.1 SATB-Partitur ... 161
A.4.2 SATB-Partitur und automatischer Klavierauszug ... 162
A.4.3 SATB mit zugehörigen Kontexten ... 164
A.5 Vorlagen für alte Notation ... 166
A.5.1 Transkription mensuraler Musik ... 166
A.5.2 Vorlage zur Transkription von Gregorianik ... 171
A.6 Jazz-Combo ... 172
A.7 Lilypond-book-Vorlagen ... 178
A.7.1 LaTeX ... 178
A.7.2 Texinfo ... 179
A.7.3 xelatex ... 179

Anhang B Scheme-Übung ... 181
B.1 Optimierungen mit Scheme ... 183

Anhang C GNU Free Documentation License ... 184

Anhang D LilyPond-Index ... 190

Vorwort

Es muss wohl während einer Probe des EJE (Eindhovens Jugendorchester) etwa 1995 gewesen sein, als Jan, einer der schrägen Bratschisten, Han-Wen, einem der verstimmten Hornisten, von seinem großartigen neuen Projekt erzählte, an dem er gerade arbeitete. Es sollte ein automatisiertes System für den Notensatz werden (um genauer zu sein, es war MPP, ein Präprozessor für MusiXTeX). Zufällig wollte Han-Wen gerade einige Stimmen einer Partitur ausdrucken, und so schaute er sich das Programm an und war schnell begeistert. Man entschied sich, dass MPP in eine Sackgasse führte, und nach vielem Philosophieren und hitzigem E-Mail-Austausch begann Han-Wen mit LilyPond 1996. Dieses Mal wurde Jan mitgerissen von Han-Wens neuem Projekt.

Die Entwicklung eines Computerprogramms erinnert in vielem an das Erlernen eines Musikinstrumentes. Am Anfang macht es Spaß herauszufinden, wie alles funktioniert und alles, was man noch nicht kann, stellt eine Herausforderung dar. Nach der ersten Begeisterung muss man jedoch viel üben. Tonleitern und Etüden können furchtbar langweilig sein, und wenn keine Ermunterung von anderen – Lehrern, Dirigenten oder dem Publikum – kommt, ist es oft eine große Versuchung, einfach aufzuhören. Aber man macht weiter, und langsam wird das Instrument zu einem Teil des eigenen Lebens. An manchen Tagen geht alles wie von selbst und es macht Spaß, an anderen Tagen ist es nur Arbeit, aber man macht trotzdem weiter, jeden Tag.

Die Arbeit an LilyPond kann genauso wie das Spiel eines Instruments sehr langweilig sein, und manchmal kommt es vor lauter Fehlern so vor, als stapfe man durch einen Morast. Trotzdem ist die Arbeit schon Teil unseres Lebens geworden und wir machen einfach weiter. Die wahrscheinlich wichtigste Motivation ist wohl, dass unser Programm wirklich nützlich ist. Wenn wir im Internet surfen, finden wir viele Leute, die LilyPond benutzen und damit außerordentlich beeindruckende Partituren erstellen. Das zu sehen fühlt sich auf angenehme Weise etwas unwirklich an.

Unsere Stimmung heben aber nicht nur die Benutzer unseres Programmes, sondern auch die vielen Menschen, die uns helfen, indem sie Vorschläge einbringen, auf verschiedene Art an LilyPond mitwirken oder Fehlerberichte schicken. Ihnen allen möchten wir hier Dank sagen!

Musik spielen und Musik zu Papier zu bringen ist mehr als eine nette Analogie. Zusammen zu programmieren macht viel Spaß und Menschen helfen zu können ist sehr zufriedenstellend, aber letzten Endes geht es uns darum, durch dieses Programm unsere Liebe zur Musik auszudrücken. Wir hoffen, Sie können viele schöne Partituren damit setzen!

Han-Wen und Jan

Utrecht/Eindhoven, Niederlande, Juli 2002.

1 Einleitung

Dieses Kapitel stellt dem Leser die Idee hinter LilyPond und die Dokumentation von LilyPond vor.

1.1 Hintergrund

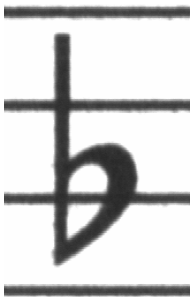
Dieser Abschnitt behandelt die allgemeinen Ziele und die Architektur von LilyPond.

Notensatz

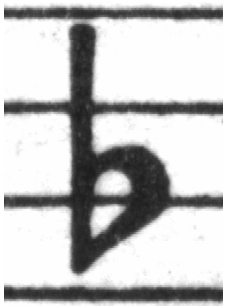
Die Kunst des Notensatzes wird auch als Notenstich bezeichnet. Dieser Begriff stammt aus dem traditionellen Notendruck. Noch bis vor etwa 20 Jahren wurden Noten erstellt, indem man sie in eine Zink- oder Zinnplatte schnitt oder mit Stempeln schlug. Diese Platte wurde dann mit Druckerschwärze versehen, so dass sie in den geschnittenen und gestempelten Vertiefungen blieb. Diese Vertiefungen schwärzten dann ein auf die Platte gelegtes Papier. Das Gravieren wurde vollständig von Hand erledigt. Es war darum sehr mühsam, Korrekturen anzubringen, weshalb man von vornherein richtig schneiden musste. Es handelte sich dabei um ein sehr spezialisiertes Handwerk.

Heutzutage wird fast alle gedruckte Musik von Computern erstellt. Das hat einige deutliche Vorteile: Drucke sind billiger als die gravierten Platten und der Computersatz kann per E-Mail verschickt werden. Leider hat der intensive Einsatz des Computers die graphische Qualität des Notensatzes vermindert. Mit dem Computer erstellte Noten sehen langweilig und mechanisch aus, was es erschwert, von ihnen zu spielen.

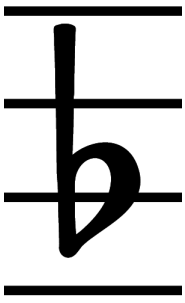
Die Abbildung unten illustriert den Unterschied zwischen traditionellem Notensatz und einem typischen Computersatz. Das dritte Bild zeigt, wie LilyPond die Formen des traditionellen Satzes nachahmt. Das linke Bild zeigt ein eingescanntes b-Vorzeichen aus einer 2000 herausgegebenen Edition. Das mittlere Bild zeigt das b-Vorzeichen der selben Musik aus einer handgestochenen Bärenreiter-Ausgabe. Das linke Bild zeigt die typischen Makel des Computer-Satzes: Die Notenlinien sind sehr dünn, die Schwärze des Vorzeichens entspricht den dünnen Linien und hat eine gerade Form mit scharfen Ecken und Kanten. Im Gegensatz dazu hat das Bärenreiter-Vorzeichen dicke, gerade zu sinnlich rundliche Formen. Unser Symbol für das Vorzeichen hat neben anderen auch dieses b als Vorbild. Es ist abgerundet und passt zu unseren Notenlinien, die sehr viel dicker sind als die der entsprechenden Computer-Ausgabe.



Henle (2000)
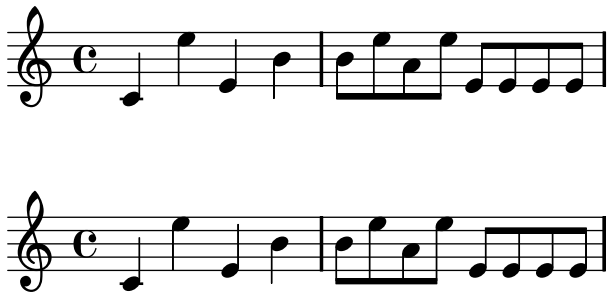


Bärenreiter (1950)



LilyPond
Feta-Schriftart
(2003)

Die Verteilung der Noten innerhalb des Taktes sollte ihrer Dauer entsprechen. Moderne Partituren zeigen diese Verhältnisse jedoch mit einer mathematischen Präzision, die nur sehr schlechte Ergebnisse bringt. Im nächsten Beispiel ist ein Motiv zweimal gesetzt: einmal mit den exakten mathematischen Längenverhältnissen, dann mit kleinen Korrekturen. Welches von beiden ist mit dieser Korrektur gesetzt?



In diesem Ausschnitt kommen nur Viertel vor, Noten, die in einem gleichmäßigen Rhythmus gespielt werden. Die Abstände sollten das widerspiegeln. Leider lässt uns aber das Auge im Stich: es beachtet nicht nur den Abstand von aufeinander folgenden Notenköpfen, sondern auch den ihrer Hälse. Also müssen Noten, deren Hälse in direkter Folge zuerst nach oben und dann nach unten ausgerichtet sind, weiter auseinander gezogen werden, während die unten/oben-Folge engere Abstände fordert, und das alles auch noch in Abhängigkeit von der vertikalen Position der Noten. Das obere Beispiel ist mit dieser Korrektur gesetzt, das untere ohne. In letzterem Fall bilden sich für das Auge bei unten/oben-Folgen Notenklumpen mit schmalen Abständen zwischen den Notenhälsen.

Musiker sind üblicherweise zu zu konzentriert, die Musik aufzuführen, als das Aussehen der Noten zu studieren; und diese Beschäftigung mit typographischen Details mag akademisch wirken. Das ist sie aber nicht. Unser Beispielstück hat einen monotonen Rhythmus, und wenn alle Zeilen gleich aussehen, wird das Notenblatt zu einem Labyrinth. Wenn der Spieler auch nur einmal wegschaut oder kurze Zeit unkonzentriert ist, findet er nicht mehr zurück zu der Stelle, an der er war.

Der dichtere Eindruck, den die dickeren Notenlinien und schwereren Notationssymbole schaffen, eignet sich auch besser für Noten, die weit vom Leser entfernt stehen, etwa auf einem Notenständer. Eine sorgfältige Verteilung der Zwischenräume erlaubt es, die Noten sehr dicht zu setzen, ohne dass die Symbole zusammenklumpen. Dadurch werden unnötige Seitenumbrüche vermieden, sodass man nicht so oft blättern muss.

Dies sind die Anforderungen der Typographie: Das Layout sollte schön sein – nicht aus Selbstzweck, sondern um dem Leser zu helfen. Für Aufführungsmaterial ist das umso wichtiger, denn Musiker haben eine begrenzte Aufmerksamkeit. Je weniger Mühe nötig ist, die Noten zu erfassen, desto mehr Zeit bleibt für die Gestaltung der eigentlichen Musik. Das heißt: Gute Typographie führt zu besseren Aufführungen!

Die Beispiele haben gezeigt, dass der Notensatz eine subtile und komplexe Kunst ist und gute Ergebnisse nur mit viel Erfahrung erlangt werden können, die Musiker normalerweise nicht haben. LilyPond stellt unser Bemühen dar, die graphische Qualität handgestochener Notenseiten ins Computer-Zeitalter zu transportieren und sie für normale Musiker erreichbar zu machen. Wir haben unsere Algorithmen, die Gestalt der Symbole und die Programm-Einstellungen darauf abgestimmt, einen Ausdruck zu erzielen, der der Qualität der alten Editionen entspricht, die wir so gerne betrachten und von denen wir gerne spielen.

Automatisierter Notensatz

Wie sollen wir also jetzt die Typographie anwenden? Wie können wir erwarten, dass wir in der Lage wären, ein Programm zu schreiben, dass den Beruf des Notenstechers ersetzt, wo dieser doch mehr als zehn Jahre braucht, um ein Meister zu werden?

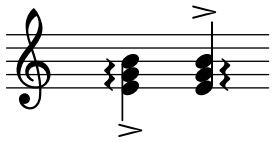
Wir können es tatsächlich nicht! Da Typographie allein durch das menschliche Auge bestimmt ist, kann der Mensch nicht ersetzt werden. Aber sehr viel mechanische Arbeit kann automatisiert werden. Indem etwa LilyPond die üblichen Situationen kennt und bewältigt, können die restlichen Fehler von Hand beseitigt werden. Das ist schon ein großer Fortschritt im Vergleich zu den existierenden Programmen. Und mit der Zeit können immer mehr Fälle automatisiert werden, so dass immer weniger Eingriffe von Hand notwendig werden.

Als wir anfingen, haben wir LilyPond vollständig in der Programmiersprache C++ geschrieben. Das hieß, dass der Funktionsumfang des Programms vollständig durch die Programmierer festgelegt war. Das stellte sich aus einer Reihe von Gründen als unzureichend heraus:
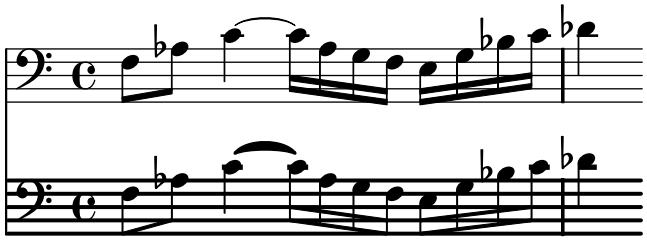
- Wenn LilyPond Fehler macht, muss der Benutzer die Einstellungen ändern können. Er muss also Zugang zur Formatierungsmaschinerie haben. Deshalb können die Regeln und Einstellungen nicht beim Kompilieren des Programms festgelegt werden, sondern sie müssen während des Laufes zugänglich sein.
- Notensatz ist eine Frage des Augenmaßes, und damit auch vom Geschmack abhängig. Benutzer können mit unseren Entscheidungen unzufrieden sein. Darum müssen also auch die Definitionen des typographischen Stils dem Benutzer zugänglich sein.
- Schließlich verfeinern wir unseren Formatierungsalgorithmus immer weiter, also müssen die Regeln auch flexibel sein. Die Sprache C++ zwingt zu einer bestimmten Gruppierungsmethode, die nicht den Regeln für den Notensatz entspricht.

Diese Probleme wurden angegangen, indem ein Übersetzer für die Programmiersprache Scheme integriert wurde und Teile von LilyPond in Scheme neu geschrieben wurden. Die derzeitige Formatierungsarchitektur ist um die Notation von graphischen Objekten herum aufgebaut, die von Scheme-Variablen und -Funktionen beschrieben werden. Diese Architektur umfasst Formatierungsregeln, typographische Stile und individuelle Formatierungsentscheidungen. Der Benutzer hat direkten Zugang zu den meisten dieser Einstellungen.

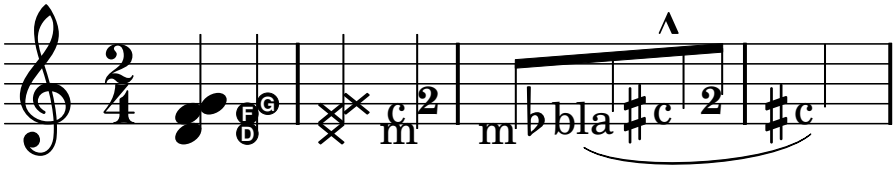
Scheme-Variablen steuern Layout-Entscheidungen. Zum Beispiel haben viele graphische Objekte eine Richtungsvariable, die zwischen oben und unten (oder rechts und links) wählen kann. Hier etwa sind zwei Akkorde mit Akzenten und Arpeggien. Beim ersten Akkord sind alle Objekte nach unten (oder links) ausgerichtet, beim zweiten nach oben (rechts).



Der Prozess des Notensetzens besteht für das Programm darin, die Variablen der graphischen Objekte zu lesen und zu schreiben. Einige Variablen haben festgelegte Werte. So ist etwa die Dicke von vielen Linien – ein Charakteristikum des typographischen Stils – von vornherein festgelegt. Wenn sie geändert werden, ergibt sich ein anderer typographischer Eindruck.



Formatierungsregeln sind auch vorbelegte Variablen. Zu jedem Objekt gehören Variablen, die Prozeduren enthalten. Diese Prozeduren machen die eigentliche Satzarbeit, und wenn man sie durch andere ersetzt, kann die Darstellung von Objekten verändert werden. Im nächsten Beispiel wird die Regel, nach der die Notenköpfe gezeichnet werden, während des Ausschnitts verändert.



Welche Symbole?

Während des Notensatzprozesses entscheidet sich, wo Symbole platziert werden. Das kann aber nur gelingen, wenn vorher entschieden wird, *welche* Symbole gesetzt werden sollen, also welche Notation benutzt werden soll.
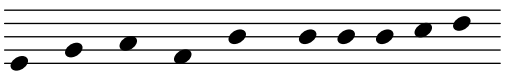
Die heutige Notation ist ein System zur Musikaufzeichnung, das sich über die letzten 1000 Jahre entwickelt hat. Die Form, die heute üblicherweise benutzt wird, stammt aus dem frühen Barock. Auch wenn sich die grundlegenden Formen (also die Notenköpfe, das Fünfliniensystem) nicht verändert haben, entwickeln sich die Details trotzdem immer noch weiter, um die Errungenschaften der Neuen Musik darstellen zu können. Die Notation umfasst also 500 Jahre Musikgeschichte. Ihre Anwendung reicht von monophonen Melodien bis zu ungeheurem Kontrapunkt für großes Orchester.

Wie bekommen wir dieses vielköpfige Monster zu fassen? Unsere Lösung ist es, eine strikte Trennung zwischen der Notation, also welche Symbole benutzt werden, und dem Satz, also wohin sie gesetzt werden, zu machen. Um das Problem anzupacken, haben wir es in kleine (programmierbare) Happen zerteilt, so dass jede Art von Symbol durch ein eigenes Plugin verarbeitet wird. Alle Plugins kooperieren durch die LilyPond-Architektur. Sie sind vollständig modular und unabhängig und können somit auch unabhängig voneinander entwickelt werden. Der Schreiber, der die Musik in Graphik umwandelt, ist ein Kopist oder Notenstecher (engl. engraver). Darum werden die Plugins als `engraver` bezeichnet.

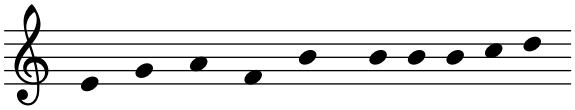
Im nächsten Beispiel wird gezeigt, wie mit dem Plugin für die Notenköpfe, dem `Note_heads_engraver` („Notenkopfstecher“) der Satz begonnen wird.
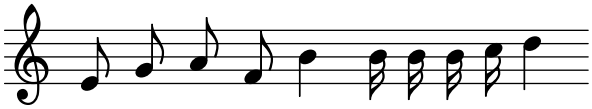


Dann fügt ein `Staff_symbol_engraver` („Notensystemstecher“) die Notenlinien hinzu.



Der `Clef_engraver` („Notenschlüsselstecher“) definiert eine Referenzstelle für das System.
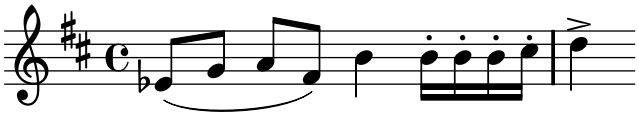


Der `Stem_engraver` („Halsstecher“) schließlich fügt Hälse hinzu.
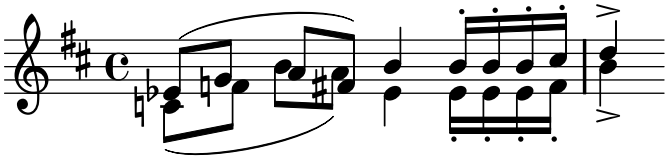


Dem `Stem_engraver` wird jeder Notenkopf mitgeteilt, der vorkommt. Jedes Mal, wenn ein Notenkopf erscheint (oder mehrere bei einem Akkord), wird ein Hals-Objekt erstellt und an den

Kopf geheftet. Wenn wir dann noch engraver für Balken, Bögen, Akzente, Vorzeichen, Taktlinien, Taktangaben und Tonartbezeichnungen hinzufügen, erhalten wir eine vollständige Notation.



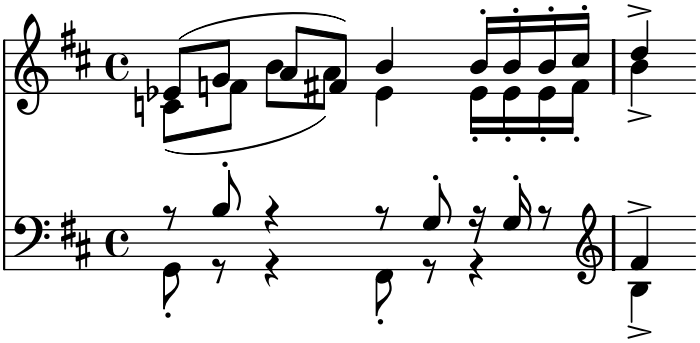
Dieses System funktioniert gut für monophone Musik, aber wie geht es mit Polyphonie? Hier müssen sich mehrere Stimmen ein System teilen.



In diesem Fall benutzen beide Stimmen das System und die Vorzeichen gemeinsam, aber die Hälse, Bögen, Balken usw. sind jeder einzelnen Stimme eigen. Die engraver müssen also gruppiert werden. Die Köpfe, Hälse, Bögen usw. werden in einer Gruppe mit dem Namen „Voice context“ (Stimmenkontext) zusammengefasst, die engraver für den Schlüssel, die Vorzeichen, Taktstriche usw. dagegen in einer Gruppe mit dem Namen „Staff context“ (Systemkontext). Im Falle von Polyphonie hat ein Staff-Kontext dann also mehr als nur einen Voice-Kontext. Auf gleiche Weise können auch mehrere Staff-Kontexte in einen großen Score-Kontext (Partiturkontext) eingebunden werden.

Siehe auch

Programmreferenz: Abschnitt “Contexts” in *Referenz der Interna*.
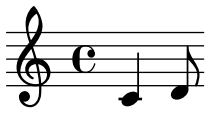


Die Darstellung der Musik

Idealerweise ist das Eingabeformat für ein höheres Satzsystem die abstrakte Beschreibung des Inhaltes. In diesem Fall wäre das die Musik selber. Das stellt uns aber vor ein ziemlich großes Problem, denn wie können wir definieren, was Musik wirklich ist? Anstatt darauf eine Antwort zu suchen, haben wir die Frage einfach umgedreht. Wir schreiben ein Programm, das den Notensatz beherrscht und passen das Format an, so einfach wie möglich zu sein. Wenn es nicht mehr vereinfacht werden kann, haben wir per Definition nur noch den reinen Inhalt. Unser Format dient als die formale Definition eines Musiktextes.

Die Syntax ist gleichzeitig die Benutzerschnittstelle bei LilyPond, darum soll sie einfach zu schreiben sein; z. B. bedeutet

```
c'4 d'8
```

eine Viertel c’ und eine Achtel d’, wie in diesem Beispiel:



In kleinem Rahmen ist diese Syntax sehr einfach zu benutzen. In größeren Zusammenhängen aber brauchen wir Struktur. Wie sonst kann man große Opern oder Symphonien notieren? Diese Struktur wird gewährleistet durch sog. music expressions (Musikausdrücke): indem kleine Teile zu größeren kombiniert werden, kann komplexere Musik dargestellt werden. So etwa hier:

```
f4
```



Gleichzeitig erklingende Noten werden hinzugefügt, indem man alle in << und >> einschließt.

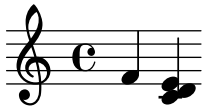
```
<<c4 d4 e4>>
```



Um aufeinanderfolgende Noten darzustellen, werden sie in geschweifte Klammern gefasst:
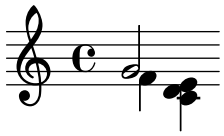
```
{ ... }
```

```
{ f4 <<c4 d4 e4>> }
```



Dieses Gebilde ist in sich wieder ein Ausdruck, und kann daher mit einem anderen Ausdruck kombiniert werden (hier mit einer Halben).

```
<< g2 \\ { f4 <<c4 d4 e4>> } >>
```
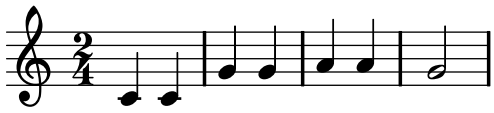


Solche geschachtelten Strukturen können sehr gut in einer kontextunabhängigen Grammatik beschrieben werden. Der Programmcode für den Satz ist auch mit solch einer Grammatik erstellt. Die Syntax von LilyPond ist also klar und ohne Zweideutigkeiten definiert.

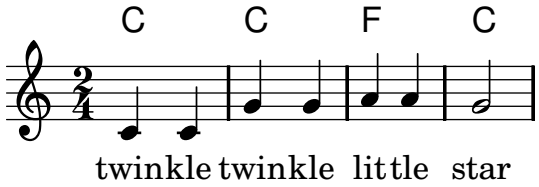
Die Benutzerschnittstelle und die Syntax werden als erstes vom Benutzer wahrgenommen. Teilweise sind sie eine Frage des Geschmackes und werden viel diskutiert. Auch wenn Geschmacksfragen ihre Berechtigung haben, sind sie nicht sehr produktiv. Im großen Rahmen von LilyPond spielt die Eingabe-Syntax nur eine geringe Rolle, denn eine logische Syntax zu schreiben ist einfach, guten Formatierungscode aber sehr viel schwieriger. Das kann auch die Zeilenzahl der Programmzeilen zeigen: Analysieren und Darstellen nimmt nur etwa 10% des Codes ein:

Beispielanwendung

Wir haben LilyPond als einen Versuch geschrieben, wie man die Kunst des Musiksatzes in ein Computerprogramm gießen kann. Dieses Programm kann nun dank vieler harter Arbeitsstunden benutzt werden, um sinnvolle Aufgaben zu erledigen. Die einfachste ist dabei der Notendruck.



Indem wir Akkordsymbole und einen Text hinzufügen, erhalten wir ein Lead Sheet.

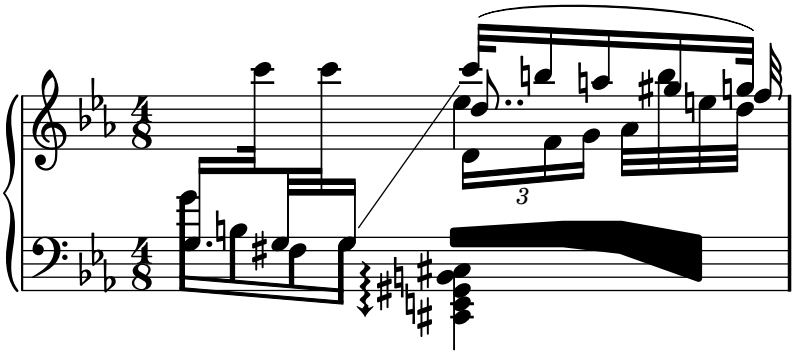


Mehrstimmige Notation und Klaviermusik kann auch gesetzt werden. Das nächste Beispiel zeigt einige etwas exotischere Konstruktionen:

Screech and boink

Random complex notation

Han-Wen Nienhuys



Die obenstehenden Beispiele wurde manuell erstellt, aber das ist nicht die einzige Möglichkeit. Da der Satz fast vollständig automatisch abläuft, kann er auch von anderen Programmen angesteuert werden, die Musik oder Noten verarbeiten. So können etwa ganze Datenbanken musikalischer Fragmente automatisch in Notenbilder umgewandelt werden, die dann auf Internetseiten oder in Multimediapräsentation Anwendung finden.

Dieses Benutzerhandbuch zeigt eine weitere Möglichkeit: Die Noten werden als reiner Text eingegeben und können darum sehr einfach integriert werden in andere textbasierte Formate wie etwa LaTeX, HTML oder, wie in diesem Fall, Texinfo. Durch ein spezielles Programm werden die Eingabefragmente durch Notenbilder in der resultierenden PDF- oder HTML-Datei ersetzt. Dadurch ist es sehr einfach, Noten und Text zu kombinieren.

1.2 Über die Dokumentation

Die Dokumentation zu LilyPond ist unterteilt in mehrere Handbücher.

Über das Handbuch zum Lernen (LM)

Dieses Handbuch erklärt die Grundbegriffe von LilyPond und stellt die fundamentalen Konzepte hinter dem Programm vor. Diese Kapitel sollten in linearer Reihenfolge gelesen werden.

Am Ende jedes Abschnitts findet sich ein Absatz **Siehe auch**, der Kreuzreferenzen zu anderen Abschnitten enthält. Beim ersten Durchlesen empfiehlt es sich nicht, diesen gleich zu folgen, da meist noch zahlreiche Grundbegriffe zum Verständnis fehlen. Wenn Sie sich durch das Handbuch zum Lernen geackert haben, wollen Sie vielleicht einzelne Abschnitte nochmal durchgehen und dann den Kreuzverweisen zur Vertiefung der Zusammenhänge folgen.

- Kapitel 1 [Einleitung], Seite 2: erklärt den Hintergrund und das Ziel von LilyPond.
- Kapitel 2 [Übung], Seite 12: liefert eine einfache Einführung in das Setzen von Musik mit LilyPond. Neulinge sollten mit diesem Kapitel beginnen.
- Kapitel 3 [Grundbegriffe], Seite 41: erklärt etliche allgemeine Konzepte hinter dem Dateiformat von LilyPond. Wenn Sie sich nicht sicher sind, an welcher Stelle Sie einen Befehl in die Datei einfügen sollen, ist dieses Kapitel genau das richtige!
- Kapitel 4 [Die Ausgabe verändern], Seite 86: stellt dar, wie die Standardeinstellungen von LilyPond verändert werden können.
- Kapitel 5 [An LilyPond-Projekten arbeiten], Seite 139: liefert Tipps im praktischen Umgang mit Lilypond und gibt Hinweise, wie gängige Fehler vermieden werden können. Bevor Sie mit einem großen Projekt beginnen, sollten Sie dieses Kapitel unbedingt gelesen haben!

Das Handbuch zum Lernen enthält auch zahlreiche Anhänge, die nicht zum linearen Durchlesen geeignet sind. Sie sind allerdings zur späteren Referenz sehr gut geeignet:

- Anhang A [Vorlagen], Seite 150: zeigt einige fertige Dokumentvorlagen für diverse Stücke mit unterschiedlichen Charakteristika. Kopieren Sie einfach die Vorlagen in Ihre eigene Datei, fügen Sie die Noten hinzu und Sie sind fertig!
- Anhang B [Scheme-Übung], Seite 181: liefert eine kurze Einführung in Scheme, die Programmiersprache, die die Musikfunktionen in LilyPond intern benutzen. Dies stellt tiefgehendes Wissen dar, wenn Sie LilyPond bis ins kleinste Detail konfigurieren möchten. Die meisten Benutzer brauchen dies jedoch selten bis gar nicht.

Über das Glossar (MG)

Abschnitt “Das Glossar” in *Glossar* erklärt musikalische Fachausdrücke und enthält auch deren Übersetzungen in diverse Sprachen. Wenn Sie mit Musiknotation oder der (englisch-sprachigen) Musikterminologie nicht vertraut sind (vor allem, wenn Englisch nicht Ihre Muttersprache ist), ist es sehr empfehlenswert, das Glossar immer wieder zu Rate zu ziehen.

Über die Notationsreferenz (NR)

In diesem Buch werden alle LilyPond-Befehle erklärt, die Notationszeichen produzieren. Es geht von der Annahme aus, dass der Leser sich mit den Grundkonzepten des Programmes im Handbuch zum Lernen bekannt gemacht hat.

- *Abschnitt “Musikalische Notation” in Notationsreferenz*: erklärt alles über die grundlegenden Notationskonstruktionen. Dieses Kapitel ist für fast jedes Notationsprojekt nützlich.
- *Abschnitt “Spezielle Notation” in Notationsreferenz*: erklärt spezifische Schwierigkeiten, die sich bei bestimmten Notationstypen ergeben. Dieses Kapitel ist nur in entsprechenden Fällen bestimmter Instrumente oder bei Gesang zu konsultieren.

- *Abschnitt "Allgemeine Eingabe und Ausgabe" in Notationsreferenz*: erläutert allgemeine Informationen über die Eingabedateien von Lilypond und wie die Ausgabe gesteuert werden kann.
- *Abschnitt "Abstände" in Notationsreferenz*: befasst sich mit globalen Fragen wie der Definition von Papierformaten oder wie man Seitenumbrüche definiert.
- *Abschnitt "Standardeinstellungen verändern" in Notationsreferenz*: erklärt, wie des Layout getrimmt werden kann um genau zum gewünschten Ergebnis zu kommen.
- *Abschnitt "Schnittstellen für Programmierer" in Notationsreferenz*: demonstriert die Erstellung von musikalischen Funktionen.

Ds Benutzerhandbuch enthält auch Anhänge mit nützlichen Referenztabellen.

- Die Abschnitt "Literatur" in *Notationsreferenz* enthält einige wichtige Quellen für alle, die mehr über Notation und den Notensatz erfahren wollen.
- Abschnitt "Notationsübersicht" in *Notationsreferenz* sind Tabellen, in denen Akkordbezeichnungen, MIDI-Instrumente, Farbbezeichnungen und die Zeichen der Feta-Schriftart gesammelt sind.
- Die *Abschnitt "Befehlsübersicht" in Notationsreferenz* zeigt die wichtigsten LilyPond-Befehle.
- Der *Abschnitt "Index der LilyPond-Befehle" in Notationsreferenz* listet alle Befehle auf, die mit `\` anfangen.
- Der *Anhang D [LilyPond-Index], Seite 190* ist ein vollständiger Index.

Über die Anwendungsbenutzung (AU)

In diesem Buch wird erklärt, wie das Programm ausgeführt wird und wie die Notation von LilyPond in andere Programme integriert werden kann.

- *Abschnitt "Installieren" in Anwendungsbenutzung*: erklärt wie LilyPond installiert wird (inklusive Kompilation, wenn es nötig sein sollte).
- *Abschnitt "Setup" in Anwendungsbenutzung*: erklärt wie der Computer eingerichtet wird, damit LilyPond optimal genutzt werden kann. Hierzu gehören etwa spezielle Umgebungen für bestimmte Texteditoren.
- *Abschnitt "LilyPond starten" in Anwendungsbenutzung*: zeigt, wie LilyPond und seine Hilfsprogramme gestartet werden. Zusätzlich wird hier erklärt, wie Quelldateien von alten LilyPond-Versionen aktualisiert werden können.
- *Abschnitt "LilyPond-book" in Anwendungsbenutzung*: erklärt die Details, um einen Text mit eingefügten Notenbeispielen (wie etwa dieses Handbuch) zu erstellen.
- *Abschnitt "Von anderen Formaten konvertieren" in Anwendungsbenutzung*: erklärt, wie die Konvertierungsprogramme aufgerufen werden. Diese Programme kommen mit LilyPond zusammen und konvertieren eine Vielzahl von Notensatzformaten in das `.ly`-Format.

Über die Schnipselliste

Die Abschnitt "LilyPond-Schnipsel" in *Schnipsel* sind eine ausführliche Sammlung kurzer Beispiele, anhand derer Tricks, Tipps und Spezialfunktionen von LilyPond demonstriert werden. Die meisten dieser Schnipsel können auch im LilyPond-Schnipsel-Depot betrachtet werden. Diese Internetseite verfügt auch über ein durchsuchbares LilyPond-Handbuch.

Die Liste der Schnipsel zu einem Abschnitt des Benutzerhandbuchs ist auch dort jeweils im Abschnitt **Siehe auch** verlinkt.

Über die Referenz der Interna (IR)

Die Abschnitt "Referenz der Interna" in *Referenz der Interna* ist eine Sammlung intensiv verlinkter HTML-Seiten, die alle Details jeder einzelnen LilyPond-Klasse, jedes Objektes und jeder Funktion erklären. Sie wird direkt aus den Satzdefinitionen im Quellcode produziert.

So gut wie alle Formatierungsmöglichkeiten, die intern verwendet werden, sind auch direkt für den Benutzer zugänglich. Alle Variablen z. B., die Dicke-Werte, Entfernungen usw. kontrollieren, können in den Eingabe-Dateien verändert werden. Es gibt eine riesige Anzahl von Formatierungsoptionen, und alle haben einen „Siehe"-Abschnitt, der auf die Dokumentation verweist. Im HTML-Handbuch haben diese Abschnitte klickbare Links.

Die Programmreferenz ist nur auf englisch verfügbar.

Andere Dokumentation

Es gibt noch eine Reihe weiterer wertvoller Informationsquellen zu LilyPond:

- Neuigkeiten: eine Zusammenfassung der Änderungen in LilyPond seit der letzten Version.
- Das Archiv der lilypond-user Mailing-Liste: enthält alle bisher an die Liste gesendeten Mails. Viele Fragen werden immer wieder gestellt und auch beantwortet. Die Chance, dass Ihre Frage auch schon mal aufgetaucht ist, ist also relativ groß. In diesem Fall finden Sie die Antwort in diesem Archiv.
- Das Archiv der lilypond-devel Mailing-Liste: enthält alle bisher an die Entwicklerliste gesendeten Mails. Diese Diskussionen sind dementsprechend technisch gehalten. Wenn Sie eine tiefergehende Frage zu den Interna von LilyPond haben, finden Sie die Antwort vielleicht in diesem Archiv.
- Eingebettete Musikbeispiele: Auf allen HTML-Seiten mit Notenbeispielen, die von LilyPond erzeugt wurden, kann die originale Quelldatei durch einen Klick auf das Bild betrachtet werden.
- Initialisierungsdateien von LilyPond:

  Der Speicherort der Dokumentationsdateien unterscheidet sich evtl. je nach Betriebssystem. Manchmal wird hier auf Initialisierungs- oder Beispieldateien verwiesen. Das Handbuch nimmt dabei an, dass diese Dateien sich relativ zum Quellverzeichnis befinden. Zum Beispiel würde der Pfad '`input/lsr/Verzeichnis/bla.ly`' etwa auf die Datei '`lilypond2.x.y/input/lsr/Verzeichnis/bla.ly`' verweisen. In den Binärpaketen für Unix-Plattformen sind Dokumentation und Beispiele üblicherweise in einem Verzeichnis wie '`/usr/share/doc/lilypond/`' gespeichert. Initialisierungsdateien, etwa '`scm/lily.scm`' oder '`ly/engraver-init.ly`', befinden sich normalerweise im Verzeichnis '`/usr/share/lilypond/`'.

  Weiterführende Informationen finden Sie unter Abschnitt 4.6.3 [Mehr Information], Seite 135.

2 Übung

Diese Übung führt ein in die Notationssprache des Programmes LilyPond und erklärt, wie man damit Noten setzen kann. Nach einer ersten Einleitung wird erklärt, wie die häufigsten Notenbilder in schönen Notendruck umgesetzt werden können.

2.1 Erste Schritte

In diesem Abschnitt werden die Grundlagen zur Benutzung des Programmes erklärt.
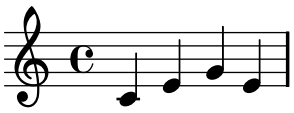
2.1.1 Eine Quelldatei übersetzen

„Kompilation“ ist der Begriff, der benutzt wird, um eine Lilypond-Eingabedatei mit dem Programm LilyPond in eine Notenausgabe umzuwandeln, die ausgedruckt werden kann. Zusätzlich besteht die Option, eine MIDI-Datei zu produzieren, die abgespielt werden kann. Das erste Beispiel zeigt, wie solch eine einfache Eingabedatei ausschauen kann.

Um Notensatz zu erstellen, muss die Notation in der Eingabedatei beschrieben werden. Wenn man z.B. schreibt:

```
{
  c' e' g' e'
}
```

so erhält man folgendes Resultat:



> **Achtung:** In jeder LilyPond-Datei müssen `{` **geschweifte Klammern** `}` um die Noten oder Gesangstext gesetzt werden. Vor und hinter den Klammern sollten Leerzeichen eingegeben werden, damit keine Unklarheiten in Verbindung mit den eigentlichen Notensymbolen entstehen. An Anfang und Ende der Zeile können diese Leerzeichen auch weggelassen werden. Es kann sein, dass in diesem Handbuch die Klammern in manchen Beispielen fehlen, aber man sollte immer daran denken, sie in den eigenen Dateien zu benutzen! Mehr Informationen zu der Darstellung der Beispiele in diesem Handbuch gibt der Abschnitt Abschnitt 2.1.4 [Wie soll das Handbuch gelesen werden], Seite 19.

Zusätzlich unterscheidet LilyPond **Groß- und Kleinschreibung**. `{ c d e }` ist zulässiger Code, `{ C D E }` dagegen resultiert in einer Fehlermeldung.

Eingabe von Noten und Ansicht des Ergebnisses

In diesem Kapitel zeigen wir, welche Kommandos eingegeben werden müssen, um ein Notenbild zu erzeugen, und wie das Resultat dann betrachtet werden kann.

Beachten Sie, dass es eine Reihe an Texteditoren mit besserer Unterstützung für LilyPond gibt. Mehr dazu im Abschnitt Abschnitt “Unterstützung von Texteditoren” in *Anwendungsbenutzung*.

Achtung: Das erste Mal, wenn Sie LilyPond benutzen, kann es eine Minute oder länger dauern, weil das Programm zuerst alle Schriftarten, die auf dem System zur Verfügung stehen, untersucht. Aber nach diesem ersten Mal läuft LilyPond sehr viel schneller.

MacOS X

Wenn Sie das `LilyPond.app`-Symbol doppelt klicken, öffnet sich eine Beispiel-Datei. Speichern Sie sie etwa als '`test.ly`' auf dem Desktop und übersetzen Sie sie mit dem Menü-Befehl „Compile > Typeset File“. Die PDF-Datei mit dem fertigen Notensatz wird automatisch geöffnet.

Das nächste Mal, wenn Sie LilyPond benutzen, sollten Sie „New“ oder „Open“ wählen. Sie müssen die Datei speichern, bevor Sie sie übersetzen können. Wenn es Fehler gibt, lesen Sie die Meldungen im Log-Fenster.

Windows

Wenn sie auf das LilyPond-Symbol auf dem Desktop doppelklicken, öffnet sich ein einfacher Texteditor mit einer Beispieldatei. Speichern Sie sie z. B. als '`test.ly`' auf dem Desktop und klicken Sie dann doppelt auf die Datei, um die Übersetzung zu beginnen (das Dateisymbol ist eine Note). Nach einigen Sekunden wird eine Datei '`test.pdf`' auf dem Desktop erscheinen. Mit einem Doppelklick kann das fertige Notenbild in der PDF-Datei angezeigt werden. Eine Alternative ist es, die '`test.ly`'-Datei mit der Maus auf das LilyPond-Symbol zu ziehen.

Um eine schon existierende Datei zu bearbeiten, klicken Sie mit der rechten Maustaste auf die Datei und wählen Sie „Edit source“. Um eine leere Datei zu erhalten, mit der Sie ein neues Stück beginnen können, öffnen Sie den Texteditor durch Doppelklick auf das LilyPond-Symbol und benutzen Sie „New“ im „File“-Menü, oder klicken Sie mit der rechten Maustaste auf den Desktop und wählen Sie „Neu...Textdatei“, ändern Sie dann den Namen so wie Sie möchten und ändern Sie die Dateiendung in `.ly`. Doppelklicken Sie auf die Datei, um Ihren LilyPond-Eingabecode einzugeben.

Mit dem Doppelklick wird nicht nur die PDF-Datei erstellt, sondern auch eine '`.log`'-Datei. Hier wird gespeichert, was LilyPond aus der Quelldatei gelesen hat. Sollten Fehler auftreten, hilft oft ein Blick in diese Datei.

UNIX

Erstellen Sie eine Text-Datei mit dem Namen '`test.ly`' und geben Sie folgenden Text ein:

```
{
  c' e' g' e'
}
```

Um die Datei zu bearbeiten, geben sie an der Konsole

```
lilypond test.ly
```

ein. Sie werden ungefähr folgende Meldungen sehen:

```
lilypond test.ly
GNU LilyPond 2.12.1

»test.ly« wird verarbeitet
Analysieren...
Interpretation der Musik...
Vorverarbeitung der grafischen Elemente...
Ideale Seitenanzahl wird gefunden...
Musik wird auf eine Seite angepasst...
Systeme erstellen...
```
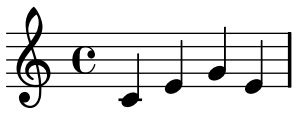
```
Layout nach »test.ps« ausgeben...
Konvertierung nach »test.pdf«...
```

Als Ergebnis erhalten Sie ein '`test.pdf`', das Sie mit den Standardprogrammen Ihres Betriebssystemes anschauen können.

2.1.2 Einfache Notation

LilyPond fügt einige Bestandteile des Notenbildes automatisch hinzu. Im nächsten Beispiel sind nur vier Tonhöhen angegeben, aber LilyPond setzt trotzdem einen Schlüssel, eine Taktangabe und Notendauern.

```
{
  c' e' g' e'
}
```



Diese Einstellungen können verändert werden, aber in den meisten Fällen sind die automatischen Werte durchaus brauchbar.
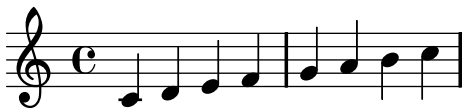
Tonhöhen

Glossar: Abschnitt "Tonhöhe" in *Glossar*, Abschnitt "Intervalle" in *Glossar*, Abschnitt "Tonleiter" in *Glossar*, Abschnitt "eingestrichenes C" in *Glossar*, Abschnitt "Oktave" in *Glossar*, Abschnitt "Versetzungszeichen" in *Glossar*.

Die Tonhöhen werden mit Kleinbuchstaben eingegeben, die den Notennamen entsprechen. Es ist jedoch wichtig zu wissen, dass LilyPond in seiner Standardeinstellung die englischen Notennamen verwendet. Bis auf eine Ausnahme entsprechen sie den deutschen, deshalb wird die Voreinstellung von LilyPond für diese Übung beibehalten. Die *Ausnahme* ist das h – in LilyPond muss man anstelle dessen b schreiben! Das deutsche b dagegen wird als bes notiert, ein his dagegen würde bis geschrieben. Siehe auch Abschnitt "Versetzungszeichen" in *Notationsreferenz* und Abschnitt "Notenbezeichnungen in anderen Sprachen" in *Notationsreferenz*, hier wird beschrieben, wie sich die deutschen Notennamen benutzen lassen.

Am einfachsten können Noten im `\relative`-Modus eingegeben werden. In diesem Modus wird die Oktave der Note automatisch gewählt, indem angenommen wird, dass die folgende Note immer so nah wie möglich in Bezug auf die vorhergehende gesetzt wird, d. h. sie wird höchstens drei Notenzeilen höher oder tiefer als die vorhergehende Note gesetzt. Fangen wir unser erstes Notationsbeispiel mit einer *scale* an, wo also die nächste Note immer nur eine Notenlinie über der vorherigen steht.
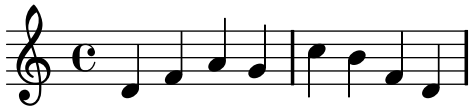
```
% Beginnpunkt auf das mittlere C setzen
\relative c' {
  c d e f
  g a b c
}
```



Die erste Note ist ein *eingestrichenes C*. Jede folgende Note befindet sich so nah wie möglich bei der vorherigen – das erste ‚C' ist also das nächste C vom eingestrichenen C aus gerechnet.

Darauf folgt das nächstmögliche D in Bezug auf die vorhergehende Note. Mit diesen Regeln können auch Melodien mit größeren Intervallen im `\relative`-Modus gebildet werden:
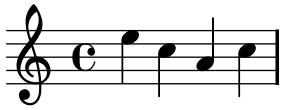
```
\relative c' {
  d f a g
  c b f d
}
```



Es ist nicht notwendig, dass die erste Note der Melodie mit der Note beginnt, die die erste Tonhöhe angibt. Die erste Note (das ‚D') des vorigen Beispiels ist das nächste D vom eingestrichenen C aus gerechnet.
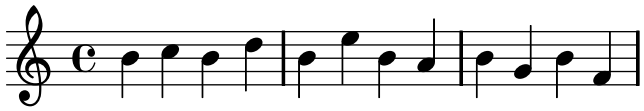
Indem man Apostrophe `'` (Taste Shift+#) oder Kommata `,` zu dem `\relative c' {` hinzufügt oder entfernt, kann die Oktave der ersten Tonhöhe verändert werden:

```
% zweigestrichenes C
\relative c'' {
  e c a c
}
```



Der relative Modus kann zunächst verwirrend erscheinen, aber es ist die einfachste Art, die meisten Melodien zu notieren. Schauen wir uns an, wie diese relative Berechnung in der Praxis funktioniert. Wenn wir mit einem H beginnen (`b` in der LilyPond-Syntax), welches sich auf der mittleren Linie im Violinschlüssel befindet, können wir C, D und E aufwärts notieren, und A, G und F unter dem H. Wenn also die Note, die auf das H folgt, ein C, D oder E ist, setzt LilyPond es oberhalb des Hs, wenn es ein A, G oder F ist, wird es darunter gesetzt.
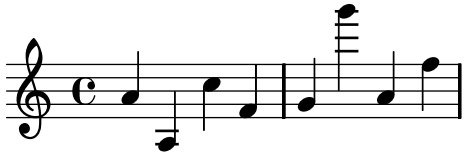
```
\relative c'' {
  b c  % c ist 1 Zeile aufwärts, also c über dem b
  b d  % d ist 2 Zeilen aufwärts, oder 5 runter, also d über dem b
  b e  % e ist 3 aufwärts oder 4 runter, also e über dem b
  b a  % a ist 6 aufwärts oder 1 runter, also a unter dem b
  b g  % g ist 5 aufwärts oder 2 runter, also g unter dem b
  b f  % f ist 4 aufwärts oder 3 runter, also f unter dem b
}
```



Die gleiche Berechnung findet auch statt, wenn eine der Noten erhöht oder erniedrigt ist. *Versetzungszeichen* werden **vollständig ignoriert** bei der Berechnung. Genau die gleiche Berechnung wird analog von jeder folgenden Tonhöhe aus für die nächste Tonhöhe neu ausgeführt.

Um Intervalle zu notieren, die größer als drei Notenzeilen sind, kann man die Oktave verändern. Mit einem Apostroph `'` (Taste Shift+#) direkt hinter dem Notennamen wird die Oktave um eins erhöht, mit einem Komma `,` um eins erniedrigt.

```
\relative c'' {
  a a, c' f,
  g g'' a,, f'
}
```



Um eine Notenhöhe um zwei (oder mehr!) Oktaven zu verändern, werden sukzessive `''` oder `,,` benutzt – es muss sich dabei wirklich um zwei einzelne Apostrophen und nicht um das Anführungszeichen `"` (Taste Shift+2) handeln! Auch die Anfangsoktave für einen `\relative c'`-Abschnitt kann so verändert werden.

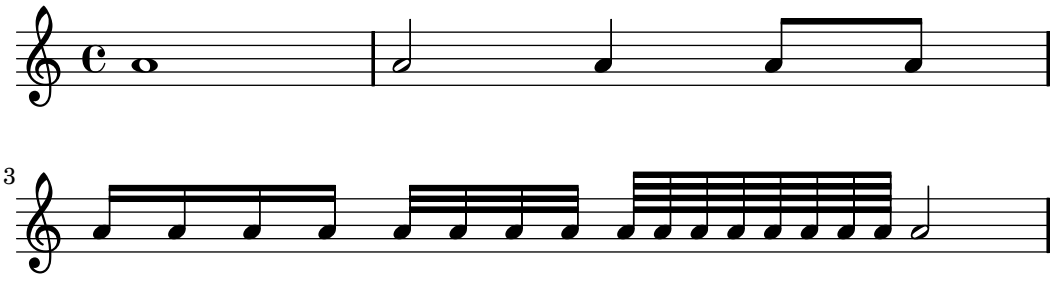
Tondauern (Rhythmen)

Glossar: Abschnitt “Balken” in *Glossar*, Abschnitt “Tondauer” in *Glossar*, Abschnitt “ganze Note” in *Glossar*, Abschnitt “halbe Note” in *Glossar*, Abschnitt “Viertelnote” in *Glossar*, Abschnitt “punktierte Note” in *Glossar*.

Die *Dauer* einer Note wird durch eine Zahl bezeichnet, die direkt auf den Notennamen folgend eingegeben wird. `1` für eine *ganze Note*, `2` für eine *halbe Note*, `4` für eine *Viertelnote* und so weiter. *Notenhälse* und *Balken* werden automatisch hinzugefügt.
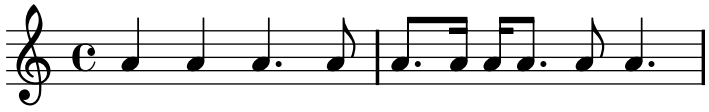
Wenn keine Dauer bezeichnet wird, wird die der vorhergehenden Note verwendet. Für die erste Note ist eine Viertel als Standard definiert.

```
\relative c'' {
  a1
  a2 a4 a8 a
  a16 a a a a32 a a a a64 a a a a a a a2
}
```



Um *punktierte Noten* zu erzeugen, wird einfach ein Punkt `.` hinter die Notendauer geschrieben. Die Dauer einer punktierten Note muss explizit, also inklusive der Nummer, angegeben werden.
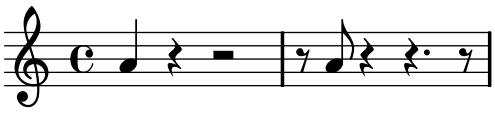
```
\relative c'' {
  a a a4. a8
  a8. a16 a a8. a8 a4.
}
```



Pausen

Eine *Pause* wird genauso wie eine Noten eingegeben; ihre Bezeichnung ist `r` :

```
\relative c'' {
  a r r2
  r8 a r4 r4. r8
}
```
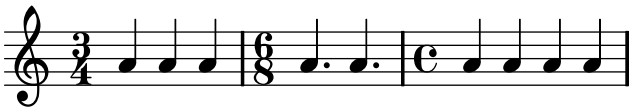


Taktangabe

Glossar: Abschnitt “Taktangabe” in *Glossar*.

Die *Taktart* kann mit dem `\time`-Befehl definiert werden:

```
\relative c'' {
  \time 3/4
  a4 a a
  \time 6/8
  a4. a
  \time 4/4
  a4 a a a
}
```
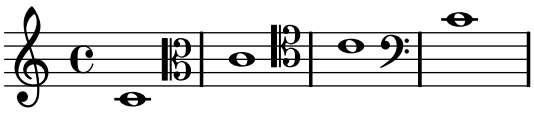


Notenschlüssel

Glossar: Abschnitt “Taktangabe” in *Glossar*.

Der *Notenschlüssel* kann mit dem `\clef`-Befehl gesetzt werden:
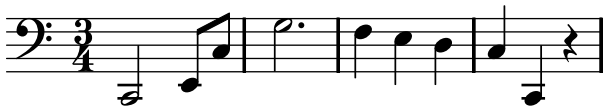
```
\relative c' {
  \clef treble
  c1
  \clef alto
  c1
  \clef tenor
  c1
  \clef bass
  c1
}
```



Alles zusammen

Hier ist ein kleines Beispiel, dass all diese Definitionen beinhaltet:

```
\relative c, {
  \time 3/4
  \clef bass
  c2 e8 c' g'2.
  f4 e d c4 c, r4
}
```



Siehe auch

Notationsreferenz: Abschnitt “Tonhöhen setzen” in *Notationsreferenz*, Abschnitt “Rhythmen eingeben” in *Notationsreferenz*, Abschnitt “Pausen eingeben” in *Notationsreferenz*, Abschnitt “Taktangabe” in *Notationsreferenz*, Abschnitt “Notenschlüssel” in *Notationsreferenz*.

2.1.3 Arbeiten an Eingabe-Dateien

LilyPonds Quelldateien ähneln Dateien in den meisten Programmiersprachen: Es ist auf Groß- und Kleinschreibung zu achten und Leerzeichen werden ignoriert. Ausdrücke werden mit geschweiften Klammern `{ }` eingeklammert und Kommentare mit dem Prozentzeichen `%` auskommentiert oder mit `%{ ... %}` umgeben.

Wenn das jetzt unverständlich erscheint, sind hier die Erklärungen:

- **Groß- und Kleinschreibung:** Die Bedeutung eines Zeichens verändert sich, je nachdem, ob es groß (`A, B, S, T`) oder klein (`a, b, s, t`) geschrieben wird. Noten müssen immer klein geschrieben werden, `{ c d e }` funktioniert, während `{ C D E }` einen Fehler produziert.
- **Leerzeichen:** Es spielt keine Rolle, wie viele Leerzeichen oder leere Zeilen sich zwischen den Zeichen der Quelldatei befinden. `{ c d e }` bedeutet das Gleiche wie `{ c  d e }` oder

```
{
c                    d
  e }
```

  Natürlich ist das letzte Beispiel etwas schwer zu lesen. Eine gute Daumenregel ist es, Code-Blöcke mit der Tab-Taste oder zwei Leerzeichen einzurücken:

```
{
  c d e
}
```

- **Ausdrücke:** Auch der kleinste Abschnitt an LilyPond-Code muss in **`{` geschweifte Klammern `}`** eingeschlossen werden. Diese Klammern zeigen LilyPond an, dass es sich um einen zusammengehörenden musikalischen Ausdruck handelt, genauso wie Klammern ‚()‘ in der Mathematik. Die Klammern sollten von jeweils einem Leerzeichen umgeben sein, um Zweideutigkeiten auszuschließen, es sei denn, sie befinden sich am Anfang oder Ende einer Zeile. Ein LilyPond-Befehl gefolgt von einem einfachen Ausdruck in Klammern (wie etwa `\relative { }`) wird auch als ein einzelner Musikausdruck gewertet.
- **Kommentare:** Ein Kommentar ist eine Bemerkung für den menschlichen Leser einer Quelldatei, es wird bei der Dateianalyse durch das Programm ignoriert, so dass es also keine Auswirkung auf die Druckausgabe der Noten hat. Es gibt zwei verschiedene Typen von Kommentaren. Das Prozentzeichen ‚%‘ geht einem Zeilen-Kommentar voraus: Alles nach diesem Zeichen wird in dieser Zeile ignoriert. Üblicherweise wird ein Kommentar *über* dem Code gesetzt, auf den es sich bezieht.

```
a4 a a a
% Dieser Kommentar bezieht sich auf das H
b2 b
```

Ein Block-Kommentar ist ein ganzer Abschnitt mit einem Kommentar. Alles, was von `%{` und `%}` umgeben ist, wird ignoriert. Das heißt, dass sich ein Block-Kommentar nicht ein einem anderen Blockkommentar befinden kann. Wenn Sie das versuchen sollten, beendet schon das erste `%}` *beide* Block-Kommentare. Das folgende Beispiel zeigt eine mögliche Anwendung von Kommentaren:

```
% Noten für twinkle twinkle hier
  c4 c g' g a a g2

%{
    Diese Zeilen, und die Noten unten werden
    ignoriert, weil sie sich in einem Block-Kommentar
    befinden.

    f f e e d d c2
%}
```

2.1.4 Wie soll das Handbuch gelesen werden
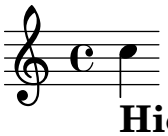
LilyPond-Code muss immer von `{ }` Zeichen oder einem `\relative c'' { ... }` umgeben sein, wie gezeigt in Abschnitt 2.1.3 [Arbeiten an Eingabe-Dateien], Seite 18. Im Rest dieses Handbuchs werden die meisten Beispiel allerdings darauf verzichten. Um sie zu reproduzieren, können Sie den entsprechenden Quellcode kopieren und in eine Textdatei einfügen, aber Sie **müssen** dabei `\relative c'' { }` einfügen, wie hier gezeigt:

```
\relative c'' {
  ... hier das Beispiel ...
}
```

Warum werden die Klammern hier meist weggelassen? Die meisten der Beispiele können in ein längeres Musikstück hineinkopiert werden, und dann ist es natürlich nicht sinnvoll, wenn auch noch `\relative c'' { }` dazukommt; ein `\relative` darf nicht innerhalb eines anderen `\relative` gesetzt werden, deshalb wird es hier weggelassen, damit die Beispiele auch innerhalb eines anderen Kontextes funktionieren. Wenn bei jedem Beispiel `\relative c'' { }` eingesetzt würde, könnten Sie die kleinen Beispiele der Dokumentation nicht einfach zu Ihrem eigenen Notentext hinzufügen. Die meisten Benutzer wollen Noten zu einer schon bestehenden Datei irgendwo in der Mitte hinzufügen, deshalb wurde der relative Modus für die Beispiele im Handbuch weggelassen.

Anklickbare Beispiele

Viele Leute lernen Programme, indem sie einfach herumprobieren. Das geht auch mit LilyPond. Wenn Sie in der HTML-Version dieses Handbuchs eine Abbildung anklicken, erhalten sie exakt den LilyPond-Code, der zum Satz der Abbildung benutzt wurde. Versuchen Sie es mit dieser Abbildung:



Hier klicken!

Wenn Sie einfach alles kopieren, was im „ly snippet“-Abschnitt steht, und in eine Text-Datei einfügen, haben Sie schon eine fertige Vorlage für weitere Experimente. Damit Sie genau das

gleiche Erscheinungsbild wie bei dem Beispiel selber erreichen, müssen Sie alles kopieren ab der Zeile „Start cut-&-pastable section“ bis ganz zum Ende der Datei.

Siehe auch

Mehr Hinweise dazu, wie LilyPond-Eingabedateien konstruiert werden sollten, finden sich in Abschnitt 5.1 [Vorschläge], Seite 139. Es ist aber wahrscheinlich am Besten, zuerst die gesamte Übung zu lesen.

2.2 Notation auf einem System

Dieses Kapitel lehrt grundlegende Bestandteile der Notation, die für eine Stimme auf einem System gebraucht werden.

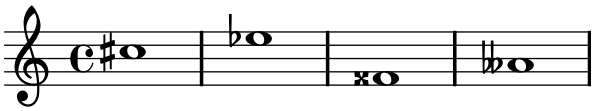
2.2.1 Versetzungszeichen und Tonartbezeichnung (Vorzeichen)

Versetzungszeichen

Glossar: Abschnitt “Kreuz” in *Glossar*, Abschnitt “B” in *Glossar*, Abschnitt “Doppelkreuz” in *Glossar*, Abschnitt “Doppel-B” in *Glossar*, Abschnitt “Versetzungszeichen” in *Glossar*.

Ein *Kreuz*-Versetzungszeichen[1] wird eingegeben, indem an den Notennamen ein ‚is‘ gehängt wird, ein *B*-Versetzungszeichen durch Anhängen von ‚es‘. Logischerweise wird dann ein *Doppelkreuz* oder *Doppel-B* durch Anhängen von ‚isis‘ oder ‚eses‘ geschrieben. Diese Syntax stammt aus der Tradition der germanischen Sprachen und ist also für deutsche Benutzer kein Problem. Es ist aber möglich, die Namen für die *Versetzungszeichen* in anderen Sprachen zu benutzen, siehe Abschnitt “Notenbezeichnungen in anderen Sprachen” in *Notationsreferenz*.

```
cis1 ees fisis, aeses
```
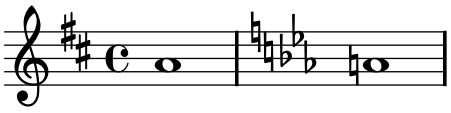


Tonartbezeichnungen (Vorzeichen)

Glossar: Abschnitt “Tonartbezeichnung” in *Glossar*, Abschnitt “Dur” in *Glossar*, Abschnitt “Moll” in *Glossar*.

Die *Tonart* eines Stückes wird erstellt mit dem Befehl `\key`, gefolgt von einer Notenbezeichnung und `\major` (für Dur) oder `\minor` (für Moll).

```
\key d \major
a1
\key c \minor
a
```



Warnung: Tonartbezeichnungen und Tonhöhen

Glossar: Abschnitt “Versetzungszeichen” in *Glossar*, Abschnitt “Tonartbezeichnung” in *Glossar*, Abschnitt “Tonhöhe” in *Glossar*, Abschnitt “B” in *Glossar*, Abschnitt “Auflösungszeichen” in *Glossar*, Abschnitt “Kreuz” in *Glossar*, Abschnitt “Transposition” in *Glossar*.
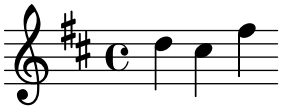
[1] In der Umgangssprache werden die Versetzungszeichen häufig auch Vorzeichen genannt. In diesem Handbuch wird jedoch zwischen Vorzeichen zur generellen Angabe der Tonart und den Versetzungszeichen, die direkt im Notentext erscheinen, unterschieden.

Um zu bestimmen, ob vor einer bestimmten Note ein Versetzungszeichen erscheinen soll, untersucht LilyPond die Notenhöhen und die Tonart. Die Tonart beeinflusst nur die *gedruckten* Versetzungszeichen, nicht die wirklichen Tonhöhen! Diese Besonderheit scheint am Anfang oft verwirrend, so dass sie hier etwas genauer betrachtet wird.

LilyPond unterscheidet strickt zwischen dem musikalischen Inhalt und dem Satz (Layout). Die Alteration (*B*, *Kreuz* oder *Auflösungszeichen*) einer Note gehört zur Tonhöhe dazu und ist deshalb musikalischer Inhalt. Ob ein Versetzungszeichen (also ein *gedrucktes* Kreuz, b oder Auflösungszeichen) auch vor der Note erscheint, hängt vom Kontext, also vom Layout ab. Das Layout gehorcht bestimmten Regeln, und Versetzungszeichen werden automatisch nach diesen Regeln gesetzt. Die Versetzungszeichen im fertigen Notenbild sind nach den Regeln des Notensatzes gesetzt. Deshalb wird automatisch entschieden, wo sie erscheinen, und man muss den Ton eingeben, den man *hören* will.

In diesem Beispiel
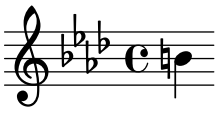
```
\key d \major
d cis fis
```



hat keine der Noten ein Versetzungszeichen, trotzdem muss im Quelltext das ‚is' für `cis` und `fis` notiert werden.

Der Code ‚b' (nach der holländischen Notenbezeichnung wird der Ton H mit `b` gesetzt) heißt also nicht: „Zeichne einen schwarzen Punkt auf die Mittellinie des Systems." Im Gegenteil, er heißt vielmehr: „Hier soll eine Note mit der Tonhöhe H gesetzt werden." In der Tonart As-Dur *bekommt* sie ein Versetzungszeichen:

```
\key aes \major
b
```



Alle diese Versetzungszeichen ausdrücklich zu schreiben, bedeutet vielleicht etwas mehr Schreibarbeit, hat aber den großen Vorteil, dass *Transpositionen* sehr viel einfacher gemacht wird und der Druck von Versetzungszeichen nach unterschiedlichen Regeln erfolgen kann. Siehe Abschnitt "Automatische Versetzungszeichen" in *Notationsreferenz* für einige Beispiele, wie Vorzeichen anhand von unterschiedlichen Regeln ausgegeben werden können.

Siehe auch

Notationsreferenz: Abschnitt "Notenbezeichnungen in anderen Sprachen" in *Notationsreferenz*, Abschnitt "Versetzungszeichen" in *Notationsreferenz*, Abschnitt "Automatische Versetzungszeichen" in *Notationsreferenz*, Abschnitt "Tonartbezeichnung" in *Notationsreferenz*.

Glossar: Abschnitt "Tonhöhenbezeichnungen" in *Glossar*.
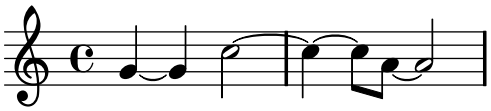
2.2.2 Bindebögen und Legatobögen

Bindebögen

Glossar: Abschnitt “Bindebogen” in *Glossar*.

Ein *Bindebogen* wird geschrieben, indem man eine Tilde ~ an die erste der zu verbindenden Noten hängt.

```
g4~ g c2~
c4 ~ c8 a8 ~ a2
```



Legatobögen

Glossar: Abschnitt “Legatobogen” in *Glossar*.

Ein *Legatobogen* ist ein Bogen, der sich über mehrere Noten erstreckt. Seine Beginn- und Endnote werden mit ‚(‘ beziehungsweise ‚)‘ markiert.

```
d4( c16) cis( d e c cis d) e( d4)
```
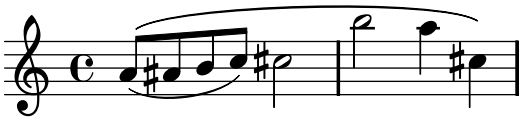


Phrasierungsbögen

Glossar: Abschnitt “Legatobogen” in *Glossar*, Abschnitt “Phrasierung” in *Glossar*.

Bögen, die längere Phrasierungseinheiten markieren (Phrasierungsbögen), werden mit `\(` und `\)` eingegeben. Es können sowohl Legato- als auch Phrasierungsbögen gleichzeitig vorkommen, aber es kann nicht mehr als jeweils einen Legato- und einen Phrasierungsbogen gleichzeitig geben.

```
a8(\( ais b c) cis2 b'2 a4 cis,\)
```
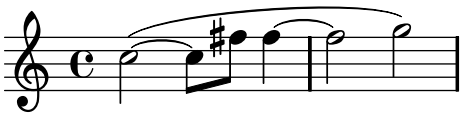


Warnung: Bindebögen sind nicht Legatobögen

Glossar: Abschnitt “Artikulationszeichen” in *Glossar*, Abschnitt “Legatobogen” in *Glossar*, Abschnitt “Bindebogen” in *Glossar*.

Ein Legatobogen sieht aus wie ein Abschnitt “Bindebogen” in *Glossar*, hat aber eine andere Bedeutung. Ein Bindebogen verlängert nur die vorhergehende Note und kann also nur bei zwei Noten gleicher Tonhöhe benutzt werden. Legatobögen dagegen zeigen die Artikulation von Noten an und können für größere Notengruppen gesetzt werden. Binde- und Legatobögen können geschachtelt werden.

```
c2~( c8 fis fis4 ~ fis2 g2)
```



Siehe auch

Notationsreferenz: Abschnitt “Bindebögen” in *Notationsreferenz*, Abschnitt “Legatobögen” in *Notationsreferenz*, Abschnitt “Phrasierungsbögen” in *Notationsreferenz*.
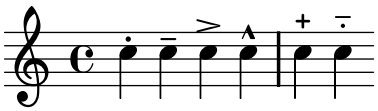
2.2.3 Artikulationszeichen und Lautstärke

Artikulationszeichen

Glossar: Abschnitt “Artikulationszeichen” in *Glossar*.

Übliche *Artikulationszeichen* können durch Anfügen von Minus (‚-‘) und einem entsprechenden Zeichen eingegeben werden:

```
c-. c-- c-> c-^ c-+ c-_
```
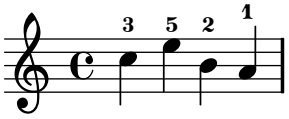


Fingersatz

Glossar: Abschnitt “Fingersatz” in *Glossar*.
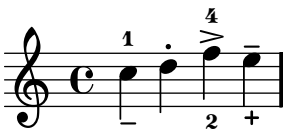
Auf gleiche Weise können Fingersatzbezeichnungen hinzugefügt werden, indem nach dem Minus (‚-‘) eine Zahl geschrieben wird:

```
c-3 e-5 b-2 a-1
```



Artikulationszeichen und Fingersätze werden normalerweise automatisch platziert, aber man kann ihre Position auch vorgeben durch die Zeichen ‚^‘ (oben) oder ‚_‘ (unten) anstelle des Minuszeichen. An eine Note können auch mehrfache Artikulationszeichen gehängt werden. Meistens findet aber LilyPond alleine die beste Möglichkeit, wie die Artikulationen platziert werden sollen.

```
c_-^1 d^. f^4_2-> e^-_+
```
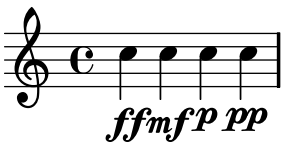


Dynamik

Glossar: Abschnitt “Dynamik” in *Glossar*, Abschnitt “Crescendo” in *Glossar*, Abschnitt “Decrescendo” in *Glossar*.
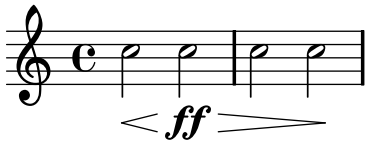
Die Dynamik innerhalb eines Stückes wird eingegeben, indem man die Markierungen (mit einem Backslash) an die Note hängt:

```
c\ff c\mf c\p c\pp
```



Crescendi und *Decrescendi* werden mit dem Befehl `\<` beziehungsweise `\>` begonnen. Das nächste absolute Dynamik-Zeichen, etwa `\f`, beendet das (De)Crescendo. Auch mit dem Befehl `\!` kann es explizit beendet werden.

```
c2\< c2\ff\> c2 c2\!
```



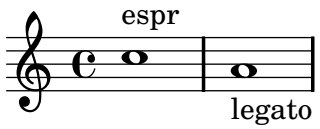
Siehe auch

Notationsreferenz: Abschnitt “Artikulationszeichen und Verzierungen” in *Notationsreferenz*, Abschnitt “Fingersatzanweisungen” in *Notationsreferenz*, Abschnitt “Dynamik” in *Notationsreferenz*.

2.2.4 Text hinzufügen

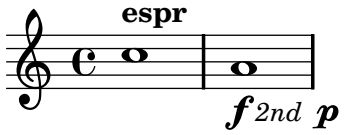
Text können Sie auf folgende Art in die Partitur einfügen:

```
c1^"espr" a_"legato"
```



Zusätzliche Formatierung kann eingesetzt werden, wenn Sie den `\markup`-Befehl benutzen:

```
c1^\markup{ \bold espr}
a1_\markup{
  \dynamic f \italic \small { 2nd } \hspace #0.1 \dynamic p
}
```
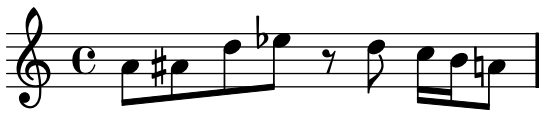


Siehe auch

Notationsreferenz: Abschnitt “Text eingeben” in *Notationsreferenz*.

2.2.5 Automatische und manuelle Balken
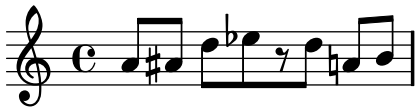
Alle *Balken* werden automatisch gesetzt:

```
a8 ais d ees r d c16 b a8
```
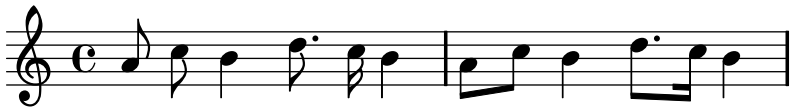


Wenn diese automatisch gesetzten Balken nicht gewollt sind, können sie manuell geändert werden. Wenn nur ein Balken hier und da korrigiert werde muss, erhält die Note, an der der Balken anfängt, ein ‚[‘ (AltGr+8) und die, an der er enden soll, ein ‚]‘ (AltGr+9).

```
a8[ ais] d[ ees r d] a b
```



Wenn Sie die automatischen Balken vollständig oder für einen längeren Abschnitt ausschalten wollen, benutzen Sie den Befehl `\autoBeamOff`, um die Balken abzuschalten, und `\autoBeamOn`, um sie wieder einzuschalten.

```
\autoBeamOff
a8 c b4 d8. c16 b4
\autoBeamOn
a8 c b4 d8. c16 b4
```



Siehe auch
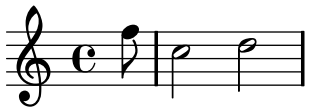
Notationsreferenz: Abschnitt “Automatische Balken” in *Notationsreferenz*, Abschnitt “Manuelle Balken” in *Notationsreferenz*.

2.2.6 Zusätzliche rhythmische Befehle

Auftakt

Ein *Auftakt* wird mit dem Befehl `\partial` eingegeben. Darauf folgt die Länge des Auftaktes: `\partial 4` heißt eine Viertelnote Auftakt und `\partial 8` eine Achtelnote.

```
\partial 8
f8 c2 d
```

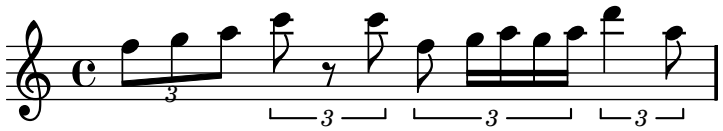


Andere rhythmische Aufteilungen

Glossar: Abschnitt “Notenwert” in *Glossar*, Abschnitt “Triole” in *Glossar*.

Triolen und *N-tolen* werden mit dem `\times`-Befehl erzeugt. Er braucht zwei Argumente: einen Bruch und die Noten, auf die er sich bezieht. Die Dauer des Abschnittes wird mit dem Bruch multipliziert. In einer Triole dauern die Noten 2/3 ihrer normalen Länge, also hat eine Triole 2/3 als Bruch:

```
\times 2/3 { f8 g a }
\times 2/3 { c r c }
\times 2/3 { f,8 g16[ a g a] }
\times 2/3 { d4 a8 }
```
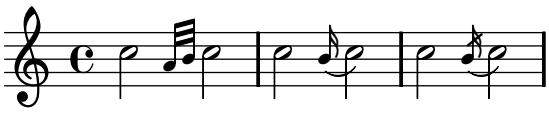


Verzierungen

Glossar: Abschnitt “Verzierungen” in *Glossar*, Abschnitt “Vorschlag” in *Glossar*, Abschnitt “Vorhalt” in *Glossar*.

Verzierungen werden mit dem Befehl `\grace` eingegeben, Vorhalte durch den Befehl `\appoggiatura` und Vorschläge mit `\acciaccatura`.

```
c2 \grace { a32[ b] } c2
c2 \appoggiatura b16 c2
c2 \acciaccatura b16 c2
```



Siehe auch

Notationsreferenz: Abschnitt “Verzierungen” in *Notationsreferenz*, Abschnitt “Andere rhythmische Aufteilungen” in *Notationsreferenz*, Abschnitt “Auftakte” in *Notationsreferenz*.

2.3 Mehrere Noten auf einmal

In diesem Kapitel wird gezeigt, wie mehr als eine Note zur gleichen Zeit gesetzt werden kann: auf unterschiedlichen Systemen für verschiedene Instrumente oder für ein Instrument (z. B. Klavier) und in Akkorden.

Polyphonie nennt man in der Musik das Vorkommen von mehr als einer Stimme in einem Stück. Polyphonie bzw. Mehrstimmigkeit heißt für LilyPond allerdings das Vorkommen von mehr als einer Stimme pro System.

2.3.1 Musikalische Ausdrücke erklärt
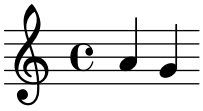
In LilyPond-Quelldateien wird Musik durch *musikalische Ausdrücke* dargestellt. Eine einzelne Note ist ein musikalischer Ausdruck.

```
a4
```
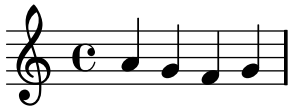


Eine Gruppe von Noten innerhalb von Klammern bildet einen neuen Ausdruck. Dieser ist nun ein *zusammengesetzter musikalischer Ausdruck*. Hier wurde solch ein zusammengesetzter musikalischer Ausdruck mit zwei Noten erstellt:

```
{ a4 g4 }
```



Wenn eine Gruppe von musikalischen Ausdrücken (also beispielsweise Noten) in geschweifte Klammern gesetzt wird, bedeutet das, dass eine Gruppe nach der anderen gesetzt wird. Das Resultat ist ein neuer musikalischer Ausdruck.

```
{ { a4 g } f g }
```



Analogie: mathematische Ausdrücke

Die Anordnung von Ausrücken funktioniert ähnlich wie mathematische Gleichungen. Eine längere Gleichung entsteht durch die Kombination kleinerer Gleichungen. Solche Gleichungen werden auch Ausdruck genannt und ihre Definition ist rekursiv, sodass beliebig komplexe und lange Ausdrücke erstellt werden können. So etwa hier:

```
1

1 + 2

(1 + 2) * 3

((1 + 2) * 3) / (4 * 5)
```

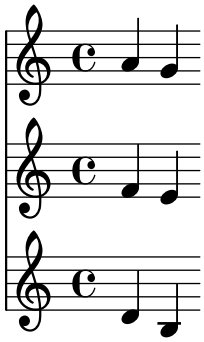
Das ist eine Folge von (mathematischen) Ausdrücken, in denen jeder Ausdruck in dem folgenden (größeren) enthalten ist. Die einfachsten Ausdrücke sind Zahlen, und größere werden durch die Kombination von Ausdrücken mit Hilfe von Operatoren (wie ‚+‘, ‚*‘ und ‚/‘) sowie Klammern. Genauso wie mathematische Ausdrücke können auch musikalische Ausdrücke beliebig tief verschachtelt werden. Das wird benötigt für komplexe Musik mit vielen Stimmen.

Gleichzeitige musikalische Ausdrücke: mehrere Notensysteme

Glossar: Abschnitt “Polyphonie” in *Glossar*.

Mit dieser Technik kann *polyphone* Musik gesetzt werden. Musikalische Ausdrücke werden einfach parallel kombiniert, damit sie gleichzeitig als eigene Stimmen in dem gleichen Notensystem gesetzt werden. Um anzuzeigen, dass an dieser Stelle gleichzeitige Noten gesetzt werden, muss nur ein Kombinationszeichen eingefügt werden. Parallel werden musikalische Ausdrücke kombiniert, indem man sie mit `<<` und `>>` einrahmt. Im folgenden Beispiel sind drei Ausdrücke (jeder mit zwei Noten) parallel kombiniert:

```
\relative c'' {
  <<
    { a4 g }
    { f e }
    { d b }
  >>
}
```
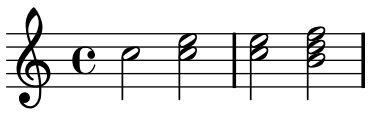


Es ist noch zu bemerken, dass wir hier für jede Ebene innerhalb der Quelldatei eine andere Einrückung geschrieben haben. Für LilyPond spielt es keine Rolle, wie viele Leerzeichen am Anfang einer Zeile sind, aber für Menschen ist es eine große Hilfe, sofort zu sehen, welche Teile des Quelltextes zusammen gehören.

> **Achtung:** Jede Note ist relativ zu der vorhergehenden in der Datei, nicht relativ zu dem zweigestrichenen C (`c''`), das im `\relative`-Befehl angegeben ist. Die Klammern haben darauf keinen Einfluss.

Gleichzeitige musikalische Ausdrücke: ein Notensystem

Um die Anzahl der Notensysteme zu bestimmen, analysiert LilyPond den Anfang des ersten Ausdrucks. Wenn sich hier eine einzelne Note befindet, wird nur ein System gesetzt, wenn es sich um eine parallele Anordnung von Ausdrücken handelt, wird mehr als ein System gesetzt. Das folgende Beispiel beginnt mit einer Note:

```
\relative c'' {
  c2 <<c e>>
  << { e f } { c <<b d>> } >>
}
```
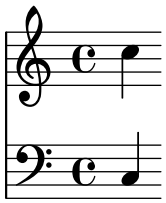


2.3.2 Mehrere Notensysteme

Wie wir in Abschnitt 2.3.1 [Musikalische Ausdrücke erklärt], Seite 26 gesehen haben, sind LilyPond-Quelldateien aus musikalischen Ausdrücken konstruiert. Wenn die Noteneingabe mit parallelen Ausdrücken beginnt, werden mehrere Notensysteme erstellt. Es ist aber sicherer und einfacherer zu verstehen, wenn diese Systeme explizit erstellt werden.

Um mehr als ein System zu schreiben, wird jedem Notenausdruck, der in einem eigenen System stehen soll, der Befehl `\new Staff` vorne angefügt. Diese `Staff` (engl. für Notensystem)-Elemente werden dann parallel angeordnet mit den `<<` und `>>`-Zeichen:

```
\relative c'' {
  <<
    \new Staff { \clef treble c }
    \new Staff { \clef bass c,, }
  >>
}
```
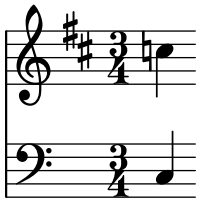


Der Befehl `\new` beginnt einen neuen „Notationskontext“. Ein solcher Notationskontext ist eine Umgebung, in der musikalische Ereignisse (wie Noten oder `\clef` (Schlüssel)-Befehle) interpretiert werden. Für einfache Stücke werden diese Umgebungen automatisch erstellt. Für kompliziertere Musik ist es aber am besten, die Umgebungen explizit zu erstellen.

Es gibt verschiedene Kontext-Typen. `Score` (Partitur), `Staff` (Notensystem) und `Voice` (Stimme) verarbeiten die Eingabe von Noten, während die `Lyrics` (Text)-Umgebung zum Setzen von Liedtexten und die `ChordNames` (Akkordbezeichnungs)-Umgebung für Akkordsymbole verwendet wird.

Indem `\new` vor einen musikalischen Ausdruck gesetzt wird, wird ein größerer Ausdruck erstellt. In diesem Sinne erinnert die Syntax des `\new`-Befehls an das Minuszeichen in der Mathematik. Genauso wie $(4 + 5)$ ein Ausdruck ist, der durch $-(4 + 5)$ zu einem größeren Ausdruck erweitert wurde, werden auch musikalische Ausdrücke durch den `\new`-Befehl erweitert.

Die Taktangabe, die in einem einzelnen System angegeben wird, wirkt sich auf alle anderen System aus. Die Angabe der Tonart in einem System hingegen beeinflusst *nicht* die Tonart der anderen Systeme. Dieses Verhalten ist darin begründet, dass Partituren mit transponierenden Instrumenten häufiger sind als Partituren mit unterschiedlichen Taktarten.

```
\relative c'' {
  <<
    \new Staff { \clef treble \key d \major \time 3/4 c }
    \new Staff { \clef bass c,, }
  >>
}
```



2.3.3 Notensysteme gruppieren
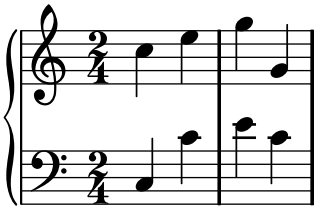
Glossar: Abschnitt “Klammer” in *Glossar*.

Musik für das Klavier wird üblicherweise auf zwei Systemen notiert, die durch eine *geschweifte Klammer* verbunden sind (Akkolade). Um ein derartiges Notensystem zu erstellen, geht man ähnlich vor wie in dem Beispiel aus Abschnitt 2.3.2 [Mehrere Notensysteme], Seite 28, nur dass der gesamte Ausdruck jetzt in eine `PianoStaff`-Umgebung eingefügt wird.

```
\new PianoStaff <<
  \new Staff ...
  \new Staff ...
>> >>
```

Hier ein kleines Beispiel:

```
\relative c'' {
  \new PianoStaff <<
    \new Staff { \time 2/4 c4 e g g, }
    \new Staff { \clef bass c,, c' e c }
  >>
}
```



Andere typische Gruppen von Notensystemen können mit den Befehlen `\new StaffGroup` für Orchestersätze und `\new ChoirStaff` für ein Chorsystem erstellt werden. Jede dieser Systemgruppen erstellt einen neuen Kontext, der dafür sorgt, dass die Klammern zu Beginn des Systems erstellt werden und der zusätzlich auch darüber entscheidet, ob die Taktlinien nur auf dem System oder auch zwischen System gesetzt werden.

Siehe auch
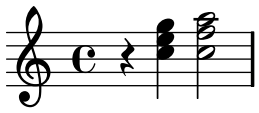
Notationsreferenz: Abschnitt “Tasteninstrumente und andere Instrumente mit mehreren Systemen” in *Notationsreferenz*, Abschnitt “Systeme anzeigen lassen” in *Notationsreferenz*.

2.3.4 Noten zu Akkorden verbinden

Glossar: Abschnitt “Akkord” in *Glossar*.
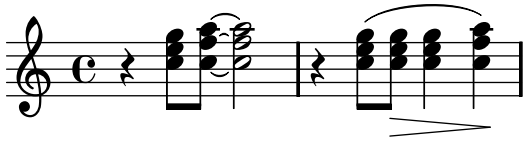
Wir haben schon weiter oben gesehen, wie Akkorde ersetellt werden können, indem sie mit spitzen Klammern eingeschlossen und somit als gleichzeitig erklingend markiert werden. Die normale Art, Akkorde zu notieren, ist aber, sie in *einfache* spitze Klammern (‚<‘ und ‚>‘) einzuschließen. Beachten Sie, dass alle Noten eines Akkordes die gleiche Dauer haben müssen, und diese Dauer wird nach der schließenden Klammer geschrieben.

```
r4 <c e g>4 <c f a>2
```



Akkorde sind im Grunde gleichwertig mit einfachen Noten: Fast alle Markierungen, die an einfache Noten angehängt werden können, kann man auch an Akkorde hängen. So können Markierungen wie Balken oder Bögen mit den Akkorden kombiniert werden. Sie müssen jedoch außerhalb der spitzen Klammern gesetzt werden.

```
r4 <c e g>8[ <c f a>]~ <c f a>2
r4 <c e g>8( <c e g>\> <c e g>4 <c f a>\!)
```
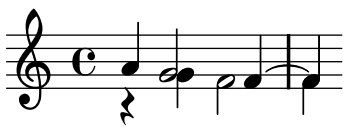


2.3.5 Mehrstimmigkeit in einem System

Wenn unterschiedliche Melodien oder Stimmen in einem System kombiniert werden sollen, werden sie als „polyphone Stimmen“ realisiert: Jede Stimme hat eigene Hälse, Balken und Legatobögen, und die Hälse der oberen Stimme zeigen immer nach oben, während die Hälse der unteren Stimme nach unten zeigen.
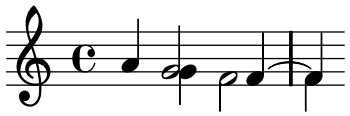
Diese Art von Notenbild wird erstellt, indem jede Stimme für sich als Abfolge notiert wird (mit `{...}`) und diese dann parallel kombiniert werden, indem die einzelnen Stimmen durch `\\` voneinander getrennt werden.

```
<<
  { a4 g2 f4~ f4 } \\
  { r4 g4 f2 f4 }
>>
```



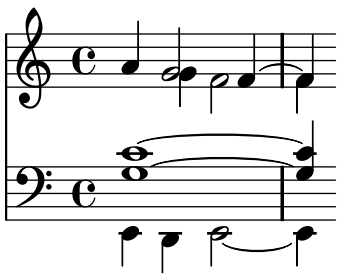
Für den Satz von mehrstimmigen Stücken kann es auch angebracht sein, unsichtbare Pausen zu verwenden. Hiermit können Stimmen ausgefüllt werden, die gerade nicht aktiv sind. Hier ist das obige Beispiel mit einer unsichtbaren Pause (‚s‘) anstelle einer normalen (‚r‘):

```
<<
  { a4 g2 f4~ f4 } \\
  { s4 g4 f2 f4 }
>>
```



Auch diese Ausdrücke wiederum könne beliebig miteinander kombiniert werden.

```
<<
  \new Staff <<
    { a4 g2 f4~ f4 } \\
    { s4 g4 f2 f4 }
  >>
  \new Staff <<
    \clef bass
    { <c g>1 ~ <c g>4 } \\
    { e,,4 d e2 ~ e4}
  >>
>>
```



Siehe auch

Notationsreferenz: Abschnitt “Gleichzeitig erscheinende Noten” in *Notationsreferenz*.

2.4 Lieder
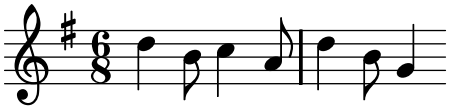
In diesem Kapitel wird in die Kombination von Musik mit Text eingeführt und die Erstellung einfacher Song-Blätter gezeigt.

2.4.1 Einfache Lieder setzen

Glossar: Abschnitt “Gesangtext” in *Glossar*.
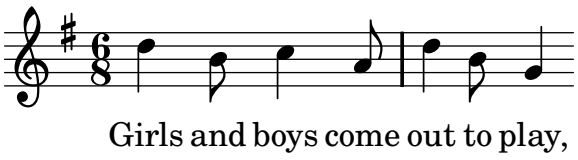
Hier ist der Beginn eines einfachen Kinderliedes, *Girls and boys come out to play*:

```
\relative c'' {
  \key g \major
  \time 6/8
  d4 b8 c4 a8 d4 b8 g4
}
```



Zu diesen Noten kann Text hinzugefügt werden, indem beide mit dem `\addlyrics`-Befehl kombiniert werden. Text wird eingegeben, indem jede Silbe durch ein Leerzeichen getrennt wird.

```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
  }
  \addlyrics {
    Girls and boys come out to play,
  }
>>
```



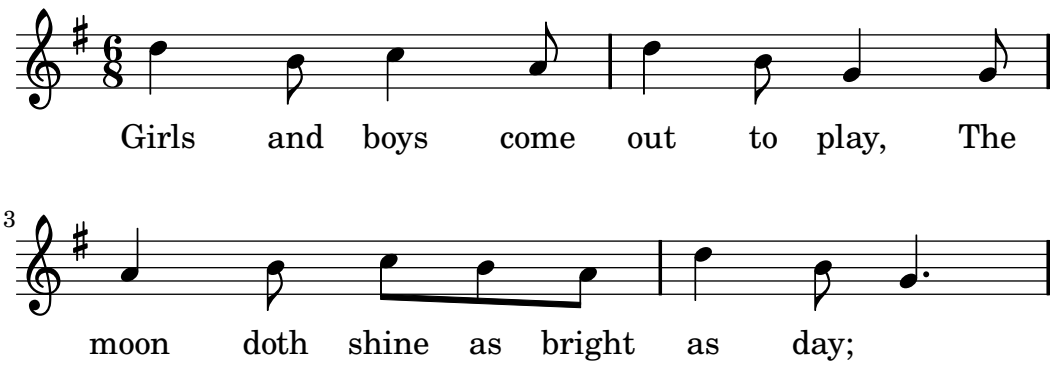
Sowohl die Noten als auch der Text sind jeweils in geschweifte Klammern eingefasst, und der gesamte Ausdruck ist zwischen `<< ... >>` positioniert. Damit wird garantiert, dass Text und Noten gleichzeitig gesetzt werden.

2.4.2 Text an einer Melodie ausrichten

Glossar: Abschnitt “Melisma” in *Glossar*, Abschnitt “Fülllinie” in *Glossar*.

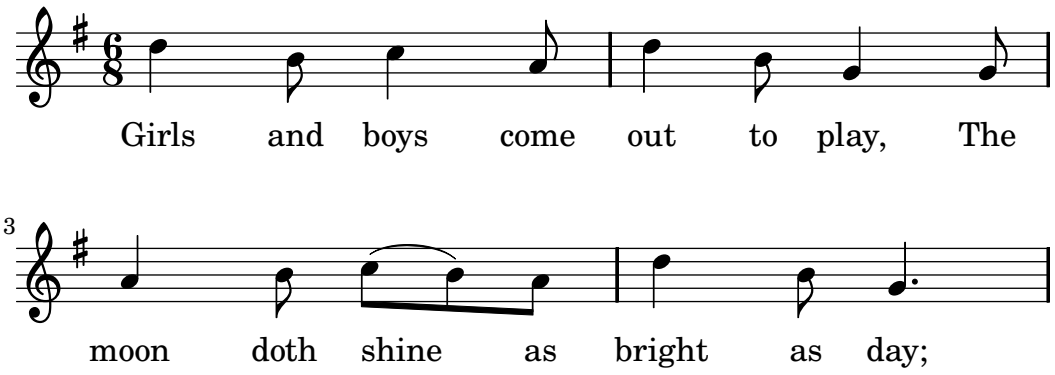
Die nächste Zeile des Kinderliedes lautet: *The moon doth shine as bright as day*. So sieht es notiert aus:

```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
    g8 a4 b8 c b a d4 b8 g4.
  }
  \addlyrics {
    Girls and boys come out to play,
    The moon doth shine as bright as day;
  }
>>
```
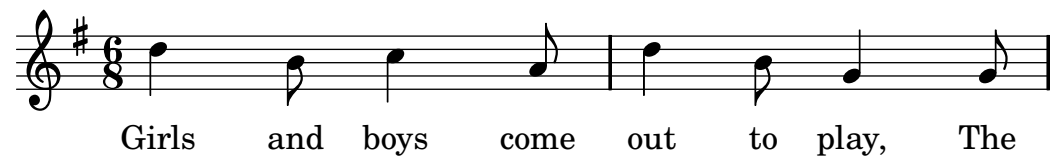


Die zusätzlichen Noten sind nicht korrekt an den Noten positioniert. Das Wort *shine* sollte eigentlich zu zwei Noten gesungen werden, nicht nur zu einer. Das wird als *Melisma* bezeichnet, wenn eine Silbe zu mehreren Noten gesungen wird. Es gibt mehrere Möglichkeiten, eine Silbe über mehrere Noten zu erweitern. Die einfachste ist, einen Legatobogen über die betroffenen Noten zu notieren, zu Einzelheiten siehe Abschnitt 2.2.2 [Bindebögen und Legatobögen], Seite 21.

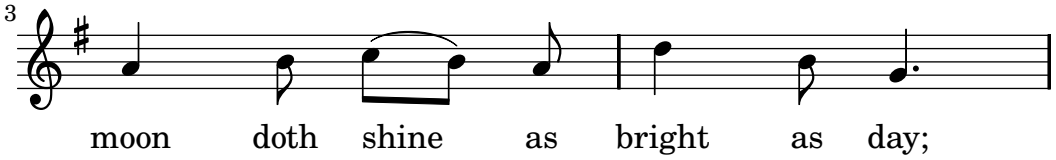
```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
    g8 a4 b8 c( b) a d4 b8 g4.
  }
  \addlyrics {
    Girls and boys come out to play,
    The moon doth shine as bright as day;
  }
>>
```



Die Wörter orientieren sich jetzt richtig an den Noten, aber der automatische Balken für die Noten zu *shine as* sieht nicht richtig aus. Wir können das korrigieren, indem wir die Balkenlänge manuell eingrenzen, damit sie der üblichen Notationsweise für Gesang entspricht. Für Einzelheiten siehe Abschnitt 2.2.5 [Automatische und manuelle Balken], Seite 24.

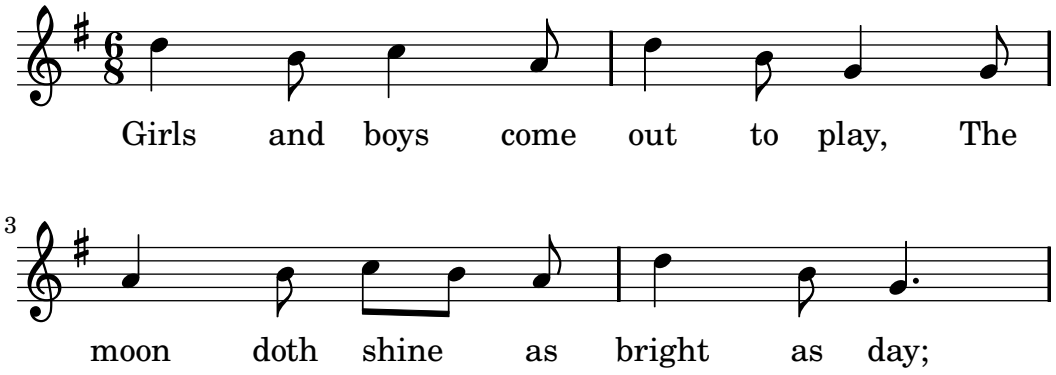
```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
    g8 a4 b8 c([ b]) a d4 b8 g4.
  }
  \addlyrics {
    Girls and boys come out to play,
    The moon doth shine as bright as day;
  }
>>
```
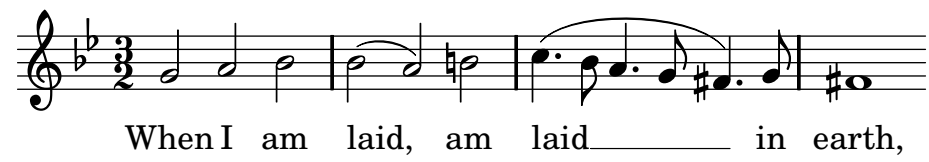




Alternativ kann das Melisma auch im Text notiert werden, indem für jede Note, die übersprungen werden soll, ein Unterstrich _ im Text geschrieben wird:

```
<<
  \relative c'' {
    \key g \major
    \time 6/8
    d4 b8 c4 a8 d4 b8 g4
    g8 a4 b8 c[ b] a d4 b8 g4.
  }
  \addlyrics {
    Girls and boys come out to play,
    The moon doth shine _ as bright as day;
  }
>>
```



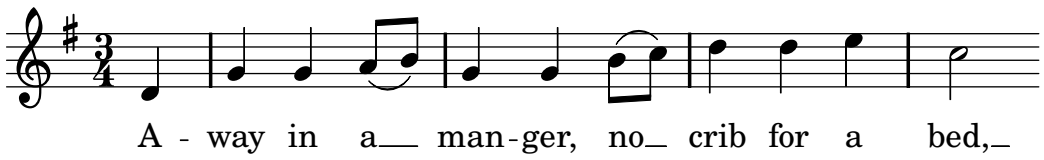
Wenn die letzte Silbe eines Wortes sich über mehrere Noten oder eine sehr lange Note erstreckt, wird üblicherweise eine Fülllinie gesetzt, die sich über alle Noten erstreckt, die zu der Silbe gehören. Diese Fülllinie wird mit zwei Unterstrichen __ notiert. Hier ein Beispiel der ersten drei Takte aus *Didos Klage*, aus Purcells *Dido and Æneas*:

```
<<
  \relative c'' {
    \key g \minor
    \time 3/2
    g2 a bes bes( a)
    b c4.( bes8 a4. g8 fis4.) g8 fis1
  }
  \addlyrics {
    When I am laid,
    am laid __ in earth,
  }
>>
```



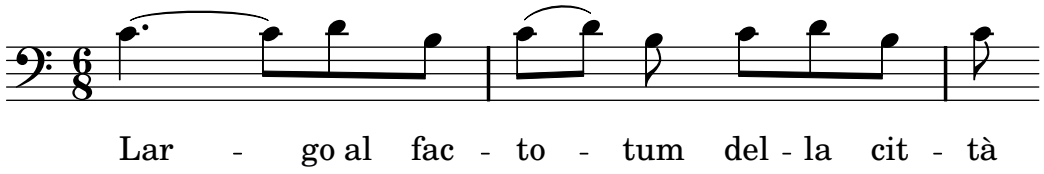
Keins der bisherigen Beispiele hat bisher Wörter benutzt, die länger als eine Silbe waren. Solche Wörter werden üblicherweise auf die Noten aufgeteilt, eine Silbe pro Note, mit Bindestrichen zwischen den Silben. Diese Silben werden durch zwei Minuszeichen notiert und von LilyPond als ein zentrierter Bindestrich zwischen den Silben gesetzt. Hier ein Beispiel, das dies und alle anderen Tricks zeigt, mit denen Text an den Noten ausgerichtet werden kann:

```
<<
  \relative c' {
    \key g \major
    \time 3/4
    \partial 4
    d4 g4 g a8( b) g4 g4
    b8( c) d4 d e4 c2
  }
  \addlyrics {
    A -- way in a __ man -- ger,
    no __ crib for a bed, __
  }
>>
```



Einige Texte, besonders in italienischer Sprache, brauchen das Gegenteil: mehr als eine Silbe muss zu einer einzelnen Note gesetzt werden. Das ist möglich, indem die Silben durch einen einzelnen Unterstrich _ zusammengekoppelt werden. Dazwischen dürfen sich keine Leerzeichen befinden, oder indem man die relevanten Silben in Anführungszeichen " setzt. Hier ein Beispiel aus dem *Figaro* von Rossini, wo die Silbe *al* auf der selben Note wie *go* des Wortes *Largo* in Figaros Arie *Largo al factotum* gesungen werden mus.

```
<<
  \relative c' {
    \clef bass
    \key c \major
    \time 6/8
    c4.~ c8 d b c([ d]) b c d b c
  }
  \addlyrics {
    Lar -- go_al fac -- to -- tum del -- la cit -- tà
  }
>>
```
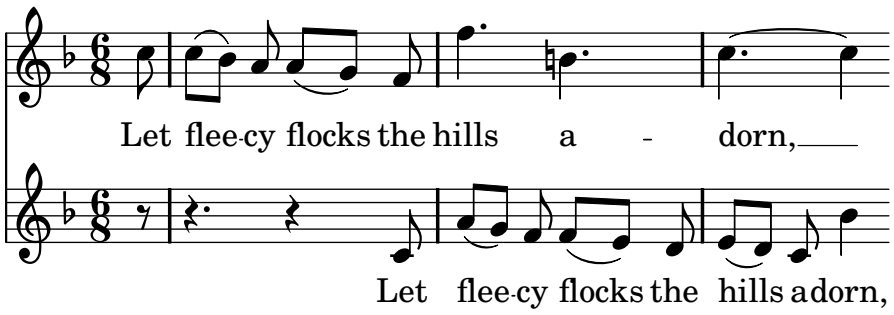


Siehe auch

Notationsreferenz: Abschnitt “Notation von Gesang” in *Notationsreferenz*.

2.4.3 Text zu mehreren Systemen

Die einfache Lösung mit `\addlyrics` kann benutzt werden, um Text zu einem oder mehreren Systemen zu setzen. Hier ein Beispiel aus Händels *Judas Maccabäus*:

```
<<
  \relative c'' {
    \key f \major
    \time 6/8
    \partial 8
    c8 c([ bes]) a a([ g]) f f'4. b, c4.~ c4
  }
  \addlyrics {
    Let flee -- cy flocks the hills a -- dorn, __
  }
  \relative c' {
    \key f \major
    \time 6/8
    \partial 8
    r8 r4. r4 c8 a'([ g]) f f([ e]) d e([ d]) c bes'4
  }
  \addlyrics {
    Let flee -- cy flocks the hills a -- dorn,
  }
>>
```



Aber Partituren, die komplizierter als dieses Beispiel sind, werden besser notiert, indem man die Systemstruktur von den Noten und dem Gesangstext durch Variablen trennt. Die Benutzung von Variablen wird erklärt im Abschnitt Abschnitt 2.5.1 [Stücke durch Bezeichner organisieren], Seite 36.

Siehe auch

Notationsreferenz: Abschnitt “Notation von Gesang” in *Notationsreferenz*.

2.5 Letzter Schliff

Das ist das letzte Kapitel der Übung. Hier soll demonstriert werden, wie man den letzten Schliff an einfachen Stücken anbringen kann. Gleichzeitig dient es als Einleitung zum Rest des Handbuches.

2.5.1 Stücke durch Bezeichner organisieren

Wenn alle die Elemente, die angesprochen wurden, zu größeren Dateien zusammengefügt werden, werden auch die musikalischen Ausdrücke sehr viel größer. In polyphonen Dateien mit vielen
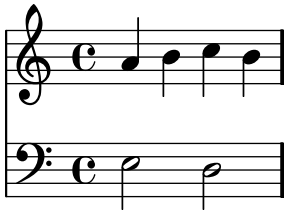
Systemen kann das sehr chaotisch aussehen. Das Chaos kann aber deutlich reduziert werden, wenn `Variablen` definiert und verwendet werden.

Variablen (die auch als Bezeichner oder Makros bezeichnet werden) können einen Teil der Musik aufnehmen. Sie werden wie folgt definiert:

```
bezeichneteMusik = { ... }
```

Der Inhalt des musikalischen Ausdrucks `bezeichneteMusik` kann dann später wieder benutzt werden, indem man einen Backslash davor setzt (`\bezeichneteMusik`), genau wie bei jedem LilyPond-Befehl.

```
Geige = \new Staff
  { \relative c'' {
    a4 b c b
  }
}
Cello = \new Staff
  { \relative c {
    \clef bass
    e2 d
  }
}
{
  <<
    \Geige
    \Cello
  >>
}
```
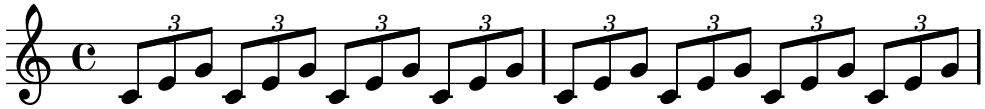


In den Namen der Variablen dürfen nur Buchstaben des Alphabets verwendet werden, keine Zahlen oder Striche.

Variable müssen *vor* dem eigentlichen musikalischen Ausdruck definiert werden. Sie können dann aber beliebig oft verwendet werden, nachdem sie einmal definiert worden sind. Sie können sogar eingesetzt werden, um später in der Datei eine neue Variable zu erstellen. Damit kann die Schreibarbeit erleichtert werden, wenn Notengruppen sich oft wiederholen.

```
trioleA = \times 2/3 { c,8 e g }
TaktA = { \trioleA \trioleA \trioleA \trioleA }

\relative c'' {
 \TaktA \TaktA
}
```



Man kann diese Variablen auch für viele andere Objekte verwenden, etwa:

```
Breite = 4.5\cm
Name = "Tim"
aFünfPapier = \paper { paperheight = 21.0 \cm }
```

Abhängig vom Kontext kann solch ein Bezeichner in verschiedenen Stellen verwendet werden. Das folgende Beispiel zeigt die Benutzung der eben definierten Bezeichner:

```
\paper {
  \aFünfPapier
  line-width = \width
}
{
  c4^\Name
}
```

2.5.2 Versionsnummer

Der `\version`-Befehl zeigt an, welche LilyPond-Version für eine bestimmte Quelldatei benutzt worden ist:

```
\version "2.12.1"
```

Üblicherweise wird dieser Befehl am Anfang der Textdatei eingefügt.

Durch diese Versionsmarkierung werden zukünftige Aktualisierungen des LilyPond-Programmes einfacher gemacht. Syntax-Änderungen zwischen den Programmversionen werden von einem speziellen Programm, `convert-ly`, vorgenommen. Dieses Programm braucht `\version`, um zu entscheiden, welche Regeln angewandt werden müssen. Für Einzelheiten, siehe Abschnitt “Dateien mit convert-ly aktualisieren” in *Anwendungsbenutzung*.

2.5.3 Titel hinzufügen

Titel, Komponist, Opusnummern und ähnliche Information werden in einer `\header`-Umgebung eingefügt. Diese Umgebung befindet sich außerhalb der musikalischen Ausdrücke, meistens wird die `\header`-Umgebung direkt nach der Abschnitt 2.5.2 [Versionsnummer], Seite 38 eingefügt.

```
\version "2.12.1"
\header {
  title = "Symphony"
  composer = "Ich"
  opus = "Op. 9"
}

{
  ... Noten ...
}
```
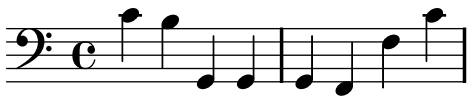
Wenn die Datei übersetzt wird, werden Titel- und Komponisteneinträge über der Musik ausgegeben. Mehr Information über die Titelei findet sich im Kapitel Abschnitt “Titel erstellen” in *Notationsreferenz*.

2.5.4 Absolute Notenbezeichnungen

Bis jetzt haben wir immer `\relative` benutzt, um Tonhöhen zu bestimmen. Das ist die einfachste Eingabeweise für die meiste Musik. Es gibt aber noch eine andere Möglichkeit, Tonhöhen darzustellen: durch absolute Bezeichnung.
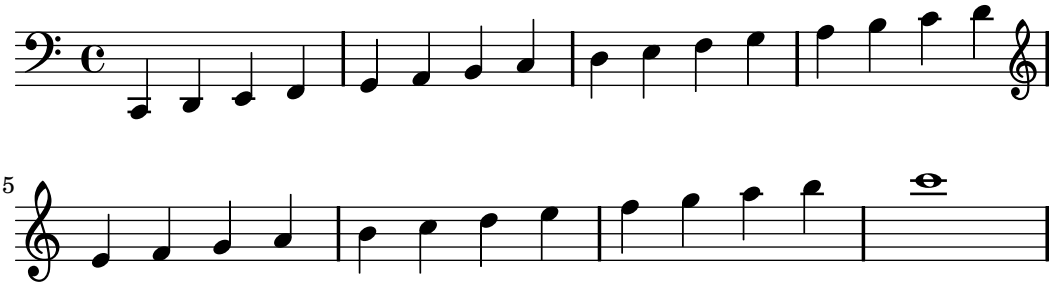
Wenn man das `\relative` weglässt, werden alle Tonhöhen von LilyPond als absolute Werte interpretiert. Ein `c'` ist dann also immer das eingestrichene C, ein `b` ist immer das kleine h unter dem eingestrichenen C, und ein `g,` ist immer das große G – also die Note auf der letzten Linie im Bass-Schlüssel.

```
{
  \clef bass
  c' b g, g,
  g, f, f c'
}
```



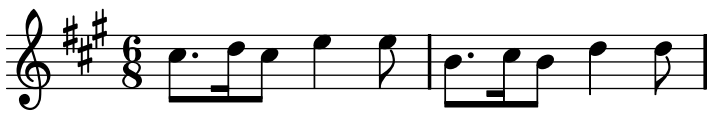
Hier eine Tonleiter über vier Oktaven:

```
{
  \clef bass
  c, d, e, f,
  g, a, b, c
  d e f g
  a b c' d'
  \clef treble
  e' f' g' a'
  b' c'' d'' e''
  f'' g'' a'' b''
  c'''1
}
```



Wie leicht zu sehen ist, muss man sehr viele Apostrophe schreiben, wenn die Melodie im Sopranschlüssel notiert ist. Siehe etwa dieses Fragment von Mozart:
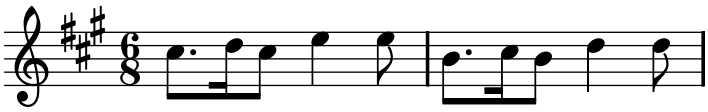
```
{
  \key a \major
  \time 6/8
  cis''8. d''16 cis''8 e''4 e''8
  b'8. cis''16 b'8 d''4 d''8
}
```



Alle diese Apostrophe machen den Quelltext schlecht lesbar und sind eine mögliche Fehlerquelle. Mit dem `\relative`-Befehl ist das Beispiel sehr viel einfacher zu lesen:

```
\relative c'' {
  \key a \major
  \time 6/8
```

```
  cis8. d16 cis8 e4 e8
  b8. cis16 b8 d4 d8
}
```



Wenn man einen Fehler durch ein Oktavierungszeichen (`'` oder `,`) im `\relative`-Modus macht, ist er sehr schnell zu finden, denn viele Noten sind nacheinander in der falschen Oktave. Im absoluten Modus dagegen ist ein einzelner Fehler nicht so deutlich und deshalb auch nicht so einfach zu finden.

Trotz allem ist der absolute Modus gut für Musik mit sehr großen Sprüngen und vor allem für computergenerierte LilyPond-Dateien.

2.5.5 Nach der Übung

Wenn Sie diese Übung absolviert haben, sollten Sie am besten ein paar Stücke selber notieren. Beginnen Sie mit den Anhang A [Vorlagen], Seite 150 und fügen Sie einfach Ihre Noten dazu. Wenn Sie irgendetwas brauchen, das nicht in der Übung besprochen wurde, schauen Sie sich den Abschnitt Alles über die Notation an, angefangen mit Abschnitt “Musikalische Notation” in *Notationsreferenz*. Wenn Sie für ein Instrument oder Ensemble Noten schreiben wollen, für das es keine Vorlage gibt, schauen Sie sich Abschnitt 3.4 [Erweiterung der Beispiele], Seite 75 an.

Wenn Sie ein paar kurze Stücke notiert haben, lesen Sie den Rest des Handbuchs zum Lernen (Kapitel 3–5). Natürlich können Sie auch sofort weiterlesen. Die nächsten Kapitel sind aber mit der Annahme geschrieben, dass Sie die Eingabesprache von LilyPond beherrschen. Sie können die weiteren Kapitel auch überfliegen und dann darauf wieder zurückkommen, wenn Sie einige Erfahrungen im Notieren gewonnen haben.

In dieser Übung, genauso wie im gesamten Handbuch zum Lernen, befindet sich ein Abschnitt **Siehe auch** am Ende jedes Abschnittes, wo sich Verweise auf andere Abschnitte befinden. Diesen Verweisen sollten Sie nicht beim ersten Durchlesen folgen; erst wenn Sie das gesamte Handbuch zum Lernen gelesen haben, können Sie bei Bedarf diesen Verweisen folgen, um ein Thema zu vertiefen.

Bitte lesen Sie jetzt Abschnitt 1.2 [Über die Dokumentation], Seite 9, wenn Sie es bisher noch nicht getan haben. Es gibt ungeheuer viel Information über LilyPond, so dass Neulinge sich nicht sofort zurecht finden. Wenn Sie auch nur ein paar Minuten in diesem Abschnitt lesen, können Sie sich Stunden frustrierendes Suchen an der falschen Stelle ersparen!

3 Grundbegriffe

Nachdem im Tutorial gezeigt wurde, wie aus einfachen Text-Dateien wunderschön formatierte Musiknoten erzeugt werden können, stellt dieses Kapitel die Konzepte und Techniken vor, wie auch komplexere Partituren erstellt werden können.

3.1 Wie eine LilyPond-Datei funktioniert

Das LilyPond Eingabeformat hat eine ziemlich freie Form, so dass für erfahrene Benutzer viel Freiheit besteht, die Struktur ihrer Quelldateien anzulegen. Für Neulinge kann diese Flexibilität aber erst einmal verwirrend sein. In diesem Kapitel soll darum ein Teil dieser Strukturen dargestellt werden, vieles aber zur Vereinfachung auch weggelassen werden. Für eine komplette Beschreibung des Eingabeformats siehe Abschnitt “Die Dateistruktur” in *Notationsreferenz*.

Die meisten Beispiele in diesem Handbuch sind kleine Schnipsel, wie etwa dieser:
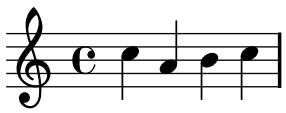
```
c4 a b c
```

Wie hoffentlich bekannt ist, lässt sich solch ein Schnipsel nicht in dieser Form übersetzen. Diese Beispiele sind also nur Kurzformen von wirklichen Beispielen. Sie müssen wenigstens zusätzlich in geschweifte Klammern gesetzt werden.

```
{
  c4 a b c
}
```

Die meisten Beispiele benutzen auch den `\relative c'`-Befehl. Der ist nicht nötig, um die Dateien zu übersetzen, aber in den meisten Fällen sieht der Notensatz seltsam aus, wenn man den Befehl weglässt.

```
\relative c'' {
  c4 a b c
}
```



Eine komplette Definition des Eingabeformats findet sich im Kapitel Abschnitt “Die Dateistruktur” in *Notationsreferenz*.

3.1.1 Einführung in die Dateistruktur von LilyPond

Ein grundlegendes Beispiel einer Eingabedatei für LilyPond lautet:

```
\version "2.12.1"
\header { }
\score {
  ...zusammengesetzter Musik-Ausdruck...  % Die gesamten Noten kommen hier hin!
  \layout { }
  \midi { }
}
```

Aufgrund der Flexibilität von LilyPond gibt es viele Variationen dieses Schemas, aber dieses Beispiel dient als einfacher Ausgangspunkt.

Bisher hat noch keines der Beispiele den `\score{}`-Befehl benutzt, da Lilypond derartige zusätzliche Befehle automatisch bei Bedarf einfügt, wenn die Eingabedatei eine einfache Struktur hat.

Sehen wir uns als ein solches einfaches Beispiel an:

```
\relative c'' {
  c4 a d c
}
```

Im Hintergrund kommen hier noch einige Ebenen dazu: LilyPond-Code in der obigen Form ist in Wirklichkeit eine Abkürzung. Auch wenn man so Dateien schreiben kann und sie auch korrekt gesetzt werden, heißt der vollständige Code, der hier gemeint ist, eigentlich:

```
\book {
  \score {
    \new Staff {
      \new Voice {
        \relative c'' {
          c4 a b c
        }
      }
    }
    \layout { }
  }
}
```

Mit anderen Worten: Wenn die Eingabedatei einen einfachen Musik-Ausdruck enthält, wird LilyPond die Datei so interpretieren, als ob dieser Ausdruck in den oben gezeigten Befehlen eingegeben wurde. Diese nötige Stuktur wird automatisch im Speicher beim Aufruf von LilyPond erzeugt, ohne dass der Benutzer davon etwas bemerkt.

Ein Wort der Warnung ist jedoch angebracht! Viele der Beispiele in der Dokumentation von LilyPond lassen die `\new Staff` und `\new Voice` Befehle zur Erzeugung einer Notenzeile und einer Stimme (beides ist in LilyPond ein sogenannter Kontext) bewusst aus, damit sie implizit von LilyPond im Speicher erzeugt werden. Für einfache Dokumente funktioniert das im Allgemeinen sehr gut, für komplexere Partituren können dadurch aber unerwartete Ergebnisse entstehen, teilweise sogar unerwartete leere Notenzeilen. Um die entsprechenden Kontexte in diesem Fall explizit zu erzeugen, siehe Abschnitt 3.3 [Kontexte und Engraver], Seite 64.

> **Achtung:** Wenn mehr als ein paar Zeilen an Musik eingegeben werden, empfielt es sich, die Notenzeilen und die Stimmen immer explizit mit `\new Staff` und `\new Voice` zu erzeugen.

Im Moment wollen wir aber zu unserem ersten Beispiel zurückkehren und nur den `\score`-Befehl näher betrachten.

Eine Partitur (`\score`) muss immer mit einem musikalischen Ausdruck beginnen. Das ist letztendlich alle Musik, angefangen bei einer einzelnen Note bis hin zu einer riesigen Partitur mit vielen Notensystemen (bezeichnet durch `GrandStaff`):

```
{
  \new GrandStaff <<
    ...hier die gesamte Partitur...
  >>
}
```

Da sich alles innerhalb der geschweiften Klammern `{ ... }` befindet, wird es wie ein einziger musikalischer Ausdruck behandelt.

Ein `\score` auch andere Dinge enthalten, wie etwa

```
\score {
  { c'4 a b c' }
  \layout { }
```

```
  \midi { }
  \header { }
}
```

Wie man sieht, sind die drei Befehle `\header`, `\layout` und `\midi` von spezieller Natur: Im Gegensatz zu vielen Anderen Befehlen, die auch mit einem `\` beginnen, liefern sie *keinen* Musikausdruck und sind auch nicht Teil eines musikalischen Ausdrucks. Daher können sie sowohl innerhalb eines `\score`-Blocks als auch außerhalb plaziert werden. Tatsächlich werden einige dieser Befehle meist außerhalb des `\score`-Blocksgesetzt, zum Beispiel findet sich der `\header` sehr oft oberhalb der `\score`-Umgebung. Das funktioniert genauso gut.

Zwei bisher noch nicht aufgetauchte Befehle sind `\layout { }` und `\midi {}`. Wenn sie in einer Datei vorkommen, führt dies dazu, dass Lilypond eine druckfähige PDF-Datei bzw. eine MIDI-Datei erzeugt. Genauer beschrieben werden sie im Benutzerhandbuch – Abschnitt “Partiturlayout” in *Notationsreferenz* und Abschnitt “MIDI-Dateien erstellen” in *Notationsreferenz*.

Ihr LilyPond Code kann auch mehrere `\score`-Blöcke enthalten. Jeder davon wird als eigenständige Partitur interpretiert, die allerdings alle in dieselbe Ausgabedatei plaziert weden. Ein `\book`-Befehl ist nicht explizit notwendig – er wird implizit erzeugt. Wenn jedoch für jeden `\score`-Block in einer einzigen `.ly`-Datei eine eigene Ausgabe-Datei erzeugt werden soll, dann muss jeder dieser Blöcke in einen eigenen `\book`-Block gesetzt werden: Jeder `\book`-Block erzeugt dann eine eigene Ausgabedatei.

Zusammenfassung:

Jeder `\book`-Block erzeugt eine eigene Ausgabedatei (z.B. eine PDF-Datei). Wenn Sie keinen derartigen Block explizit angegeben haben, setzt LilyPond den gesamten Dateiinhalt innerhalb eines einzigen impliziten `\book`-Blocks.

Jeder `\score`-Block beschreibt ein eigenständiges Musikstück innerhalb des `\book`-Blocks.

Jeder `\layout`-Block wirkt sich auf den `\score`- oder `\book`-Block aus, in dem er auftritt. So wirkt z.B. ein `\layout`-Block innerhalb eines `\score`-Blocks nur auf diesen einen Block und seinen gesamten Inhalt, ein `\layout`-Block außerhalb eines `\score`-Blocks (und daher innerhalb des implizit erzeugten oder explizit angegebenen `\book`-Blocks) jedoch auf alle `\score`-Blocks innerhalb dieses `\book`-Blocks.

Nähere Details finden sich im Abschnitt Abschnitt “Mehrere Partituren in einem Buch” in *Notationsreferenz*.

Eine gute Möglichkeit zur Vereinfachung sind selbst definierte Variablen. Alle Vorlagen verwenden diese Möglichkeit.

```
melodie = \relative c' {
  c4 a b c
}

\score {
  { \melodie }
}
```

Wenn LilyPond diese Datei analysiert, nimmt es den Inhalt von `melodie` (alles nach dem Gleichheitszeichen) und fügt ihn immer dann ein, wenn ein `\melodie` vorkommt. Die Namen sind frei wählbar, die Variable kann genauso gut `melodie`, `GLOBAL`, `rechteHandklavier`, oder `foofoobarbaz` heißen. Für mehr Information siehe Abschnitt 5.1.4 [Tipparbeit sparen durch Bezeichner und Funktionen], Seite 141. Als Variablenname kann fast jeder beliebige Name benutzt werden, allerdings dürfen nur Buchstaben vorkommen (also keine Zahlen, Unterstriche, Sonderzeichen, etc.) und er darf nicht wie ein LilyPond-Befehl lauten. Die genauen Einschränkungen sind beschrieben in Abschnitt “Die Dateistruktur” in *Notationsreferenz*.

Siehe auch

Eine vollständige Definition des Eingabeformats findet sich in Abschnitt “Die Dateistruktur” in *Notationsreferenz*.

3.1.2 Score ist ein (einziger) zusammengesetzter musikalischer Ausdruck

Im vorigen Kapitel, Abschnitt 3.1.1 [Einführung in die Dateistruktur von LilyPond], Seite 41, wurde die allgemeine Struktur einer LilyPond-Quelldatei beschrieben. Aber anscheinend haben wir die wichtigste Frage ausgelassen, nämlich wie man herausfindet, was nach dem `\score` geschrieben werden soll.

In Wirklichkeit ist das aber gar kein Geheimnis. Diese Zeile ist die Antwort:

> *Eine Partitur fängt immer mit* `\score` *an, gefolgt von einem einzelnen musikalischen Ausdruck.*

Vielleicht wollen Sie noch einmal Abschnitt 2.3.1 [Musikalische Ausdrücke erklärt], Seite 26 überfliegen. In diesem Kapitel wurde gezeigt, wie sich große musikalische Ausdrücke aus kleinen Teilen zusammensetzen. Noten können zu Akkorden verbunden werden usw. Jetzt gehen wir aber in die andere Richtung und betrachten, wie sich ein großer musikalischer Ausdruck zerlegen lässt.

```
\score {
  {   % diese Klammer startet den großen mus. Ausdruck
    \new StaffGroup <<
      ...hier eine ganze Wagner-Oper einfügen...
    >>
  }   % diese Klammer beendet den Ausdruck
  \layout { }
}
```

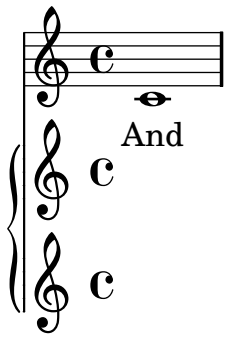
Eine Wagner-Oper ist mindestens doppelt so lang wie dieses Handbuch, beschränken wir uns also auf einen Sänger und Klavier. Wir brauchen keine ganze Orchesterpartitur, infolgedessen können wir die Systemgruppe (StaffGroup) auslassen, aber wir brauchen einen Sänger und ein Klavier.

```
\score {
  {
    <<
      \new Staff = "Sänger" <<
      >>
      \new PianoStaff = "Klavier" <<
      >>
    >>
  }
  \layout { }
}
```

Zur Erinnerung: mit `<<` und `>>` werden Noten gleichzeitig gesetzt; wir wollen ja auch Klavier- und Sängerstimme gleichzeitig und nicht hintereinander haben. Bei genauerem Hinsehen fällt auf, dass die `<< ... >>`-Konstruktion für die Notenzeile des Sängers eigentlich nicht unbedingt nötig wäre, da sie ja nur einen (sequenzielle) musikalischen Ausdruck enthält, nämlich alle Noten des Sängers hintereinander. Daher könnte an sich auch einfach ein `{...}` benutzt werden. Die Spitzklammern sind allerdings notwendig, sobald die Notenzeile mehrere parallelle Ausdrücke – wie etwa zwei parallele Stimmen oder eine Stimme mit zugehörigem Text – enthält. Wir werden die Musik später in das Beispiel einfügen, im Moment begnügen wir uns mit einigen Platzhalter-Noten und -Texten.

```
\score {
```

```
  <<
    \new Staff = "Sänger" <<
      \new Voice = "Singstimme" { c'1 }
      \addlyrics { And }
    >>
    \new PianoStaff = "Klavier" <<
      \new Staff = "oben" { }
      \new Staff = "unten" { }
    >>
  >>
  \layout { }
}
```



Jetzt haben wir viel mehr Details. Wir haben ein System (engl. staff) für einen Sänger, in dem sich wieder eine Stimme (engl. voice) befindet. `Voice` bedeutet für LilyPond eine Stimme (sowohl gesungen als auch gespielt) und evtl. zusätzlich einen Text. Zusätzlich werden zwei Notensysteme für das Klavier mit dem Befehl `\new PianoStaff` gesetzt. `PianoStaff` bezeichnet die Piano-Umgebung (etwa durchgehende Taktstriche und die geschweifte Klammer am Anfang), in der dann wiederum zwei eigene Systeme (`"oben"` für die rechte Hand und `"unten"` für die linke) erstellt werden.

Jetzt könnte man in diese Umgebung Noten einfügen. Innerhalb der geschweiften Klammern neben `\new Voice = "Singstimme"` könnte man

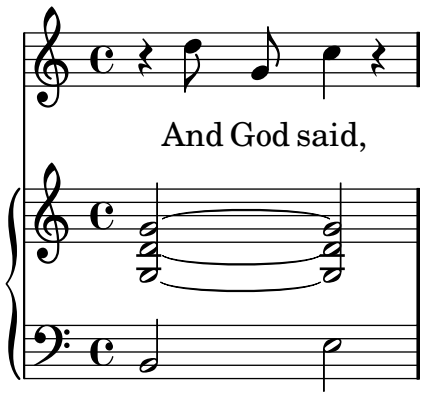
```
\relative c'' {
  r4 d8\noBeam g, c4 r
}
```

schreiben. Aber wenn man seine Datei so direkt schreibt, wird der `\score`-Abschnitt sehr lang und es wird ziemlich schwer zu verstehen, wie alles zusammenhängt. Darum bietet es sich an, Bezeichner (oder Variablen) zu verwenden.

```
melodie = \relative c'' { r4 d8\noBeam g, c4 r }
Text    = \lyricmode { And God said, }
oben    = \relative c'' { <g d g,>2~ <g d g,> }
unten   = \relative c { b2 e2 }

\score {
  <<
    \new Staff = "Sänger" <<
      \new Voice = "Singstimme" { \melodie }
      \addlyrics { \Text }
    >>
    \new PianoStaff = "Klavier" <<
```

```
      \new Staff = "oben" { \oben }
      \new Staff = "unten" {
        \clef "bass"
        \unten
      }
    >>
  >>
  \layout { }
}
```



Achten Sie auf den Unterschied zwischen Noten, die mit `\relative` oder direkt in einem musikalischen Ausruck eingegeben werden, und dem Text des Lieds, der innerhalb `\lyricmode` angegeben werden muss. Diese Unterscheidung ist für LilyPond essentiell, um zu entscheiden, ob der folgende Inhalt als Musik oder Text interpretiert werden soll. Wie könnte LilyPond sonst entscheiden, ob `{a b c}` die drei Noten a, b und c darstellen soll oder den Text eines Lieds über das Alphabet!

Beim Schreiben (oder Lesen) einer `\score`-Umgebung sollte man langsam und sorgfältig vorgehen. Am besten fängt man mit dem größten Gebilde an und definiert dann die darin enthaltenen kleineren der Reihe nach. Es hilft auch, sehr genau mit den Einzügen zu sein, so dass jede Zeile, die der gleichen Ebene angehört, wirklich horizontal an der gleichen Stelle beginnt.

Siehe auch
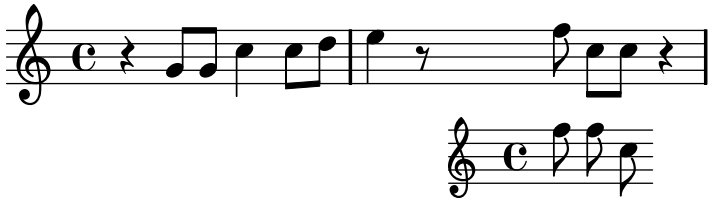
Benutzerhandbuch: Abschnitt “Struktur einer Partitur” in *Notationsreferenz*.

3.1.3 Musikalische Ausdrücke ineinander verschachteln

Notenzeilen (die ‚Staff‘-Kontexte) müssen nicht unbedingt gleich zu Beginn erzeugt werden – sie können auch zu einem späteren Zeitpunkt eingeführt weden. Das ist vor allem nützlichm um Abschnitt “Ossias” in *Glossar* zu erzeugen. Hier folgt ein kures Beispiel, wie eine zusätzliche temporäre Notenzeile für nur drei Noten erzeugt werden kann:

```
\new Staff {
  \relative g' {
    r4 g8 g c4 c8 d |
    e4 r8
    <<
      { f c c }
      \new Staff {
        f8 f c
      }
```
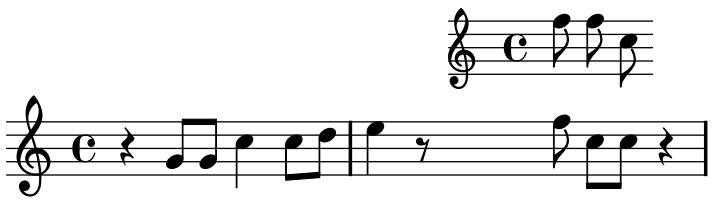
```
    >>
    r4 |
  }
}
```



Wie man sieht, ist die Größe des Notenschlüssels dieselbe, wie sie auch bei einer Schlüsseländerung auftritt – etwas kleiner als der Schlüssel am Beginn einer Notenzeile. Dies ist normal für Notenschlüssel, die innerhalb einer Notenzeile gesetzt werden.

Der Ossia-Abschnitt kann auch oberhalb der Hauptnotenzeile gesetzt werden:

```
\new Staff = "Hauptzeile" {
  \relative g' {
    r4 g8 g c4 c8 d |
    e4 r8
    <<
      { f c c }
      \new Staff \with {
        alignAboveContext = #"Hauptzeile" }
      { f8 f c }
    >>
    r4 |
  }
}
```



Dieses Beispiel benutzt den `\with`-Befehl, der später noch genauer erklärt wird. Damit kann das Standardverhalten einer einzelnen Notenzeile geändert werden: Hier wird einfach angegeben, dass die neue Notenzeile oberhalb der bereits existierenden Zeile mit Namen „Hauptzeile“ plaziert werden soll, anstatt standardmäßig unterhalb.

Siehe auch

Ossia werden oft ohne Notenschlüssel und Taktangabe gedruck, meist auch etwas kleiner als die anderen Notenzeilen. Dies ist natürlich auch in LilyPond möglich, benötigt aber Befehle, die bisher noch nicht vorgestellt wurden. Siehe Abschnitt 4.3.2 [Größe von Objekten], Seite 102 und Abschnitt “Ossia-Systeme” in *Notationsreferenz*.

3.1.4 Über die Nicht-Schachtelung von Klammern und Bindebögen

Sie haben bisher zahlreiche verschiedene Arten von Klammern beim Schreiben von Musik mit LilyPond kennengelernt. Diese folgen verschiedenen Regeln, die zu Beginn vielleicht verwirrend

wirken. Bevor die genauen Regeln vorgestellt werden, wollen wir die diversen Klammerarten kurz rekapitulieren:
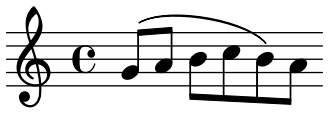
Klammerart	**Funktion**
`{ .. }`	Umschließt ein sequenzielles Musiksegment
`< .. >`	Umschließt die Noten eines Akkords
`<< .. >>`	Umschließt parallele Musikausdrücke
`( .. )`	Markiert den Beginn und das Ende eines Haltebogens
`\( .. \)`	Markiert den Beginn und das Ende eines Phasierungsbogens
`[ .. ]`	Markiert den Beginn und das Ende eines manuell erzeugten Balkens

Zusätzlich sollten vielleicht noch einige weiter Konstruktionen erwähnt werden, die Noten auf irgendeine Art und Weise verbinden: Haltebögen (durch eine Tilde `~` markiert), Triolen (als `\times x/y {..}` geschrieben) und Vorschlagnoten (als `\grace{..}` notiert).

Außerhalb von LilyPond fordert die übliche Benutzung von Klammern, dass die entsprechenden Arten korrekt verschachtelt werden, wie z.B. in `<< [ { ( .. ) } ] >>`. Die schließenden Klammern kommen dabei in der umgekehrten Reihenfolge wie die öffnenden Klammern vor. Dies ist auch in LilyPond ein **Muss** für die drei Klammerarten, die in obiger Tabelle mit dem Wort ‚Umschließt' beschrieben werden – sie müssen korrekt geschachtelt werden. Die restlichen Klammerarten (durch ‚Markiert' beschrieben), die Haltebögen und die Triolen brauchen jedoch mit den anderen Klammerarten **nicht** unbedingt korrekt geschachtelt werden. Tatsächlich sind sie auch keine Klammern in dem Sinn, dass sie etwas umschließen, sondern viel mehr Indikatoren, an welcher Stelle ein bestimmtes musikalisches Objekt beginnt oder endet.

So kann also z.B. einen Phrasierungsbogen vor einem manuellen Balken beginn, jedoch schon vor dem Ende des Balkens enden. Dies mag zwar musikalisch wenig Sinn ergeben, ist aber in LilyPond auch möglich:

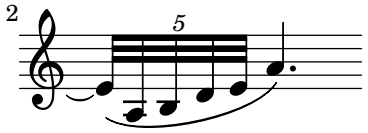
```
{ g8\( a b[ c b\) a] }
```



Im Allgemeinen können die verschiedenen Klammerarten, Haltebögen, Triolen und Vorschlagnoten beliebig kombiniert werden. Das folgende Beispiel zeigt einen Balken, der in eine Triole reicht (Zeile 1), eine Bindebogen, der ebenfalls in eine Triole reicht (Zeile 2), einen Balken und einen Bindeboden in eine Triole, ein Haltebogen, der über zwei Triolen läuft, sowie einen Phrasierungsbogen, der in einer Triole beginnt (Zeilen 3 und 4).

```
{
  r16[ g16 \times 2/3 {r16 e'8] }
  g16( a \times 2/3 {b d) e' }
  g8[( a \times 2/3 {b d') e'~]}
  \times 4/5 {e'32\( a b d' e'} a'4.\)
}
```





3.2 Voice enthält Noten

Sänger brauchen Stimmen zum Singen, und LilyPond braucht sie auch: in der Tat sind alle Noten für alle Instrumente in einer Partitur innerhalb von Stimmen gesetzt. Die Stimme ist das grundlegendste Prinzip von LilyPond.
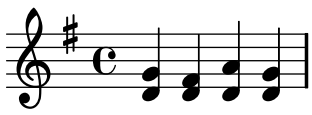
3.2.1 Ich höre Stimmen

Die grundegendsten und innersten Ebenen ein einer LilyPond-Partitur werden „Voice context“ (Stimmenkontext) oder auch nur „Voice“ (Stimme) genannt. Stimmen werden in anderen Notationsprogrammen manchmal auch als „layer“ (Ebene) bezeichnet.

Tatsächlich ist die Stimmenebene die einzige, die wirklich Noten enthalten kann. Wenn kein Stimmenkontext explizit erstellt wird, wird er automatisch erstellt, wie am Anfang dieses Kapitels gezeigt. Manche Instrumente wie etwa die Oboe können nur eine Note gleichzeitig spielen. Noten für solche Instrumente sind monophon und brauchen nur eine einzige Stimme. Instrumente, die mehrere Noten gleichzeitig spielen können, wie das Klavier, brauchen dagegeben oft mehrere Stimmen, um die verschiedenen gleichzeitig erklingenden Noten mit oft unterschiedlichen Rhythmen darstellen zu können.

Eine einzelne Stimme kann natürlich auch vielen Noten in einem Akkord enhalten – wann also braucht man dann mehrere Stimmen? Schauen wir uns zuerst dieses Beispiel mit vier Akkorden an:

```
\key g \major
<d g>4 <d fis> <d a'> <d g>
```
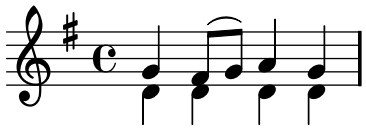


Das kann ausgedrückt werden, indem man die einfachen spitzen Klammern `< ... >` benützt, um Akkorde anzuzeigen. Hierfür braucht man nur eine Stimme. Aber gesetzt der Fall das Fis sollte eigentlich eine Achtelnote sein, gefolgt von einer Achtelnote G (als Durchgangsnote hin zum A)? Hier haben wir also zwei Noten, die zur gleichen Zeit beginnen, aber unterschiedliche Dauern haben: die Viertelnote D und die Achtelnote Fis. Wie können sie notiert werden? Als Akkord kann man sie nicht schreiben, weil alle Noten in einem Akkord die gleiche Länge besitzen müssen. Sie können auch nicht als aufeinanderfolgende Noten geschrieben werden, denn sie beginnen ja zur selben Zeit. In diesem Fall also brauchen wir zwei Stimmen.

Wie aber wird das in der LilyPond-Syntax ausgedrückt?

Die einfachste Art, Fragmente mit mehr als einer Stimme auf einem System zu notieren, ist, die Stimmen nacheinander (jeweils mit den Klammern `{ ... }`) zu schreiben und dann mit den spitzen Klammern (`<<...>>`) simultan zu kombinieren. Die beiden Fragmente müssen zusätzlich noch mit zwei Backslash-Zeichen (`\\`) voneinander getrennt werden, damit sie als zwei unterschiedliche Stimmen erkannt werden. Ohne diese Trenner würden sie als eine einzige Stimme notiert werden. Diese Technik ist besonders dann angebracht, wenn es sich bei den Noten um hauptsächlich homophone Musik handelt, in der hier und da polyphone Stellen vorkommen.

So sieht es aus, wenn die Akkorde in zwei Stimmen aufgeteilt werden und zur Durchgangsnote noch ein Bogen hinzugefügt wird:
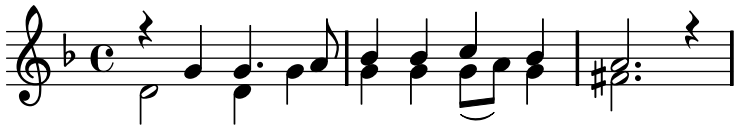
```
\key g \major
%    Voice "1"                  Voice "2"
<< { g4 fis8( g) a4 g }    \\ { d4 d d d }  >> |
```



Beachte, dass die Hälse der zweiten Stimme nun nach unten zeigen.
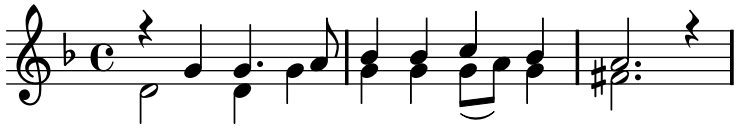
Hier ein anderes Beispiel:

```
\key d \minor
%    Voice "1"              Voice "2"
<< { r4 g g4. a8 }    \\ { d,2 d4 g }       >> |
<< { bes4 bes c bes } \\ { g4 g g8( a) g4 } >> |
<< { a2. r4 }         \\ { fis2. s4 }       >> |
```



Es ist nicht notwendig, für jeden Takt eine eigene << \\ >>-Konstruktion zu benutzen. Bei Musik mit nur wenigen Noten pro Takt kann es die Quelldatei besser lesbar machen, aber wenn in einem Takt viele Noten vorkommen, kann man die gesamten Stimmen separat schreiben, wie hier:

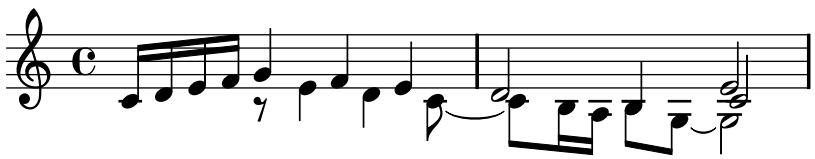
```
\key d \minor
<< {
  % Voice "1"
  r4 g g4. a8 |
  bes4 bes c bes |
  a2. r4 |
} \\ {
  % Voice "2"
  d,2 d4 g |
  g4 g g8( a) g4 |
  fis2. s4 |
} >>
```



Dieses Beispiel hat nur zwei Stimmen, aber die gleiche Konstruktion kann angewendet werden, wenn man drei oder mehr Stimmen hat, indem man weitere Backslash-Trenner hinzufügt.

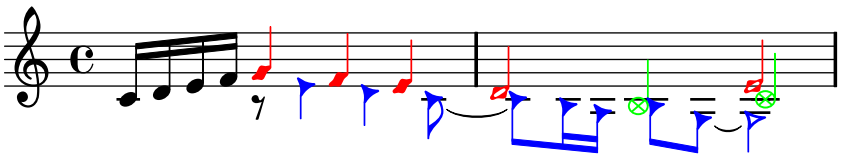
Die Stimmenkontexte tragen die Namen "1", "2" usw. In jedem dieser Kontexte wird die vertikale Ausrichtung von Hälsen, Bögen, Dynamik-Zeichen usw. entsprechend ausgerichtet.

```
\new Staff \relative c' {
  % Hauptstimme
  c16 d e f
  %    Voice "1"     Voice "2"                  Voice "3"
  << { g4 f e } \\ { r8 e4 d c8 ~ } >> |
  << { d2 e2 }  \\ { c8 b16 a b8 g ~ g2 } \\ { s4 b4 c2 } >> |
}
```



Diese Stimmen sind alle getrennt von der Hauptstimme, die die Noten außerhalb der `<< .. >>`-Konstruktion beinhaltet. Lassen wir es uns die *simultane Konstruktion* nennen. Bindebögen und Legatobögen können nur Noten in der selben Stimmen miteinander verbinden und können also somit nicht aus der simultanen Konstruktion hinausreichen. Umgekehrt gilt, dass parallele Stimmen aus eigenen simultanen Konstruktionen auf dem gleichen Notensystem die gleiche Stimme sind. Auch andere, mit dem Stimmenkontext verknüpfte Eigenschaften erstrecken sich auf alle simultanen Konstrukte. Hier das gleiche Beispiel, aber mit unterschiedlichen Farben für die Notenköpfe der unterschiedlichen Stimmen. Beachten Sie, dass Änderungen in einer Stimme sich nicht auf die anderen Stimmen erstrecken, aber sie sind weiterhin in der selben Stimme vorhanden, auch noch später im Stück. Beachten Sie auch, dass übergebundene Noten über die gleiche Stimme in zwei Konstrukten verteilt werden können, wie hier an der blauen Dreieckstimme gezeigt.

```
\new Staff \relative c' {
  % Hauptstimme
  c16 d e f
  << % Takt 1
    {
      \voiceOneStyle
      g4 f e
    }
  \\
    {
      \voiceTwoStyle
      r8 e4 d c8 ~
    }
  >>
  << % Takt 2
     % Stimme 1 geht weiter
    { d2 e2 }
  \\
     % Stimme 2 geht weiter
    { c8 b16 a b8 g ~ g2 }
  \\
    {
      \voiceThreeStyle
      s4 b4 c2
    }
  >>
}
```



Die Befehle `\voiceXXXStyle` sind vor allem dazu da, um in pädagogischen Dokumenten wie diesem hier angewandt zu werden. Sie verändern die Farbe des Notenkopfes, des Halses und des Balkens, und zusätzlich die Form des Notenkopfes, damit die einzelnen Stimmen einfach auseinander gehalten werden können. Die erste Stimme ist als rote Raute definiert, die zweite Stimme als blaue Dreiecke, die dritte Stimme als grüne Kreise mit Kreuz und die vierte Stimme (die hier nicht benutzt wird) hat dunkelrote Kreuze. `\voiceNeutralStyle` (hier auch nicht

benutzt) macht diese Änderungen rückgängig. Später soll gezeigt werden, wie Befehle wie diese vom Benutzer selber erstellt werden können. Siehe auch Abschnitt 4.3.1 [Sichtbarkeit und Farbe von Objekten], Seite 97 und Abschnitt 4.6.2 [Variablen für Optimierungen einsetzen], Seite 134.

Polyphonie ändert nicht die Verhältnisse der Noten innerhalb eines `\relative { }`-Blocks. Jede Note wird weiterhin relativ zu der vorherigen Note errechnet, oder relativ zur ersten Note des vorigen Akkords. So ist etwa hier
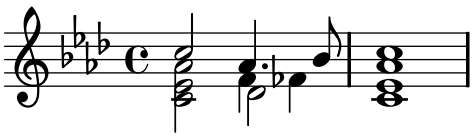
```
\relative c' { NoteA << < NoteB NoteC > \\ NoteD >> NoteE }
```

`NoteB` bezüglich `NoteA`
`NoteC` bezüglich `NoteB`, nicht `noteA`;
`NoteD` bezüglich `NoteB`, nicht `NoteA` oder `NoteC`;
`NoteE` bezüglich `NoteD`, nicht `NoteA` errechnet.

Eine andere Möglichkeit ist, den `\relative`-Befehl vor jede Stimme zu stellen. Das bietet sich an, wenn die Stimmen weit voneinander entfernt sind.

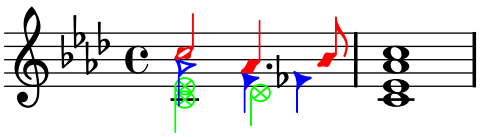
```
\relative c' { NoteA ... }
<<
  \relative c'' { < NoteB NoteC > ... }
\\
  \relative g' { NoteD ... }
>>
\relative c' { NoteE ... }
```

Zum Schluss wollen wir die Stimmen in einem etwas komplizierteren Stück analysieren. Hier die Noten der ersten zwei Takte von Chopins *Deux Nocturnes*, Op. 32. Dieses Beispiel soll später in diesem und dem nächsten Kapitel benutzt werden, um verschiedene Techniken, Notation zu erstellen, zu demonstrieren. Ignorieren Sie deshalb an diesem Punkt alles in folgendem Code, das Ihnen seltsam vorkommt, und konzentrieren Sie sich auf die Noten und die Stimmen. Die komplizierten Dinge werden in späteren Abschnitten erklärt werden.
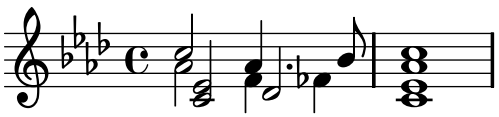


Die Richtung der Hälse wird oft benutzt, um anzuzeigen, dass zwei gleichzeitige Melodien sich fortsetzen. Hier zeigen die Hälse aller oberen Noten nach oben und die Hälse aller unteren Noten nach unten. Das ist der erste Anhaltspunkt, dass mehr als eine Stimme benötigt wird.

Aber die wirkliche Notwendigkeit für mehrere Stimmen tritt erst dann auf, wenn unterschiedliche Noten gleichzeitig erklingen, aber unterschiedliche Dauern besitzen. Schauen Sie sich die Noten auf dem dritten Schlag im ersten Takt an. Das As ist eine punktierte Viertel, das F ist eine Viertel und das Des eine Halbe. Sie können nicht als Akkord geschrieben werden, denn alle Noten in einem Akkord besitzen die gleiche Dauer. Sie können aber auch nicht nacheinander geschrieben werden, denn sie beginnen auf der gleichen Taktzeit. Dieser Taktabschnitt benötigt drei Stimmen, und normalerweise schreibt man drei Stimmen für den ganzen Takt, wie im Beispiel unten zu sehen ist; hier sind unterschiedliche Köpfe und Farben für die verschiedenen Stimmen eingesetzt. Nocheinmal: der Quellcode für dieses Beispiel wird später erklärt werden, deshalb ignorieren Sie alles, was Sie hier nicht verstehen können.
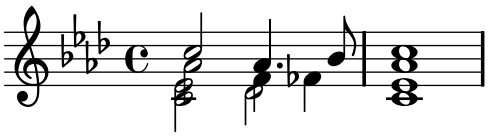


Versuchen wir also, diese Musik selber zu notieren. Wie wir sehen werden, beinhaltet das einige Schwierigkeiten. Fangen wir an, wie wir es gelernt haben, indem wir mit der << \\ >>-Konstruktion die drei Stimmen des ersten Taktes notieren:

```
\new Staff \relative c'' {
  \key aes \major
  <<
    { c2 aes4. bes8 } \\ { aes2 f4 fes } \\ { <ees c>2 des2 }
  >>
  <c ees aes c>1
}
```



Die Richtung des Notenhalses wird automatisch zugewiesen; die ungeraden Stimmen tragen Hälse nach oben, die gerade Hälse nach unten. Die Hälse für die Stimmen 1 und 2 stimmen, aber die Hälse in der dritten Stimme sollen in diesem Beispiel eigentlich nach unten zeigen. Wir können das korrigieren, indem wir die dritte Stimme einfach auslassen und die Noten in die vierte Stimme verschieben:

```
\new Staff \relative c'' {
  \key aes \major
  << % erste Stimme
    { c2 aes4. bes8 }
  \\ % zweite Stimme
    { aes2 f4 fes   }
  \\ % Stimme drei auslassen
  \\ % vierte Stimme
    { <ees c>2 des2 }
  >> |
  <c ees aes c>1 |
}
```



Wie zu sehen ist, ändert das die Richtung der Hälse, aber zeigt ein anderes Problem auf, auf das man manchmal bei mehreren Stimmen stößt: Die Hälse einer Stimme können mit den Hälsen anderer Stimmen kollidieren. LilyPond erlaubt Noten in verschiedenen Stimmen sich auf der gleichen vertikalen Position zu befinden, wenn die Hälse in entgegengesetzte Richtungen zeigen, und positioniert die dritte und vierte Stimme dann so, dass Zusammenstöße möglichst vermieden werden. Das funktioniert gewöhnlich recht gut, aber in diesem Beispiel sind die Noten der untersten Stimme eindeutig standardmäßig schlecht positioniert. LilyPond bietet verschiedene Möglichkeiten, die horizontale Position von Noten anzupassen. Wir sind aber noch nicht so weit, dass wir diese Funktionen anwenden könnten. Darum heben wir uns das Problem für einen späteren Abschnitt auf; siehe `force-hshift`-Eigenschaft in Abschnitt 4.5.2 [Überlappende Notation in Ordnung bringen], Seite 119.

Siehe auch

Notationsreferenz: Abschnitt “Mehrere Stimmen” in *Notationsreferenz*.

3.2.2 Stimmen explizit beginnen

Voice-Kontexte können auch manuell innerhalb eines << >>-Abschnittes initiiert werden. Mit den Befehlen \voiceOne bis hin zu \voiceFour kann jeder Stimme entsprechendes Verhalten von vertikaler Verschiebung und Richtung von Hälsen und anderen Objekten hinzugefügt werden. In längeren Partituren können die Stimmen damit besser auseinander gehalten werden.
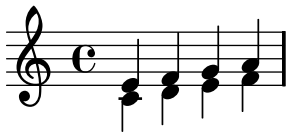
Die << \\ >>-Konstruktion, die wir im vorigen Abschnitt verwendet haben:

```
\new Staff {
  \relative c' {
    << { e4 f g a } \\ { c,4 d e f } >>
  }
}
```

ist identisch mit

```
\new Staff <<
  \new Voice = "1" { \voiceOne \relative c' { e4 f g a } }
  \new Voice = "2" { \voiceTwo \relative c' { c4 d e f } }
>>
```
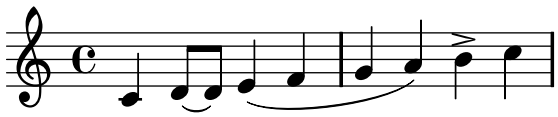
Beide würden folgendes Notenbild erzeugen:
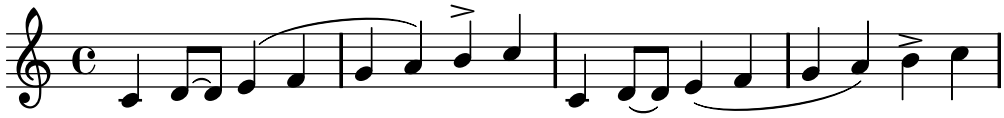


Der \voiceXXX-Befehl setzt die Richtung von Hälsen, Bögen, Artikulationszeichen, Text, Punktierungen und Fingersätzen. \voiceOne und \voiceThree lassen diese Objekte nach oben zeigen, \voiceTwo und \voiceFour dagegen lassen sie abwärts zeigen. Diese Befehle erzeugen eine horizontale Verschiebung, wenn es erforderlich ist, um Zusammenstöße zu vermeiden. Der Befehl \oneVoice stellt wieder auf das normale Verhalten um.

Schauen wir uns in einigen einfachen Beispielen an, was genau die Befehle \oneVoice, \voiceOne und voiceTwo mit Text, Bögen und Dynamikbezeichnung anstellen:

```
\relative c'{
  % Standard oder Verhalten nach \oneVoice
  c d8 ~ d e4 ( f g a ) b-> c
}
```
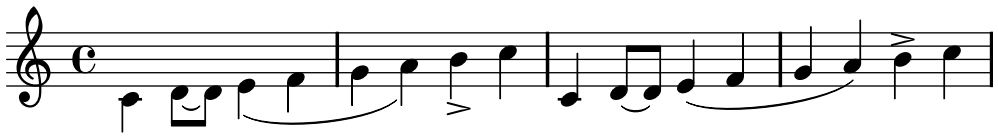


```
\relative c'{
  \voiceOne
  c d8 ~ d e4 ( f g a ) b-> c
  \oneVoice
  c, d8 ~ d e4 ( f g a ) b-> c
}
```



```
\relative c'{
```

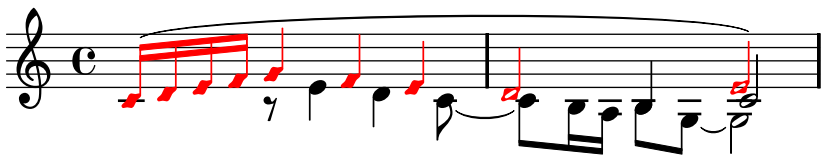
```
  \voiceTwo
  c d8 ~ d e4 ( f g a ) b-> c
  \oneVoice
  c, d8 ~ d e4 ( f g a ) b-> c
}
```



Schauen wir und nun drei unterschiedliche Arten an, den gleichen Abschnitt polyphoner Musik zu notieren, jede Art mit ihren Vorteilen in unterschiedlichen Situationen. Wir benutzen dabei das Beispiel vom vorherigen Abschnitt.

Ein Ausdruck, der direkt innerhalb einer << >>-Umgebung auftritt, gehört der Hauptstimme an. Das ist nützlich, wenn zusätzliche Stimme auftreten, während die Hauptstimme sich fortsetzt. Hier also eine bessere Version des Beispiels aus dem vorigen Abschnitt. Die farbigen Kreuz-Notenköpfe zeigen, dass die Hauptstimme sich jetzt in einem einzigen Stimmen (`voice`)-Kontext befindet. Somit kann ein Phrasierungsbogen ober sie gesetzt werden.

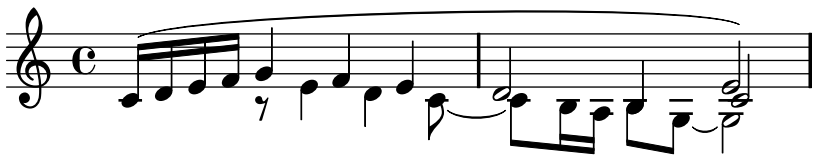
```
\new Staff \relative c' {
  \voiceOneStyle
  % Folgende Noten sind monophon
  c16^( d e f
  % Beginn von drei Stimmen gleichzeitig
  <<
    % Die Hauptstimme weiterlaufen lassen
    { g4 f e | d2 e2) }
    % Zweite Stimme einsetzen
    \new Voice {
      % Hälse usw. nach unten ausrichten
      \voiceTwo
      r8 e4 d c8 ~ | c8 b16 a b8 g ~ g2
    }
    % Die dritte Stimme beginnen
    \new Voice {
      % Hälse usw. nach oben ausrichten
      \voiceThree
      s2. | s4 b4 c2
    }
  >>
}
```



Tiefer verschachtelte polyphone Konstrukte sind möglich, und wenn eine Stimme nur kurz auftaucht, kann das der bessere Weg sein, Noten zu setzen:

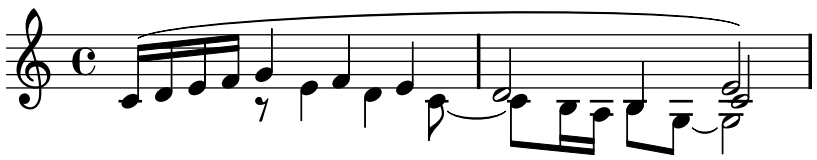
```
\new Staff \relative c' {
  c16^( d e f
```

```
  <<
    { g4 f e | d2 e2) }
    \new Voice {
      \voiceTwo
      r8 e4 d c8 ~ |
      <<
        {c8 b16 a b8 g ~ g2}
        \new Voice {
          \voiceThree
          s4 b4 c2
        }
      >>
    }
  >>
}
```



Diese Methode, neue Stimmen kurzzeitig zu verschachteln, bietet sich an, wenn nur sehr kleine Abschnitte polyphonisch gesetzt sind. Wenn aber die ganze Partitur polyphon ist, ist es meistens klarer, direkt unterschiedliche Stimmen über die gesamte Partitur hinweg einzusetzen. Hierbei kann man mit unsichtbaren Noten dann die Stellen überspringen, an denen die Stimme nicht auftaucht, wie etwa hier:

```
\new Staff \relative c' <<
  % Erste Stimme einrichten
  \new Voice {
    \voiceOne
    c16^( d e f g4 f e | d2 e2) |
  }
  % Zweite Stimme einsetzen
  \new Voice {
    % Hälse usw. nach unten ausrichten
    \voiceTwo
    s4 r8 e4 d c8 ~ | c8 b16 a b8 g ~ g2 |
  }
  % Die dritte Stimme beginnen
  \new Voice {
    % Hälse usw. nach oben ausrichten
    \voiceThree
    s1 | s4 b4 c2 |
  }
>>
```



Notenkolumnen

Dicht notierte Noten in einem Akkord, oder Noten auf der gleichen Taktzeit aber in unterschiedlichen Stimmen, werden in zwei, manchmal auch mehreren Kolumnen getzt, um die Noten am Überschneiden zu hindern. Wir bezeichnen sie als Notenkolumnen. Jede Stimme hat eine eigene Kolumne, und ein stimmenabhängiger Verschiebunsbefehl (engl. shift) wird eingesetzt, wenn eine Kollision auftreten könnte. Das zeigt das Beispiel oben. Im zweiten Takt wird das C der zweiten Stimme nach rechts verschoben, relativ gesehen zum D der ersten Stimme, und im letzten Akkord wird das C der dritten Stimme auch nach rechts verschoben im Verhältnis zu den anderen Stimmen.

Die Befehle `\shiftOn`, `\shiftOnn`, `\shiftOnnn` und `\shiftOff` bestimmen den Grad, zu dem Noten und Akkorde verschoben werden sollen, wenn sich sonst eine Kollision nicht vermeiden ließe. Die Standardeinstellung ist, dass die äußeren Stimmen (also normalerweise Stimme 1 und 2) `\shiftOff` eingestellt haben, während für die inneren Stimmen (3 und 4) `\shiftOn` eingeschaltet ist. Wenn eine Verschiebung auftritt, werden Stimmen 1 und 3 nach rechts und Stimmen 2 und 4 nach links verschoben.

`\shiftOnn` und `\shiftOnnn` definieren weitere Verschiebungsebenen, die man kurzzeitig anwählen kann, um Zusammenstöße in komplexen Situationen aufzulösen, siehe auch Abschnitt 4.5.3 [Beispiele aus dem Leben], Seite 124.

Eine Notenkolumne kann nur eine Note (oder einen Akkord) von einer Stimme mit Hälsen nach oben und eine Note (oder einen Akkord) von einer Stimme mit Hälsen nach unten tragen. Wenn Noten von zwei Stimmen mit den Hälsen in die gleiche Richtung an der selben Stelle auftreten und in beiden Stimmen ist keine Verschiebung oder die gleiche Verschiebungsebene definiert, wird die Fehlermeldung „zu viele kollidierende Notenspalten werden ignoriert“ ausgegeben.

Siehe auch
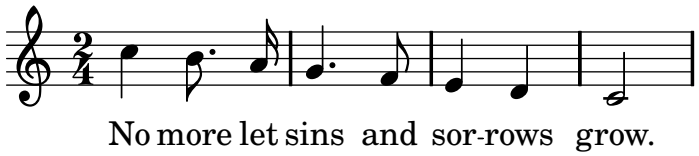
Notationsreferenz: Abschnitt “Mehrere Stimmen” in *Notationsreferenz*.

3.2.3 Stimmen und Text

Die Notation von Vokalmusik ihre eigene Schwierigkeit, nämlich die Kombination von zwei Ausdrücken: den Noten und dem Text. Achtung: Der Gesangstext wird auf Englisch „lyrics“ genannt.

Wir haben schon den `\addlyrics{}`-Befehl betrachtet, mit dem einfache Partituren gut erstellt werden können. Diese Methode ist jedoch recht eingeschränkt. Wenn der Notensatz komplexer wird, muss der Gesangstext mit einem neuen `Lyrics`-Kontext begonnen werden (mit dem Befehl `\new Lyrics`) und durch den Befehl `\lyricsto{}` mit einer bestimmten Stimme verknüpft werden, indem die Bezeichnung der Stimme benutzt wird.

```
<<
  \new Voice = "eins" \relative c'' {
    \autoBeamOff
    \time 2/4
    c4 b8. a16 g4. f8 e4 d c2
  }
  \new Lyrics \lyricsto "eins" {
    No more let sins and sor -- rows grow.
  }
>>
```
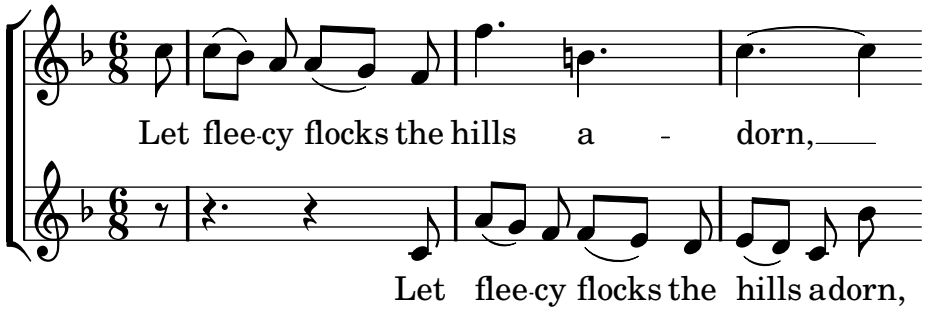


Beachten Sie, dass der Notentext nur mit einem `Voice`-Kontext verknüpft werden kann, nicht mit einem `Staff`-Kontext. In diesem Fall also müssen Sie ein System (`Staff`) und eine Stimme (`Voice`) explizit erstellen, damit alles funktioniert.

Die automatischen Balken, die LilyPond in der Standardeinstellung setzt, eignen sich sehr gut für instrumentale Musik, aber nicht so gut für Musik mit Text, wo man entweder gar keine Balken benutzt oder sie einsetzt, um Melismen zu verdeutlichen. Im Beispiel oben wird deshalb der Befehl `\autoBeamOff` eingesetzt um die automatischen Balken (engl. beam) auszuschalten.

Wir wollen das frühere Beispiel von *Judas Maccabæus* benutzen, um diese flexiblere Technik für Gesangstexte zu illustrieren. Das Beispiel wurde so umgeformt, dass jetzt Variablen eingesetzt werden, um den Text und die Noten von der Partiturstruktur zu trennen. Es wurde zusätzlich eine Chorpartiturklammer hinzugefügt. Der Gesangtext muss mit `\lyricmode` eingegeben werden, damit er als Text und nicht als Noten interpretiert werden kann.

```
global = { \time 6/8 \partial 8 \key f \major}
SoprEinsNoten = \relative c'' {
  c8 | c([ bes)] a a([ g)] f | f'4. b, | c4.~ c4 }
SoprZweiNoten = \relative c' {
  r8 | r4. r4 c8 | a'([ g)] f f([ e)] d | e([ d)] c bes' }
SopEinsText = \lyricmode {
  Let | flee -- cy flocks the | hills a -- dorn, __ }
SoprZweiText = \lyricmode {
  Let | flee -- cy flocks the | hills a -- dorn, }

\score {
  \new ChoirStaff <<
    \new Staff <<
      \new Voice = "SoprEins" {
        \global
        \SoprEinsNoten
      }
      \new Lyrics \lyricsto "SoprEins" {
        \SopEinsText
      }
    >>
    \new Staff <<
      \new Voice = "SoprZwei" {
        \global
        \SoprZweiNoten
      }
      \new Lyrics \lyricsto "SoprZwei" {
        \SoprZweiText
      }
    >>
  >>
}
```

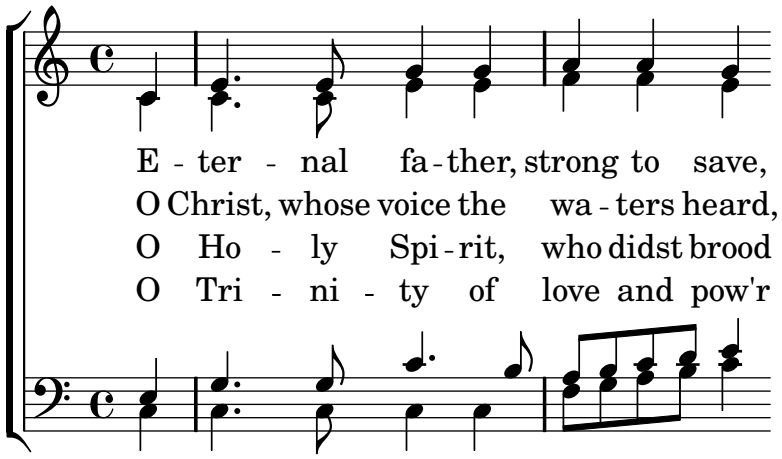


Dies ist die Grundstruktur für alle Chorpartituren. Mehr Systeme können hinzugefügt werden, wenn sie gebraucht werden, mehr Stimmen können zu jedem System hinzugefügt werden, mehr Strophen können zum Text hinzugefügt werden, und schließlich können die Variablen schnell in eine eigene Datei verschoben werden, wenn sie zu lang werden sollten.

Hier ein Beispiel der ersten Zeile eines Chorals mit vier Strophen für gemischten Chor. In diesem Fall ist der Text für alle vier Stimmen identisch. Beachten Sie, wie die Variablen eingesetzt werden, um Inhalt (Noten und Text) und Form (die Partitur) voneinander zu trennen. Eine Variable wurde eingesetzt, um die Elemente, die auf beiden Systemen auftauchen, aufzunehmen, nämlich Taktart und Tonart. Solch eine Variable wird oft auch mit „global“ bezeichnet.

```
Zeitangabe = { \time 4/4 \partial 4 \key c \major}
SoprNoten   = \relative c' { c4 | e4. e8 g4  g  | a a g }
AltNoten  = \relative c' { c4 | c4. c8 e4  e  | f f e }
TenorNoten = \relative c  { e4 | g4. g8 c4. b8 | a8 b c d e4 }
BassNoten  = \relative c  { c4 | c4. c8 c4  c  | f8 g a b c4 }
StropheEins   = \lyricmode {
  E -- | ter -- nal fa -- ther, | strong to save, }
StropheZwei   = \lyricmode {
  O | Christ, whose voice the | wa -- ters heard, }
StropheDrei = \lyricmode {
  O | Ho -- ly Spi -- rit, | who didst brood }
StropheVier  = \lyricmode {
  O | Tri -- ni -- ty of | love and pow'r }

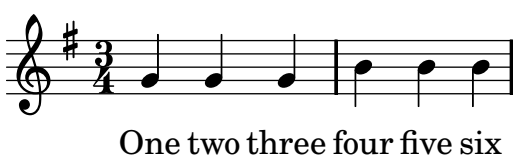
\score {
  \new ChoirStaff <<
    \new Staff <<
      \clef "treble"
      \new Voice = "Sopr"  { \voiceOne \Zeitangabe \SoprNoten }
      \new Voice = "Alt" { \voiceTwo \AltNoten }
      \new Lyrics \lyricsto "Sopr" { \StropheEins   }
      \new Lyrics \lyricsto "Sopr" { \StropheZwei   }
      \new Lyrics \lyricsto "Sopr" { \StropheDrei }
      \new Lyrics \lyricsto "Sopr" { \StropheVier  }
    >>
    \new Staff <<
      \clef "bass"
      \new Voice = "Tenor" { \voiceOne \Zeitangabe \TenorNoten }
      \new Voice = "Bass"  { \voiceTwo \BassNoten  }
    >>
  >>
}
```



Dieser Abschnitt schließt mit einem Beispiel, das eine Solo-Strophe mit anschließendem zweistimmigem Refrain auf zwei Systemen zeigt. Die Positionierung des einstimmigen Abschnitss und der mehrstimmigen Stelle ist etwas kompliziert; es braucht etwas Aufmerksamkeit, um der Erklärung folgen zu können.

Beginnen wir mit einer `score`-Umgebung, in der eine Chorpartitur (`ChoirStaff`) gesetzt wird. Die Partitur soll schließlich mit der eckigen Klammer beginnen. Normalerweise bräuchten wir spitze Klammern im Quelltext nach dem `\new ChoirStaff`, damit die Systeme paralell gesetzt werden, aber hier wollen wir diese Parallelsierung ja erst nach dem Solo. Also benutzen wir geschweifte Klammern. Innerhalb der Chorpartitur erstellen wir zuerst das System, das die Strophe enthält. Es braucht Noten und Text parallel, also setzen wir hier die spitzen Klammern um `\new Voice` und `\new Lyrics`.

```
StrophenNoten = \relative c'' {
  \clef "treble"
  \key g \major
  \time 3/4 g g g b b b
}
StrophenText = \lyricmode {
  One two three four five six
}
\score {
  \new ChoirStaff {
    \new Staff <<
      \new Voice = "Strophe" {
        \StrophenNoten \break
      }
      \new Lyrics \lyricsto Strophe {
        \StrophenText
      }
    >>
  }
}
```



Damit erhalten wir die Strophe.

Jetzt soll *refrainA* auf dem selben System gesetzt werden, während gleichzeitig in einem neuen System darunter *refrainB* gesetzt wird. Damit die Oberstimme das gleiche System benutzt,
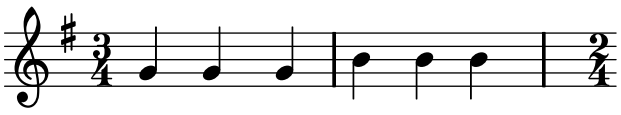
muss alles direkt auf den Zeilenumbruchbefehl (`\break` folgen, innerhalb der *verse*-Stimme. Ja, tatsächlich, *innerhalb* der *verse*-Stimme. Hier haben wir diese parallele Stelle isoliert. Weitere Systeme könnten auf die gleiche Weise hinzugefügt werden.
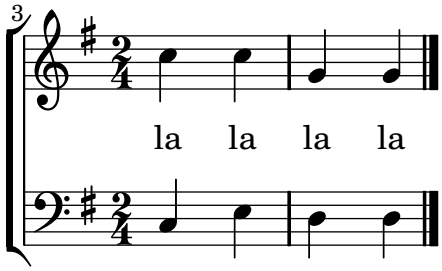
```
<<
  \refrainnotesA
  \new Lyrics \lyricsto verse {
    \refrainwordsA
  }
  \new Staff <<
    \new Voice = "refrainB" {
      \refrainnotesB
    }
    \new Lyrics \lyricsto "refrainB" {
      \refrainwordsB
    }
  >>
>>
```

Nun schließlich das Resultat mit zwei Systemen für den Refrain, man kann gut sehen, wie sich die parallele Stelle innherhalb der *verse*-Stimme befindet.

```
StrophenNoten = \relative c'' {
  \clef "treble"
  \key g \major
  \time 3/4 g g g b b b
}
RefrainNotenA = \relative c'' {
  \time 2/4
  c c g g \bar "|."
}
RefrainNotenB = \relative c {
  \clef "bass"
  \key g \major
  c e d d
}
StrophenText = \lyricmode {
  One two three four five six
}
RefrainTextA = \lyricmode {
  la la la la
}
RefrainTextB = \lyricmode {
  dum dum dum dum
}
\score {
  \new ChoirStaff {
    \new Staff <<
      \new Voice = "Strophe" {
        \StrophenNoten \break
        <<
          \RefrainNotenA
          \new Lyrics \lyricsto "Strophe" {
            \RefrainTextA
```

```
          }
          \new Staff <<
            \new Voice = "RefrainB" {
              \RefrainNotenB
            }
            \new Lyrics \lyricsto "RefrainB" {
              \RefrainTextB
            }
          >>
        >>
      }
      \new Lyrics \lyricsto "Strophe" {
        \StrophenText
      }
    >>
  }
}
```



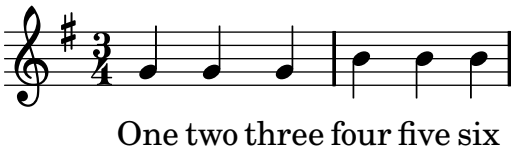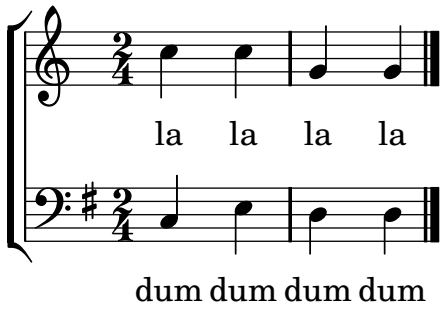


Dies ist zwar eine interessante und nützliche Übung um zu verstehen, wie sequentielle und parallele Notationsumgebungen funktionieren, in der Praxis würde man diesen Code aber vielleicht eher in zwei `\score`-Umgebungen trennnen und diese dann innerhalb einer `\book`-Umgebung einsetzen, wie im folgenden Beispiel demonstriert:

```
StrophenNoten = \relative c'' {
  \clef "treble"
  \key g \major
  \time 3/4 g g g b b b
}
RefrainNotenA = \relative c'' {
  \time 2/4
  c c g g \bar "|."
}
RefrainNotenB = \relative c {
  \clef "bass"
  \key g \major
  c e d d
}
```

```
StrophenText = \lyricmode {
  One two three four five six
}
RefrainTextA = \lyricmode {
  la la la la
}
RefrainTextB = \lyricmode {
  dum dum dum dum
}
\score {
  \new Staff <<
    \new Voice = "Strophe" {
      \StrophenNoten
    }
    \new Lyrics \lyricsto "Strophe" {
      \StrophenText
    }
  >>
}

\score {
  \new ChoirStaff <<
    \new Staff <<
      \new Voice = "RefrainA" {
        \RefrainNotenA
      }
      \new Lyrics \lyricsto "RefrainA" {
        \RefrainTextA
      }
    >>
    \new Staff <<
      \new Voice = "RefrainB" {
        \RefrainNotenB
      }
      \new Lyrics \lyricsto "RefrainB" {
        \RefrainTextB
      }
    >>
  >>
}
```





Siehe auch

Notation Reference: Abschnitt “Notation von Gesang” in *Notationsreferenz*.
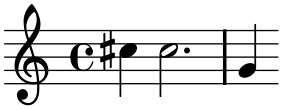
3.3 Kontexte und Engraver

Kontexte und Engraver („Stempel“) sind in den vorherigen Abschnitten schon aufgetaucht; hier wollen wir uns ihnen nun etwas ausführlicher widmen, denn sie sind sehr wichtig, um Feineinstellungen in der LilyPond-Notenausgabe vornehmen zu können.

3.3.1 Was sind Umgebungen?

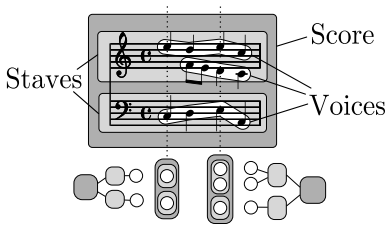
Wenn Noten gesetzt werden, müssen viele Elemente zu der Notenausgabe hinzugefügt werden, die im Quellcode gar nicht explizit vorkommen. Vergleichen Sie etwa den Quellcode und die Notenausgabe des folgenden Beispiels:

```
cis4 cis2. g4
```



Der Quellcode ist sehr kurz und knapp, während in der Notenausgabe Taktlinien, Vorzeichen, ein Schlüssel und eine Taktart hinzugefügt wurden. Während LilyPond den Eingabetext *interpretiert*, wird die musikalische Information in zeitlicher Reihenfolge inspiziert, etwa wie man eine Partitur von links nach rechts liest. Während das Programm den Code liest, merkt es sich, wo sich Taktgrenzen befinden und für welche Tonhöhen Versetzungszeichen gesetzt werden müssen. Diese Information muss auf mehreren Ebenen gehandhabt werden, denn Versetzungszeichen etwa beziehen sich nur auf ein System, Taktlinien dagegen üblicherweise auf die gesamte Partitur.

Innerhalb von LilyPond sind diese Regeln und Informationshappen in *Kontexten* (engl. contexts) gruppiert. Wir sind schon auf den `Voice` (Stimmen)-Kontext gestoßen. Daneben gibt es noch die `Staff` (Notensystem-) und `Score` (Partitur)-Kontexte. Kontexte sind hierarchisch geschichtet um die hierarchische Struktur einer Partitur zu spiegeln. Ein `Staff`-Kontext kann zum Beispiel viele `Voice`-Kontexte beinhalten, und ein `Score`-Kontext kann viele `Staff`-Kontexte beinhalten.



Jeder Kontext hat die Aufgabe, bestimmte Notationsregeln zu erzwingen, bestimmte Notationsobjekte zu erstellen und verbundene Elemente zu ordnen. Der `Voice`-Kontext zum Beispiel kann eine Vorzeichenregel einführen und der `Staff`-Kontext hält diese Regel dann aufrecht, um einzuordenen, ob ein Versetzungszeichen gesetzt werden muss oder nicht.

Ein anderes Beispiel: die Synchronisation der Taktlinien ist standardmäßig im `Score`-Kontext verankert. Manchmal sollen die Systeme einer Partitur aber unterschiedliche Taktarten enthalten, etwa in einer polymetrischen Partitur mit 4/4- und 3/4-Takt. In diesem Fall müssen also die Standardeinstellungen der `Score`- und `Staff`-Kontexte verändert werden.

In einfachen Partituren werden die Kontexte implizit erstellt, und es kann sein, dass Sie sich dessen gar nicht bewusst sind. Für etwas größere Projekte, etwa mit vielen Systemen, müssen die Kontexte aber explizit erstellt werden, um sicher zu gehen, dass man auch wirklich die erwünschte Zahl an Systemen in der richtigen Reihenfolge erhält. Wenn Stücke mit spezialisierter Notation gesetzt werden sollen, ist es üblich, die existierenden Kontexte zu verändern oder gar gänzlich neue zu definieren.

Zusätzlich zu den `Score, Staff` und `Voice`-Kontexten gibt es noch Kontexte, die zwischen der Partitur- und Systemebene liegen und Gruppen von Systemen kontrollieren. Das sind beispielsweise der `PianoStaff` und `ChoirStaff`-Kontext. Es gibt zusätzlich alternative Kontexte für Systeme und Stimmen sowie eigene Kontexte für Gesangstexte, Perkussion, Griffsymbole, Generalbass usw.

Die Bezeichnungen all dieser Kontexte werden von einem oder mehreren englischen Wörtern gebildet, dabei wird jedes Wort mit einem Großbuchstaben begonnen und direkt an das folgende ohne Bindestrich oder Unterstrich angeschlossen, etwa `GregorianTranscriptionStaff`.

Siehe auch

Notationreferenz: Abschnitt “Was sind Umgebungen?” in *Notationsreferenz*.

3.3.2 Umgebungen erstellen

Es gibt nur einen Kontext der obersten Ebene: der `Score`-Kontext. Er wird mit dem `\score`-Befehl, oder – in einfacheren Partituren – automatisch erstellt.

Wenn nur ein System vorhanden ist, kann man es ruhig LilyPond überlassen, die `Voice`- und `Staff`-Kontexte zu erstellen, aber für komplexere Partituren ist es notwendig, sie mit einem Befehl zu erstellen. Der einfachste Befehl hierzu ist `\new`. Er wird dem musikalischen Ausdruck vorangestellt, etwa so:

```
\new Typ musikalischer Ausdruck
```

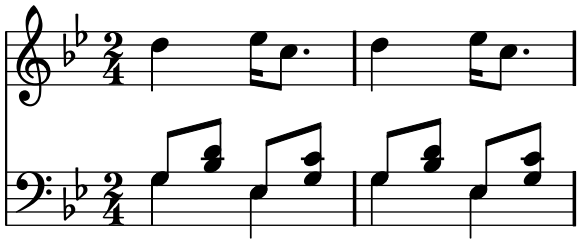
wobei *Typ* eine Kontextbezeichnung (wie etwa `Staff` oder `Voice`) ist. Dieser Befehl erstellt einen neuen Kontext und beginnt, den *muskalischen Ausdruck* innherhalb dieses Kontexts auszuwerten.

Beachten Sie, dass es keinen `\new Score`-Befehl gibt: der Partitur-Kontext der obersten Ebene wird mit dem Befehl `\score` begonnen.

Wir haben schon viele explizite Beispiel gesehen, in denen neue `Staff`- und `Voice`-Kontexte erstellt wurden, aber um noch einmal ins Gedächtnis zu rufen, wie diese Befehle benutzt werden, hier ein kommentiertes Beispiel aus dem richtigen Leben:

```
\score {  % Beginn des einen musikalischen Ausdrucks
  <<  % Beginn von gleichzeitigen Systemen
    \time 2/4
    \new Staff {  % RH-System erstellen
      \key g \minor
      \clef "treble"
      \new Voice {  % Stimme für RH Noten erstellen
        \relative c'' {  % Beginn von RH Noten
          d4 ees16 c8. |
          d4 ees16 c8. |
        }  % Ende RH-Noten
      }  % Ende der RH Stimme
```

```
    }  % Ende RH-System
    \new Staff <<  % LH System erstellen, braucht zwei gleichzeitige Stimmen
      \key g \minor
      \clef "bass"
      \new Voice {  % LH Stimme eins erstellen
        \voiceOne
        \relative g {  % Beginn von LH Stimme eins Noten
          g8 <bes d> ees, <g c> |
          g8 <bes d> ees, <g c> |
        }  % Ende von LH Stimme eins Noten
      }  % Ende LH Stimme eins
      \new Voice {  % LH Stimme zwei erstellen
        \voiceTwo
        \relative g {  % Beginn von LH Stimme zwei Noten
          g4 ees |
          g4 ees |
        }  % Ende der LH Stimme zwei Noten
      }  % Ende der LH Stimme zwei
    >>  % Ende LH System
  >>  % Ende der gleichzeitigen Systeme
}  % Ende des einen zusammengesetzten Musikausdrucks
```



(Beachten Sie, dass wir hier alle Zeilen, die eine neue Umgebung entweder mit einer geschweiften Klammer (`{`) oder doppelten spitzen Klammern (`<<`) öffnen, mit jeweils zwei Leerzeichen, und die entsprechenden schließenden Klammern mit der gleichen Anzahl Leerzeichen eingerückt werden. Dies ist nicht erforderlich, es wird aber zu einem großen Teil die nicht passenden Klammerpaar-Fehler eliminieren und ist darum sehr empfohlen. Es macht es möglich, die Struktur einer Partitur auf einen Blick zu verstehen, und alle nicht passenden Klammern erschließen sich schnell. Beachten Sie auch, dass das untere Notensystem mit eckigen Klammern erstellt wird, denn innerhalb dieses Systems brauchen wir zwei Stimmen, um die Noten darzustellen. Das obere System braucht nur einen einzigen musikalischen Ausdruck und ist deshalb von geschweiften Klammern umschlossen.)

Der `\new`-Befehl kann einem Kontext auch einen Namen zur Identifikation geben, um ihn von anderen Kontexten des selben Typs zu unterscheiden:

`\new` *`Typ = Name musikalischer Ausdruck`*

Beachten Sie den Unterschied zwischen der Bezeichnung des Kontexttyps (`Staff`, `Voice`, usw.) und dem Namen, der aus beliebigen Buchstaben bestehen kann und vom Benutzer frei erfunden werden kann. Zahlen und Leerzeichen können auch benutzt werden, in dem Fall muss der Name aber von doppelten Anführungszeichen umgeben werden, also etwa `\new Staff = "Mein System 1"` *`musikalischer Ausdruck`*. Der Name wird benutzt, um später auf genau diesen spezifischen Kontext zu verweisen. Dieses Vorgehen wurde schon in dem Abschnitt zu Gesangstexten angewandt, siehe Abschnitt 3.2.3 [Stimmen und Text], Seite 57.

Siehe auch

Notationsreferenz: Abschnitt “Umgebungen erstellen” in *Notationsreferenz*.

3.3.3 Was sind Engraver?

Jedes Zeichen des fertigen Notensatzes von LilyPond wird von einem `Engraver` (Stempel) produziert. Es gibt also einen Engraver, der die Systeme erstellt, einen, der die Notenköpfe ausgibt, einen für die Hälse, einen für die Balken usw. Insgesamt gibt es über 120 Engraver! Zum Glück braucht man für die meisten Partituren nur ein paar Engraver, und für einfache Partituren muss man eigentlich überhaupt nichts über sie wissen.

Engraver leben und wirken aus den Kontexten heraus. Engraver wie der `Metronome_mark_engraver`, dessen Aktion und Ausgabe sich auf die gesamte Partitur bezieht, wirken in der obersten Kontextebene – dem `Score`-Kontext.

Der `Clef_engraver` (Schlüssel-Stempel) und der `Key_engraver` (Vorzeichen-Stempel) finden sich in jedem `Staff`-Kontext, denn unterschiedliche Systeme könnten unterschiedliche Tonarten und Notenschlüssel brauchen.

Der `Note_heads_engraver` (Notenkopf-Stempel) und der `Stem_engraver` (Hals-Stempel) befinden sich in jedem `Voice`-Kontext, der untersten Kontextebene.

Jeder Engraver bearbeitet die bestimmten Objekte, die mit seiner Funktion assoziiert sind, und verwaltet die Eigenschaften dieser Funktion. Diese Eigenschaften, wie etwa die Eigenschaften, die mit Kontexten assoziiert sind, können verändert werden, um die Wirkungsweise des Engravers oder das Erscheinungsbild der von ihm produzierten Elemente in der Partitur zu ändern.

Alle Engraver haben zusammengesetzte Bezeichnung, die aus den (englischen) Wörtern ihrer Funktionsweise bestehen. Nur das erste Wort hat einen Großbuchstaben, und die restlichen Wörter werden mit einem Unterstrich angefügt. Ein `Staff_symbol_engraver` verantwortet also die Erstellung der Notenlinien, ein `Clef_engraver` entscheidet über die Art der Notenschlüssel und setzt die entsprechenden Symbole; damit wird gleichzeitig die Referenztonhöhe auf dem Notensystem festgelegt.

Hier die meistgebräuchlichen Engraver mit ihrer Funktion. Sie werden sehen, dass es mit etwas Englischkenntnissen einfach ist, die Funktion eines Engravers von seiner Bezeichnung abzuleiten.

Engraver	**Funktion**
Accidental_engraver	Erstellt Versetzungszeichen, vorgeschlagene und Warnversetzungszeichen.
Beam_engraver	Erstellt Balken.
Clef_engraver	Erstellt Notenschlüssel.
Completion_heads_engraver	Teilt Noten in kleiner Werte, wenn sie über die Taktlinie reichen.
New_dynamic_engraver	Erstellt Dynamik-Klammern und Dynamik-Texte.
Forbid_line_break_engraver	Verbietet Zeilenumbrüche, solange ein musikalisches Element aktiv ist.
Key_engraver	Erstellt die Vorzeichen.
Metronome_mark_engraver	Erstellt Metronom-Bezeichnungen.
Note_heads_engraver	Erstellt Notenköpfe.
Rest_engraver	Erstellt Pausen.
Staff_symbol_engraver	Ersetllt die (standardmäßig) fünf Notenlinien des Systems.
Stem_engraver	Erstellt die Notenhälse und Tremolos mit einem Hals.
Time_signature_engraver	Erstellt die Taktartbezeichnung.

Es soll später gezeigt werden, wie die LilyPond-Ausgabe verändert werden kann, indem die Wirkungsweise der Engraver beeinflusst wird.

Siehe auch

Referenz der Interna: Abschnitt “Engravers and Performers” in *Referenz der Interna*.

3.3.4 Kontexteigenschaften verändern

Kontexte sind dafür verantwortlich, die Werte bestimmter Kontext-*Eigenschaften* zu speichern. Viele davon können verändert werden, um die Interpretation der Eingabe zu beeinflussen und die Ausgabe zu verändern. Kontexte werden mit dem `\set`-Befehl geändert. Er wird in Form

```
\set KontextBezeichnung.eigenschaftsBezeichnung = #Wert
```

verwendet, wobei *KontextBezeichnung* üblicherweise `Score`, `Staff` oder `Voice` ist. Der erste Teil kann auch ausgelassen werden; in diesem Fall wird `Voice` eingesetzt.

Die Bezeichnung von Kontexten-Eigenschaften besteht aus zwei Wörtern, die ohne Unterstrich oder Bindestrich verbunden sind. Alle außer dem ersten werden am Anfang groß geschrieben. Hier einige Beispiele der gebräuchlichsten Kontext-Eigenschaften. Es gibt sehr viel mehr.

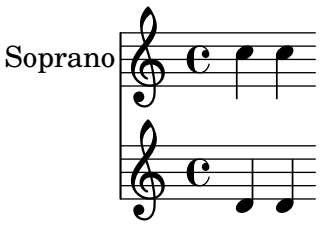
eigenschaftsBezeichnung	**Typ**	**Funktion**	**Beispiel-Wert**
extraNatural	boolescher Wert	Wenn wahr, werden zusätzliche Auflösungszeichen vor Versetzungszeichen gesetzt.	`#t`, `#f`
currentBarNumber	Integer	Setzt die aktuelle Taktnummer.	`50`
doubleSlurs	boolescher Wert	Wenn wahr, werden Legatobögen über und unter die Noten gesetzt.	`#t`, `#f`
instrumentName	Text	Setzt die Instrumentenbezeichnung am Anfang eines Systems.	`"Cello I"`
fontSize	reale Zahl	Vergrößert oder verkleinert die Schriftgröße.	`2.4`
stanza	Text	Setzt den Text zu Beginn einer Strophe.	`"2"`

Ein boolischer Wert ist entweder wahr (`#t`) oder falsch (`#f`), ein Integer eine positive ganze Zahl, eine reale Zahl eine positive oder negative Dezimalzahl, und Text wird in doppelte Anführungszeichen (Shift+2) eingeschlossen. Beachten Sie das Vorkommen des Rautenzeichens (`#`) an unterschiedlichen Stellen: als Teil eines booleschen Wertes vor dem `t` oder `f`, aber auch vor einem *Wert* in der `\set`-Befehlskette. Wenn ein boolescher Wert eingegeben werden soll, braucht man also zwei Rautenzeichen, z. B. `##t`.

Bevor eine Eigenschaft geändert werden kann, muss man wissen, in welchem Kontext sie sich befindet. Manchmal versteht das sich von selbst, aber in einigen Fällen kann es zunächst unverständlich erscheinen. Wenn der falsche Kontext angegeben wird, wird keine Fehlermeldung produziert, aber die Veränderung wird einfach nicht ausgeführt. `instrumentName` befindet sich offensichtlich innerhalb von einem `Staff`-Kontext, denn das Notensystem soll benannt werden. In dem folgenden Beispiel erhält das erste System korrekt die Instrumentenbezeichnung, das zweite aber nicht, weil der Kontext ausgelassen wurde.

```
<<
  \new Staff \relative c'' {
    \set Staff.instrumentName = #"Soprano"
    c4 c
 }
  \new Staff \relative c' {
  \set instrumentName = #"Alto"  % Falsch!
  d4 d
```
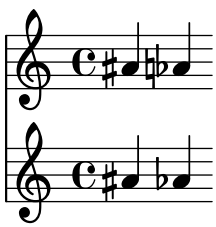
```
  }
>>
```



Denken Sie daran, dass der Standardkontext `Voice` ist; in dem zweiten `\set`-Befehl wird also die Eigenschaft `instrumentName` im `Voice`-Kontext auf „Alto“, gesetzt, aber weil LilyPond diese Eigenschaft nicht im `Voice`-Kontext vermutet, passiert einfach gar nichts. Das ist kein Fehler, und darum wird auch keine Fehlermeldung prodziert.

Ebenso gibt es keine Fehlermeldung, wenn die Kontext-Bezeichnung falsch geschrieben wird und die Änderung also nicht ausgeführt werden kann. Tatsächlich kann eine beliebige (ausgedachte) Kontextbezeichnung mit dem `\set`-Befehl eingesetzt werden, genauso wie die, die wirklich existieren. Aber wenn LilyPond diese Bezeichnung nicht zuordnen kann, bewirkt der Befehl einfach gar nichts. Manche Editoren, die Unterstützung für LilyPond-Befehle mitbringen, markieren existierende Kontextbezeichnungen mit einem Punkt, wenn man mit der Maus darüber fährt (wie etwa JEdit mit dem LilyPondTool), oder markieren unbekannte Bezeichnungen anders (wie ConTEXT). Wenn Sie keinen Editor mit LilyPond-Unterstützung einsetzen, wird empfohlen, die Bezeichnungen in der Interna-Referenz zu überprüfen: siehe Abschnitt “Tunable context properties” in *Referenz der Interna*, oder Abschnitt “Contexts” in *Referenz der Interna*.

Die Eigenschaft `instrumentName` wird erst aktiv, wenn sie ineinem `Staff`-Kontext gesetzt wird, aber manche Eigenschaften können in mehr als einem Kontext benutzt werden. Als Beispiel mag die `extraNatural`-Eigenschaft dienen, die zusätzliche Erniedrigungszeichen setzt. Die Standardeinstellung ist ##t (wahr) in allen Systemen. Wenn sie nur in einem `Staff` (Notensystem) auf ##f (falsch) gesetzt wird, wirkt sie sich auf alle Noten in diesem System aus. Wird sie dagegen in der `Score`-Umgebung gesetzt, wirkt sich das auf alle darin enthaltenen Systeme aus.
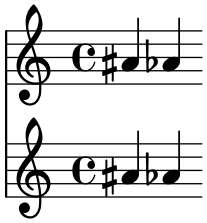
Das also bewirkt, dass die zusätzlichen Erniedrigungszeichen in einem System ausgeschaltet sind:

```
<<
  \new Staff \relative c'' {
    ais4 aes
 }
  \new Staff \relative c'' {
    \set Staff.extraNatural = ##f
    ais4 aes
 }
>>
```



während das dazu dient, sie in allen Systemen auszuschalten:

```
<<
  \new Staff \relative c'' {
    ais4 aes
 }
  \new Staff \relative c'' {
    \set Score.extraNatural = ##f
    ais4 aes
 }
>>
```
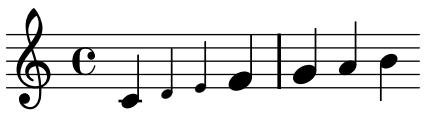


Ein anderes Beispiel ist die Eigenschaft `clefOctavation`: wenn sie im `Score`-Kontext gesetzt wird, ändert sich sofort der Wert der Oktavierung in allen aktuellen Systemen und wird auf einen neuen Wert gesetzt, der sich auf alle Systeme auswirkt.

Der gegenteilige Befehl, `\unset`, entfernt die Eigenschaft effektiv wieder von dem Kontext: in den meisten Fällen wird der Kontext auf ihre Standardeinstellungen zurückgesetzt. Normalerweise wird aber `\unset` nicht benötigt, denn ein neues `\set` erledigt alles, was man braucht.

Die `\set`- und `\unset`-Befehle könne überall im Eingabequelltext erscheinen und werden aktiv von dem Moment, an dem sie auftreten bis zum Ende der Partitur oder bis die Eigenschaft mit `\set` oder `\unset` neu gesetzt wird. Versuchen wir als Beispiel, die Schriftgröße mehrmals zu ändern, was sich unter anderem auf die Notenköpfe auswirkt. Die Änderung bezieht sich immer auf den Standard, nicht vom letzten gesetzten Wert.

```
c4
% Notenköpfe verkleinern
\set fontSize = #-4
d e
% Notenköpfe vergrößern
\set fontSize = #2.5
f g
% zur Standardgröße zurückgehen
\unset fontSize
a b
```



Wir haben jetzt gesehen, wie sich die Werte von unterschiedlichen Eigenschaften ändern lassen. Beachten Sie, dass Integre und Zahlen immer mit einem Rautenzeichen beginnen, während die Werte wahr und falsch (mit `##t` und `##f` notiert) immer mit zwei Rauten beginnen. Eine Eigenschaft, die aus Text besteht, muss in doppelte Anführungsstriche gesetzt werden, auch wenn wir später sehen werden, dass Text auf eine sehr viel allgmeinere und mächtigere Art mit dem `markup`-Befehl eingegeben werden kann.

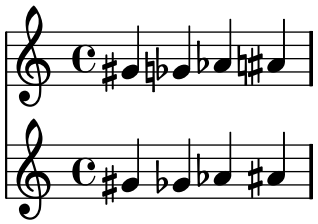
Kontexteigenschaften mit `\with` setzen

Kontexteigenschaften können auch gesetzt werden, wenn der Kontext erstellt wird. Das ist in manchen Fällen eine deutlichere Art, eine Eigenschaft zu bestimmen, die für die gesamte Partitur erhalten bleichen soll. Wenn ein Kontext mit einem `\new`-Befehl erstellt wird, dem direkt eine `\with { .. }`-Umgebung folgt, können hier die Eigenschaften bestimmt werden. Wenn also die zusätzlichen Auflösungszeichen für eine ganze Partitur gelten sollen, könnte man schreiben:

```
\new Staff \with { extraNatural = ##f }
```

etwa so:

```
<<
  \new Staff
  \relative c'' {
    gis ges aes ais
  }
  \new Staff \with { extraNatural = ##f }
  \relative c'' {
    gis ges aes ais
  }
>>
```



Eigenschaften, die auf diese Arte gesetzt werden, können immer noch dynamisch mit dem `\set`-Befehl geändert werden und mit `\unset` auf ihre Standardeinstellungen zurückgesetzt werden.
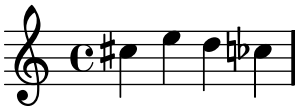
Die `fontSize`-Eigenschaft wird anders behandelt. Wenn sie mit einer `\with`-Umgebung gesetzt wird, wird die Standardschriftgröße neu gesetzt. Wenn die Schriftgröße später mit `\set` verändert wird, kann dieser neue Standardwert mit dem Befehl `\unset fontSize` erreicht werden.

Kontexteigenschaften mit `\context` setzen

Die Werte von Kontext-Eigenschaften können in *allen* Kontexten eines bestimmten Typs (etwa alle `Staff`-Kontexte) gleichzeitig mit einem Befehl gesetzt werden. Der Kontext wird spezifieziert, indem seine Bezeichnung benutzt wird, also etwa `Staff`, mit einem Backslash davor: `\Staff`. Der Befehl für die Eigenschaft ist der gleiche, wie er auch in der `\with`-Konstruktion benutzt wird, wie oben gezeigt. Er wird in eine `\context`-Umgebung eingebettet, welche wiederum innerhalb von einer `\layout`-Umgebung steht. Jede `\context`-Umgebung wirkt sich auf alle Kontexte dieses Types aus, welche sich in der aktuellen Partitur befinden (d. h. innherhalb einer `\score`- oder `\book`-Umgebung. Hier ist ein Beispiel, wie man diese Funktion anwendet:

```
\score {
  \new Staff {
    \relative c'' {
      cis4 e d ces
    }
  }
  \layout {
```

```
    \context {
      \Staff
      extraNatural = ##t
    }
  }
}
```



Kontext-Eigenschaften, die auf diese Weise gestzt werden, können für bestimmten Kontexte überschrieben werden, indem die `\with`-Konstruktion eingesetzt wird, oder mit `\set`-Befehlen innerhalb der aktuellen Noten.

Siehe auch

Notationsreferenz: Abschnitt “Die Standardeinstellungen von Umgebungen ändern” in *Notationsreferenz*.

Referenz der Interna: Abschnitt “Contexts” in *Referenz der Interna*, Abschnitt “Tunable context properties” in *Referenz der Interna*.

3.3.5 Engraver hinzufügen und entfernen

Wir haben gesehen, dass jeder Kontext eine Anzahl an Engravern (Stempeln) beinhaltet, von denen ein jeder einen bestimmten Teil des fertigen Notensatzes produziert, wie z. B. Taktlinien, Notenlinien, Notenköpfe, Hälse usw. Wenn ein Engraver aus einem Kontext entfernt wird, kann er seine Objekte nicht länger produzieren. Das ist eine eher grobe Methode, die Notenausgabe zu beeinflussen, aber es kann von großem Nutzen sein.

Einen einzelnen Kontext verändern

Um einen Engraver von einem einzelnen Kontext zu entfernen, wir der `\with`-Befehl eingesetzt, direkt hinter den Befehl zur Kontext-Erstellung geschrieben, wie in dem vorigen Abschnitt gezeigt.

Als ein Beispiel wollen wir das Beispiel aus dem letzten Abschnitt produzieren, aber die Notenlinien entfernen. Erinnern Sie sich, dass die Notenlinien vom `Staff_symbol_engraver` erstellt werden.

```
\new Staff \with {
  \remove Staff_symbol_engraver
}
\relative c' {
  c4
  \set fontSize = #-4  % Notenköpfe verkleinern
  d e
  \set fontSize = #2.5  % Notenköpfe vergrößern
  f g
  \unset fontSize  % zur Standardgröße zurückgehen
  a b
}
```
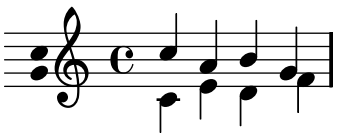


Engraver können auch zu einem bestimmten Kontext hinzugefügt werden. Dies geschieht mit dem Befehl

`\consists` *`Engraver_bezeichnung`*
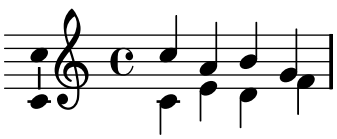
welcher auch wieder innerhalb der `\with`-Umgebung gesetzt wird. Einige Chorpartituren zeigen einen Ambitus direkt zu Beginn der ersten Notenzeile, um den Stimmumfang des Stückes anzuzeigen, siehe auch Abschnitt “Tonumfang” in *Glossar*. Der Ambitus wird vom `Ambitus_engraver` erstellt, der normalerweise in keinem Kontext enthalten ist. Wenn wir ihn zum `Voice`-Kontext hinzufügen, errechnet er automatisch den Stimmumfang für diese einzelne Stimme und zeigt ihn an:

```
\new Staff <<
  \new Voice \with {
    \consists Ambitus_engraver
  }
  \relative c'' {
    \voiceOne
    c a b g
  }
  \new Voice
  \relative c' {
    \voiceTwo
    c e d f
  }
>>
```



wenn wir den Ambitus-Engraver allerdings zum `Staff`-Kontext hinzufügen, wird der Stimmumfang aller Stimmen in diesem Notensystem errechnet:
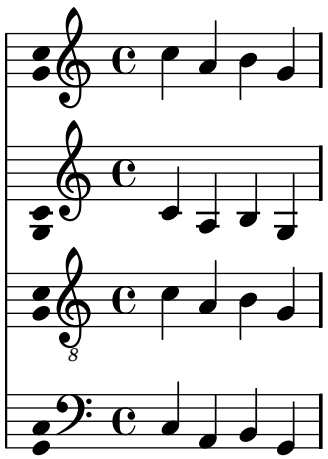
```
\new Staff \with {
    \consists Ambitus_engraver
  }
  <<
  \new Voice
  \relative c'' {
    \voiceOne
    c a b g
  }
  \new Voice
  \relative c' {
    \voiceTwo
    c e d f
  }
>>
```



Alle Kontexte des gleichen Typs verändern

Die vorigen Beispiele zeigen, wie man Engraver in einem bestimmten Kontext hinzufügen oder entfernen kann. Es ist auch möglich, Engraver in jedem Kontext eines bestimmten Typs hinzuzufügen oder zu entfernen. Dazu werden die Befehle in dem entsprechenden Kontext in einer `\layout`-Umgebung gesetzt. Wenn wir also z. B. den Ambitus für jedes Notensystem in einer Partitur mit vier Systemen anzeigen wollen, könnte das so aussehen:

```
\score {
  <<
    \new Staff <<
      \relative c'' { c a b g }
    >>
    \new Staff <<
      \relative c' { c a b g }
    >>
    \new Staff <<
      \clef "G_8"
      \relative c' { c a b g }
    >>
    \new Staff <<
      \clef "bass"
      \relative c { c a b g }
    >>
  >>
  \layout {
    \context {
      \Staff
      \consists Ambitus_engraver
    }
  }
}
```



Die Werte der Kontext-Eigenschaften können auch für alle Kontexte eines bestimmten Typs auf die gleiche Weise geändert werden, indem der `\set`-Befehl in einer `\context`-Umgebung angewendet wird.

Siehe auch

Notationsreferenz: Abschnitt “Umgebungs-Plugins verändern” in *Notationsreferenz*, Abschnitt “Die Standardeinstellungen von Umgebungen ändern” in *Notationsreferenz*.

3.4 Erweiterung der Beispiele

Sie haben sich durch die Übung gearbeitet, Sie wissen jetzt, wie Sie Notensatz produzieren, und Sie haben die grundlegenden Konzepte verstanden. Aber wie erhalten Sie genau die Systeme, die Sie brauchen? Es gibt eine ganze Anzahl an fertigen Vorlagen (siehe Anhang A [Vorlagen], Seite 150), mit denen Sie beginnen können. Aber was, wenn Sie nicht genau das finden, was Sie brauchen? Lesen Sie weiter.

3.4.1 Sopran und Cello

Beginnen Sie mit der Vorlage, die Ihren Vorstellungen am nächsten kommt. Nehmen wir einmal an, Sie wollen ein Stück für Sopran und Cello schreiben. In diesem Fall könnten Sie mit der Vorlage „Noten und Text“ (für die Sopran-Stimme) beginnen.

```
\version "2.12.1"
melody = \relative c' {
  \clef treble
  \key c \major
  \time 4/4
  a4 b c d
}

text = \lyricmode {
  Aaa Bee Cee Dee
}

\score {
  <<
    \new Voice = "one" {
      \autoBeamOff
      \melody
    }
    \new Lyrics \lyricsto "one" \text
  >>
  \layout { }
  \midi { }
}
```

Jetzt wollen wir die Cello-Stimme hinzufügen. Schauen wir uns das Beispiel „Nur Noten“ an:

```
\version "2.12.1"
melody = \relative c' {
  \clef treble
  \key c \major
  \time 4/4
  a4 b c d
}

\score {
  \new Staff \melody
  \layout { }
```

```
  \midi { }
}
```

Wir brauchen den `\version`-Befehl nicht zweimal. Wir brauchen aber den `melody`-Abschnitt. Wir wollen keine zwei `\score` (Partitur)-Abschnitte – mit zwei `\score`-Abschnitten würden wir zwei Stimmen getrennt voneinander erhalten. In diesem Fall wollen wir sie aber zusammen, als Duett. Schließlich brauchen wir innerhalb des `\score`-Abschnittes nur einmal die Befehle `\layout` und `\midi`.

Wenn wir jetzt einfach zwei `melody`-Abschnitte in unsere Datei kopieren würden, hätten wir zwei `melody`-Variable. Das würde zu keinem Fehler führen, aber die zweite von ihnen würde für beide Melodien eingesetzt werden. Wir müssen ihnen also andere Bezeichnungen zuweisen, um sie voneinander zu unterscheiden. Nennen wir die Abschnitte also `SopranNoten` für den Sopran und `CelloNoten` für die Cellostimme. Wenn wir schon dabei sind, können wir `text`auch nach `SoprText` umbenennen. Denken Sie daran, beide Vorkommen der Bezeichnung zu ändern: einmal die Definition gleich am Anfang (`melody = \relative c' {`) und dann auch noch die Benutzung der Variable innerhalb des `\score`-Abschnittes.

Gleichzeitig können wir auch noch das Notensystem für das Cello ändern – das Cello hat normalerweise einen Bassschlüssel. Wir ändern auch die Noten etwas ab.

```
\version "2.12.1"
SopranNoten = \relative c' {
  \clef treble
  \key c \major
  \time 4/4
  a4 b c d
}

SoprText = \lyricmode {
  Aaa Bee Cee Dee
}

CelloNoten = \relative c {
  \clef bass
  \key c \major
  \time 4/4
  d4 g fis8 e d4
}

\score {
  <<
    \new Voice = "eins" {
      \autoBeamOff
      \SopranNoten
    }
    \new Lyrics \lyricsto "eins" \Soprantext
  >>
  \layout { }
  \midi { }
}
```

Das sieht schon vielversprechend aus, aber die Cello-Stimme erscheint noch nicht im Notensatz – wir haben vergessen, sie in den `\score`-Abschnitt einzufügen. Wenn die Cello-Stimme unterhalb des Soprans erscheinen soll, müssen wir

```
\new Staff \CelloNoten
```

unter dem Befehl für den Sopran hinzufügen. Wir brauchen auch die spitzen Klammern (<< und >>) um die Noten, denn damit wird LilyPond mitgeteilt, dass mehr als ein Ereignis gleichzeitig stattfindet (in diesem Fall sind es zwei `Staff`-Instanzen). Der `\score`-Abschnitt sieht jetzt so aus:

```
\score {
  <<
  <<
    \new Voice = "eins" {
      \autoBeamOff
      \SopranNoten
    }
    \new Lyrics \lyricsto "eins" \SoprText
  >>
  \new Staff \CelloNoten
  >>
  \layout { }
  \midi { }
}
```

Das sieht etwas unschön aus, vor allem die Einrückung stimmt nicht mehr. Das können wir aber schnell in Ordnung bringen. Hier also die gesamte Vorlage für Sopran und Cello:

```
\version "2.12.1"

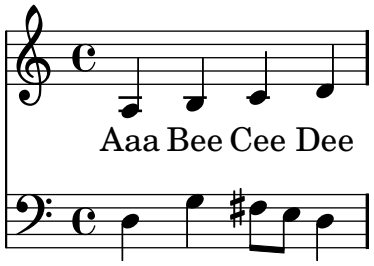
SopranNoten = \relative c' {
  \clef treble
  \key c \major
  \time 4/4
  a4 b c d
}

SoprText = \lyricmode {
  Aaa Bee Cee Dee
}

CelloNoten = \relative c {
  \clef bass
  \key c \major
  \time 4/4
  d4 g fis8 e d4
}

\score {
  <<
    <<
      \new Voice = "eins" {
        \autoBeamOff
        \SopranNoten
      }
      \new Lyrics \lyricsto "eins" \SoprText
    >>
    \new Staff \CelloNoten
  >>
  \layout { }
```

```
  \midi { }
}
```

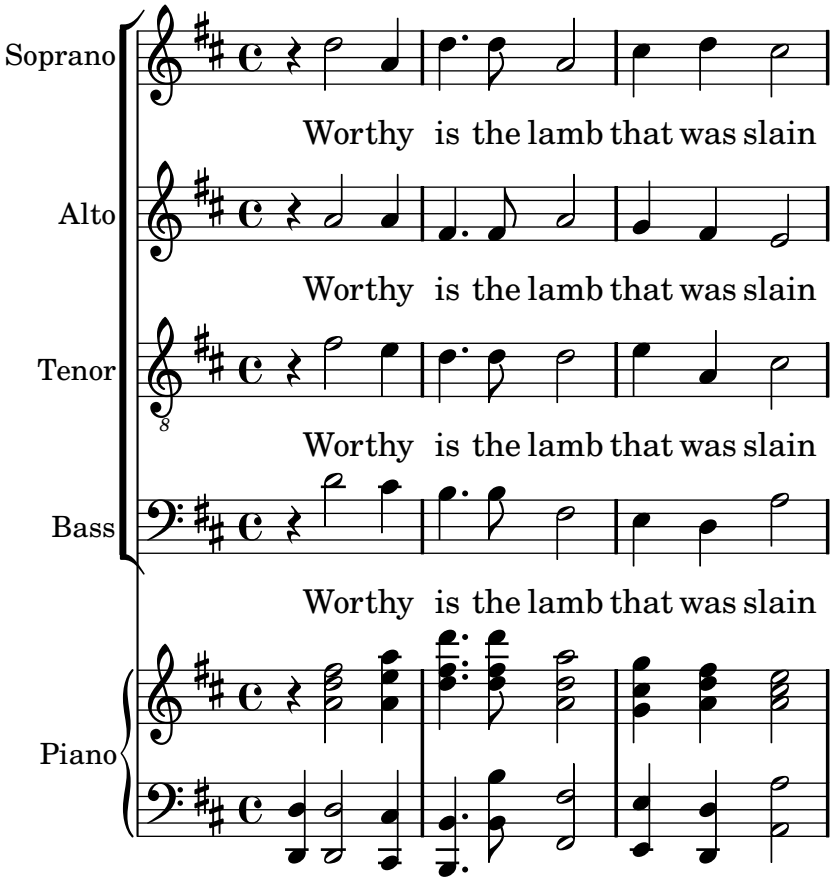


Siehe auch

Die Vorlagen, mit denen wir begonnen haben, können im Anhang „Vorlagen“ gefunden werden, siehe Abschnitt A.1 [Ein einzelnes System], Seite 150.

3.4.2 Vierstimmige SATB-Partitur

Die meisten Partituren für vierstimmigen gemischten Chor mit Orchesterbegleitung (wie etwa Mendelssohns *Elias* oder Händels *Messias*) sind so aufgebaut, dass für jede der vier Stimmen ein eigenes System besteht und die Orchesterbegleitung dann als Klavierauszug darunter notiert wird. Hier ein Beispiel aus Händels *Messias*:



Keine der Vorlage bietet diesen Aufbau direkt an. Die Vorlage, die am nächsten daran liegt, ist „SATB-Partitur und automatischer Klavierauszug“, siehe Abschnitt A.4 [Vokalensemble], Seite 160. Wir müssen diese Vorlage aber so anpassen, dass die Noten für das Klavier nicht

automatisch aus dem Chorsatz generiert werden. Die Variablen für die Noten und den Text des Chores sind in Ordnung, wir müssen nun noch Variablen für die Klaviernoten hinzufügen.

Die Reihenfolge, in welcher die Variablen in das Chorsystem (`ChoirStaff`) eingefügt werden, entspricht nicht der in dem Beispiel oben. Wir wollen sie so sortieren, dass die Texte jeder Stimme direkt unter den Noten notiert werden. Alle Stimmen sollten als `\voiceOne` notiert werden, welches die Standardeinstellung ist; wir können also die `\voiceXXX`-Befehle entfernen. Wir müssen auch noch den Schlüssel für den Tenor ändern. Die Methode, mit der der Text den Stimmen zugewiesen wird, ist uns noch nicht bekannt, darum wollen wir sie umändern auf die Weise, die wir schon kennen. Wir fügen auch noch Instrumentbezeichnungen zu den Systemen hinzu.

Damit erhalten wir folgenden `ChoirStaff`:

```
\new ChoirStaff <<
  \new Staff = "sopranos" <<
    \set Staff.instrumentName = #"Soprano"
    \new Voice = "sopranos" { \global \SopranNoten }
  >>
  \new Lyrics \lyricsto "sopranos" { \SopranText }
  \new Staff = "altos" <<
    \set Staff.instrumentName = #"Alto"
    \new Voice = "altos" { \global \AltNoten }
  >>
  \new Lyrics \lyricsto "altos" { \AltText }
  \new Staff = "tenors" <<
    \set Staff.instrumentName = #"Tenor"
    \new Voice = "tenors" { \global \TenorNoten }
  >>
  \new Lyrics \lyricsto "tenors" { \TenorText }
  \new Staff = "basses" <<
    \set Staff.instrumentName = #"Bass"
    \new Voice = "basses" { \global \BassNoten }
  >>
  \new Lyrics \lyricsto "basses" { \BassText }
>>  % Ende ChoirStaff
```

Als nächstes müssen wir das Klaviersystem bearbeiten. Das ist einfach: wir nehmen einfach den Klavierteil aus der „Piano solo“-Vorlage:

```
\new PianoStaff <<
  \set PianoStaff.instrumentName = #"Piano  "
  \new Staff = "oben" \oben
  \new Staff = "unten" \unten
>>
```

und fügen die Variablen `oben` und `unten` hinzu.

Das Chorsystem und das Pianosystem müssen mit spitzen Klammern kombiniert werden, damit beide übereinandern erscheinen:

```
<<  % ChoirStaff und PianoStaff parallel kombinieren
  \new ChoirStaff <<
    \new Staff = "sopranos" <<
      \new Voice = "sopranos" { \global \SopranNoten }
    >>
    \new Lyrics \lyricsto "sopranos" { \SopranText }
    \new Staff = "altos" <<
```

```
      \new Voice = "altos" { \global \AltNoten }
    >>
    \new Lyrics \lyricsto "altos" { \AltText }
    \new Staff = "tenors" <<
      \clef "G_8"  % Tenorschlüssel
      \new Voice = "tenors" { \global \TenorNoten }
    >>
    \new Lyrics \lyricsto "tenors" { \TenorText }
    \new Staff = "basses" <<
      \clef "bass"
      \new Voice = "basses" { \global \BassNoten }
    >>
    \new Lyrics \lyricsto "basses" { \BassText }
  >>  % Ende ChoirStaff

  \new PianoStaff <<
    \set PianoStaff.instrumentName = #"Piano"
    \new Staff = "oben" \oben
    \new Staff = "unten" \unten
  >>
>>
```

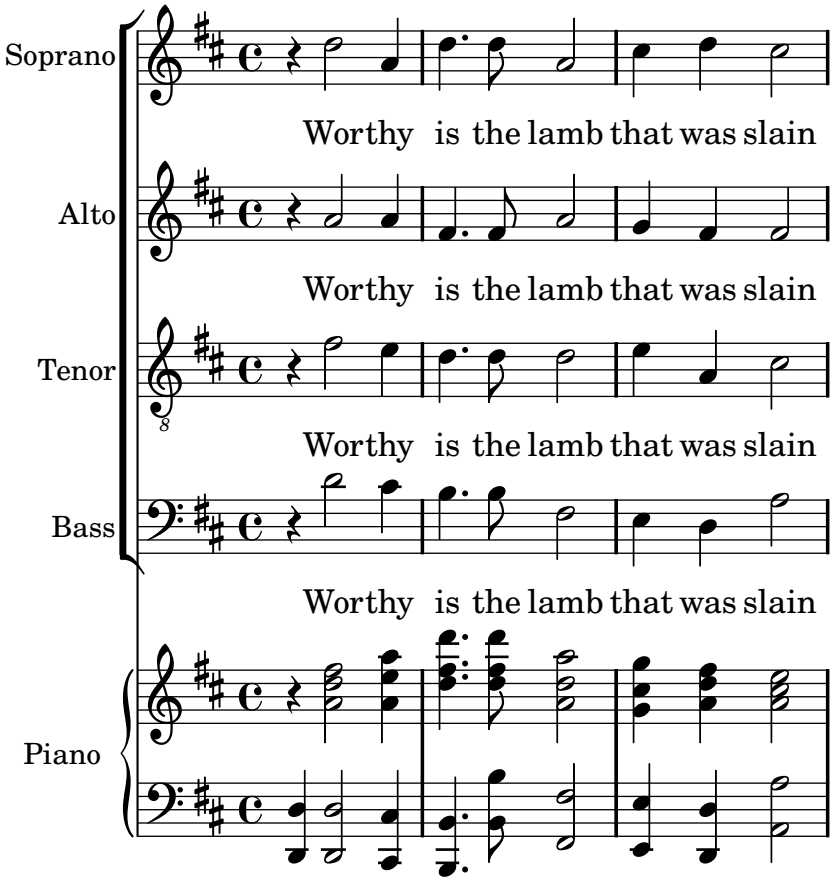
Alles miteinander kombiniert und mit den Noten für drei Takte sieht unser Beispiel nun so aus:

```
\version "2.12.1"
global = { \key d \major \time 4/4 }
SopranNoten = \relative c'' {
  \clef "treble"
  r4 d2 a4 | d4. d8 a2 | cis4 d cis2 |
}
SopranText = \lyricmode {
  Wor -- thy is the lamb that was slain
}
AltNoten = \relative a' {
  \clef "treble"
  r4 a2 a4 | fis4. fis8 a2 | g4 fis fis2 |
}
AltText = \SopranText
TenorNoten = \relative c' {
  \clef "G_8"
  r4 fis2 e4 | d4. d8 d2 | e4 a, cis2 |
}
TenorText = \SopranText
BassNoten = \relative c' {
  \clef "bass"
  r4 d2 cis4 | b4. b8 fis2 | e4 d a'2 |
}
BassText = \SopranText
oben = \relative a' {
  \clef "treble"
  \global
  r4 <a d fis>2 <a e' a>4 |
```

```
  <d fis d'>4. <d fis d'>8 <a d a'>2 |
  <g cis g'>4 <a d fis> <a cis e>2 |
}
unten = \relative c, {
  \clef "bass"
  \global
  <d d'>4 <d d'>2 <cis cis'>4 |
  <b b'>4. <b' b'>8 <fis fis'>2 |
  <e e'>4 <d d'> <a' a'>2 |
}

\score {
  <<  % ChoirStaff und PianoStaff parallel kombinieren
    \new ChoirStaff <<
      \new Staff = "Sopran" <<
        \set Staff.instrumentName = #"Soprano"
        \new Voice = "Sopran" { \global \SopranNoten }
      >>
      \new Lyrics \lyricsto "Sopran" { \SopranText }
      \new Staff = "Alt" <<
        \set Staff.instrumentName = #"Alto"
        \new Voice = "Alt" { \global \AltNoten }
      >>
      \new Lyrics \lyricsto "Alt" { \AltText }
      \new Staff = "Tenor" <<
        \set Staff.instrumentName = #"Tenor"
        \new Voice = "Tenor" { \global \TenorNoten }
      >>
      \new Lyrics \lyricsto "Tenor" { \TenorText }
      \new Staff = "Bass" <<
        \set Staff.instrumentName = #"Bass"
        \new Voice = "Bass" { \global \BassNoten }
      >>
      \new Lyrics \lyricsto "Bass" { \BassText }
    >>  % Ende ChoirStaff

    \new PianoStaff <<
      \set PianoStaff.instrumentName = #"Piano  "
      \new Staff = "oben" \oben
      \new Staff = "unten" \unten
    >>
  >>
}
```



3.4.3 Eine Partitur von Grund auf erstellen

Wenn Sie einige Fertigkeit im Schreiben von LilyPond-Code gewonnen haben, werden Sie vielleicht feststellen, dass es manchmal einfacher ist, von Grund auf anzufangen, anstatt die fertigen Vorlagen zu verändern. Auf diese Art könne Sie auch Ihren eigenen Stil entwickeln, und ihn der Musik anpassen, die Sie notieren wollen. Als Beispiel wollen wir demonstrieren, wie man die Partitur für ein Orgelpreludium von Grund auf konstruiert.

Beginnen wir mit dem Kopf, dem `header`-Abschnitt. Hier notieren wir den Titel, den Namen des Komponisten usw. Danach schreiben wir die einzelnen Variablen auf und schließlich am Ende die eigentliche Partitur, den `\score`-Abschnitt. Beginnen wir mit einer groben Struktur, in die wir dann die Einzelheiten nach und nach eintragen.

Als Beispiel benutzen wir zwei Takte aus dem Orgelpreludium *Jesu, meine Freude* von J. S. Bach, notiert für zwei Manuale und Pedal. Sie können die Noten am Ende dieses Abschnittes sehen. Das obere Manual trägt zwei Stimmen, das untere und das Pedalsystem jeweils nur eine. Wir brauchen also vier Variablen für die Noten und eine, um Taktart und Tonart zu definieren.

```
\version "2.12.1"
\header {
  title = "Jesu, meine Freude"
  composer = "J. S. Bach"
}
TimeKey = { \time 4/4 \key c \minor }
ManualOneVoiceOneMusic = {s1}
ManualOneVoiceTwoMusic = {s1}
ManualTwoMusic = {s1}
PedalOrganMusic = {s1}

\score {
}
```

Im Moment haben wir eine unsichtbare Note in jede Stimme eingesetzt (`s1`). Die Noten werden später hinzugefügt.

Als nächstes schauen wir uns an, was in die Partitur (die `\score`-Umgebung) kommt. Dazu wird einfach die Notensystemstruktur konstruiert, die wir benötigen. Orgelmusik wird meistens auf drei Systemen notiert, eins für jedes Manual und ein drittes für die Pedalnoten. Die Systeme für die Manuale werden mit einer geschweiften Klammer verbunden, wir benutzen hier also ein `PianoStaff`. Das erste Manualsystem braucht zwei Stimmen, das zweite nur eine.

```
\new PianoStaff <<
  \new Staff = "ManualOne" <<
    \new Voice { \ManualOneVoiceOneMusic }
    \new Voice { \ManualOneVoiceTwoMusic }
  >>  % end ManualOne Staff context
  \new Staff = "ManualTwo" <<
    \new Voice { \ManualTwoMusic }
  >>  % end ManualTwo Staff context
>>  % end PianoStaff context
```

Als nächstes soll das System für das Pedal hinzugefügt werden. Es soll unter das Klaviersystem gesetzt werden, aber muss gleichzeitig mit ihm erscheinen. Wir brauchen also spitze Klammern um beide Definitionen. Sie wegzulassen würde eine Fehlermeldung in der Log-Datei hervorrufen. Das ist ein sehr häufiger Fehler, der wohl auch Ihnen früher oder später unterläuft. Sie können das fertige Beispiel am Ende des Abschnittes kopieren und die Klammern entfernen, um zu sehen, wie die Fehlermeldung aussehen kann, die Sie in solch einem Fall erhalten würden.

```
<<  % PianoStaff and Pedal Staff must be simultaneous
  \new PianoStaff <<
    \new Staff = "ManualOne" <<
      \new Voice { \ManualOneVoiceOneMusic }
      \new Voice { \ManualOneVoiceTwoMusic }
    >>  % end ManualOne Staff context
    \new Staff = "ManualTwo" <<
      \new Voice { \ManualTwoMusic }
    >>  % end ManualTwo Staff context
  >>  % end PianoStaff context
  \new Staff = "PedalOrgan" <<
    \new Voice { \PedalOrganMusic }
  >>
>>
```

Es ist nicht notwendig, die simultane Konstruktion `<< .. >>` innerhalb des zweiten Manualsystems und des Pedalsystems zu benutzen, denn sie enthalten nur eine Stimme. Andererseits schadet es nichts, sie zu schreiben, und es ist eine gute Angewohnheit, immer die spitzen Klammern nach einem `\new Staff` zu schreiben, wenn mehr als eine Stimme vorkommen könnten. Für Stimmen (`Voice`) dagegengilt genau das Gegenteil: eine neue Stimme sollte immer von geschweiften Klammern (`{ .. }`) gefolgt werden, falls Sie ihre Noten in mehrere Variable aufteilen, die nacheinander gesetzt werden sollen.

Fügen wir also diese Struktur zu der `\score`-Umgebung hinzu und bringen wir die Einzüge in Ordnung. Gleichzeitig wollen wir die richtigen Schlüssel setzen und die Richtung der Hälse und Bögen in den Stimmen des oberen Systems kontrollieren, indem die obere Stimme ein `\voiceOne`, die untere dagegen ein `\voiceTwo` erhält. Die Taktart und Tonart werden mit unserer fertigen Variable `\TimeKey` eingefügt.

```
\score {
  <<  % PianoStaff and Pedal Staff must be simultaneous
```

```
    \new PianoStaff <<
      \new Staff = "ManualOne" <<
        \TimeKey  % set time signature and key
        \clef "treble"
        \new Voice { \voiceOne \ManualOneVoiceOneMusic }
        \new Voice { \voiceTwo \ManualOneVoiceTwoMusic }
      >>  % end ManualOne Staff context
      \new Staff = "ManualTwo" <<
        \TimeKey
        \clef "bass"
        \new Voice { \ManualTwoMusic }
      >>  % end ManualTwo Staff context
    >>  % end PianoStaff context
    \new Staff = "PedalOrgan" <<
      \TimeKey
      \clef "bass"
      \new Voice { \PedalOrganMusic }
    >>  % end PedalOrgan Staff
  >>
}  % end Score context
```

Damit ist das Grundgerüst fertig. Jede Orgelmusik mit drei Systemen hat die gleiche Struktur, auch wenn die Anzahl der Stimmen in einem System sich ändern kann. Jetzt müssen wir nur noch die Noten einfügen und alle Teile zusammenfügen, indem wir die Variablen mit einem Backslash in die Partitur einbauen.

```
\version "2.12.1"
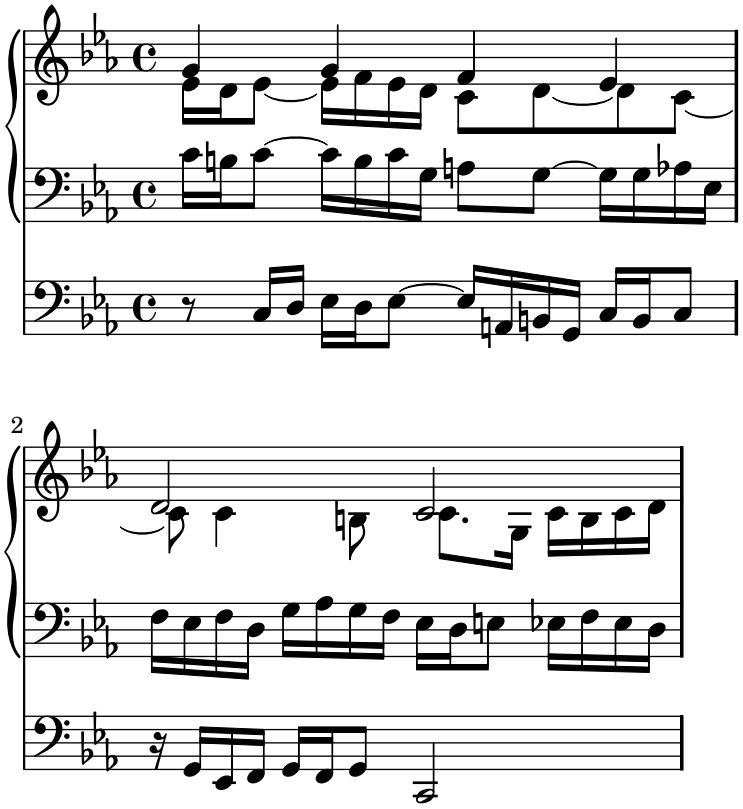
\header {
  title = "Jesu, meine Freude"
  composer = "J S Bach"
}
Zeitangabe = { \time 4/4 \key c \minor }
ManualEinsStimmeEinsNoten = \relative g' {
  g4 g f ees | d2 c2 |
}
ManualEinsStimmeZweiNoten = \relative c' {
  ees16 d ees8~ ees16 f ees d c8 d~ d c~ |
  c c4 b8 c8. g16 c b c d |
}
ManualZweiNoten = \relative c' {
  c16 b c8~ c16 b c g a8 g~ g16 g aes ees |
  f ees f d g aes g f ees d e8~ ees16 f ees d |
}
PedalOrgelNoten = \relative c {
  r8 c16 d ees d ees8~ ees16 a, b g c b c8 |
  r16 g ees f g f g8 c,2 |
  }

\score {
  <<  % PianoStaff und Pedal-System müssen gleichzeitig sein
    \new PianoStaff <<
      \new Staff = "ManualEins" <<
        \Zeitangabe  % Taktangabe und Tonart setzen
```

```
        \clef "treble"
        \new Voice { \voiceOne \ManualEinsStimmeEinsNoten }
        \new Voice { \voiceTwo \ManualEinsStimmeZweiNoten }
      >>  % Ende ManualEins Staff-Kontext
      \new Staff = "ManualZwei" <<
        \Zeitangabe
        \clef "bass"
        \new Voice { \ManualZweiNoten }
      >>  % Ende ManualZwei Staff-Kontext
    >>  % Klaviersystem beenden
    \new Staff = "OrgelPedal" <<
      \Zeitangabe
      \clef "bass"
      \new Voice { \PedalOrgelNoten }
    >>  % Orgelsystem beenden
  >>
}  % Partitur-Kontext beenden
```

Jesu, meine Freude

J S Bach



4 Die Ausgabe verändern

In diesem Kapitel wird erklärt, wie man die Notenausgabe verändern kann. In LilyPond kann man sehr viel konfigurieren, fast jedes Notenfragment kann geändert werden.

4.1 Grundlagen für die Optimierung

4.1.1 Grundlagen zur Optimierung

„Optimierung“ (engl. tweaking) ist ein LilyPond-Begriff für die verschiedenen Methoden, die Aktionen zu beeinflussen, die während der Kompilation einer Notationsdatei vorgenommen werden sowie auf das Notenbild einzuwirken. Einige dieser Opitmierungen sind sehr einfach, andere dagegen recht komplex. Aber insgesamt erlaubt das System an Optimierungen so gut wie alle möglichen Erscheindungsformen für die Notenausgabe.

In diesem Abschnitt werden die grundlegenden Konzepte vorgestellt, um die Optimierung zu verstehen. Später soll eine Anzahl von fertigen Befehlen bereitgestellt werden, die einfach in die Quelldatei kopiert werden können um den selben Effekt wie im Beispiel zu erhalten. Gleichzeitig zeigen diese Beispiele, wie die Befehle konstruiert werden, so dass Sie in der Lage sein werden, eigene Befehle auf dieser Grundlage zu entwickeln.

Bevor Sie mit diesem Kapitel beginnen, könnte Sie ein Blick in den Abschnitt Abschnitt 3.3 [Kontexte und Engraver], Seite 64 interessieren, dann Kontexte und Engraver sowie die Eigenschaften, die mit ihnen verknüpft sind, sind die Voraussetzung, um die Funktionsweise von Optimierungen verstehen zu können.

4.1.2 Objekte und Schnittstellen

Optimierung bedeutet, die internen Operationen und Strukturen des LilyPond-Programmes zu verändern, darum sollen hier zunächst die wichtigesten Begriffe erklärt werden, die zur Beschreibung dieser Operationen und Strukturen benutzt werden.

Der Begriff „Objekt“ ist ein allgemeiner Begriff, mit dem die Vielzahl an internen Strukturen bezeichnet wird, die LilyPond während der Bearbeitung des Quelltextes erstellt. Wenn etwa ein Befehl wie `\new Staff` auftritt, wird ein neues Objekt vom Typ `Staff` erstellt. Dieses Objekt `Staff` enthält dann alle Eigenschaften, die mit diesem speziellen Notensystem verknüpft sind, wie beispielsweise seine Bezeichnung, Tonart und spezifische Angaben über die Engraver, die innerhalb dieses Systems eingesetzt werden. Für alle anderen Kontexte gibt es genauso Objekte, die deren Eigenschaften beinhalten, beispielsweise für `Voice`-Objekte, `Score`-Objekte, `Lyrics`-Objekte, aber auch für Objekte, die Notationselemente wie die Taktlinien, Notenköpfe, Bögen und Dynamikbezeichnung enthalten. Jedes Objekt hat eine eigene Gruppe an Eigenschaftswerten.

Bestimmte Objekttypen tragen besondere Bezeichnungen. Objekte, die Notationselemente der gesetzten Ausgabe repräsentieren, also Notenköpfe, Hälse, Bögen, Fingersatz, Schlüssel usw., werden „Layout-Objekte“, oft auch „Graphische Objekte“ genannt. Daraus resultiert die künstliche Abkürzung „Grob“. Diese sind auch Objekte im allgemeinen Sinn und haben genauso Eigenschaften, die mit ihnen verknüpft sind, wie etwa Größe, Position, Farbe usw.

Einige Layout-Objekte sind etwas spezieller. Phrasierungsbögen, Crescendo-Klammern, Oktavierungszeichen und viele andere Grobs sind nicht an einer Stelle plaziert – sie haben vielmehr einen Anfangspunkt, einen Endpunkt und eventuell noch andere Eigenschaften, die ihre Form bestimmen. Objekte mit solch einer erweiterten Gestalt werden als „Strecker“ (engl. Spanners) bezeichnet.

Es bleibt uns noch übrig zu erklären, was „Schnittstellen“ (engl. interface) sind. Wenn auch viele Objekte sehr unterschiedlich sind, haben sie doch oft gemeinsame Eigenschaften, die auf die gleiche Weise verarbeitet werden. Alle Grobs beispielsweise haben eine Farbe, eine Größe,

eine Position usw. und alle diese Eigenschaften werden von LilyPond auf die gleiche Weise verarbeitet, während der Quelltext in Notensatz umgesetzt wird. Um die internen Operationen zu vereinfachen, sind alle diese gemeinsamen Prozesse und Eigenschaften in einem Objekt mit der Bezeichnung `grob-interface` (Schnittstelle eines graphischen Objektes) zusammengefasst. Es gibt viele andere Gruppen gemeinsamer Eigenschaften, die jede eine Bezeichnung besitzen, welche auf `-interface` endet. Insgesamt gibt es über 100 dieser Schnittstellen. Wir werden später sehen, was es damit auf sich hat.

Dies waren die Hauptbegriffe, die in diesem Kapitel zur Anwendung kommen sollen.

4.1.3 Regeln zur Benennung von Objekten und Eigenschaften

Es wurden schon früher einige Regeln zur Benennung von Objekten vorgestellt, siehe Abschnitt 3.3 [Kontexte und Engraver], Seite 64. Hier eine Referenzliste der häufigsten Objekt- und Eigenschaftsbezeichnungen mit den Regeln für ihre Bezeichnung und illustrierenden echten Bezeichnungen. Es wurde „A“ für einen beliebigen Großbuchstaben und „aaa“ für eine beliebige Anzahl an Kleinbuchstaben eingesetzt. Andere Zeichen werden explizit angegeben.

Objekt-/Eigenschaftstyp	**Naming convention**	**Beispiele**
Kontexte	Aaaa oder AaaaAaaaAaaa	Staff, GrandStaff
Layout-Objekte	Aaaa oder AaaaAaaaAaaa	Slur, NoteHead
Engraver	Aaaa_aaa_engraver	Clef_engraver, Note_heads_engraver
Schnittstellen	aaa-aaa-interface	grob-interface, break-aligned-interface
Kontext-Eigenschaften	aaa oder aaaAaaaAaaa	alignAboveContext, skipBars
Layout-Objekt-Eigenschaften	aaa oder aaa-aaa-aaa	direction, beam-thickness

Es wird bald ersichtlich werden, dass die Eigenschaften von unterschiedlichen Objekttypen mit unterschiedlichen Befehlen geändert werden. Deshalb ist es nützlich, aus der Schreibweise zu erkennen, um was für ein Objekt es sich handelt, um den entsprechenden Befehl einsetzen zu können.

4.1.4 Optimierungsmethoden

Der \override-Befehl

Wir haben uns schon mit den Befehlen `\set` und `\with` bekannt gemacht, mit welchen Eigenschaften von **Kontexten** verändert und **Engraver** entfernt oder hinzugefügt werden können. Siehe dazu Abschnitt 3.3.4 [Kontexteigenschaften verändern], Seite 68 und Abschnitt 3.3.5 [Engraver hinzufügen und entfernen], Seite 72. Jetzt wollen wir uns weitere wichtige Befehle anschauen.

Der Befehl, um die Eigenschaften von **Layout-Objekten** zu ändern, ist `\override`. Weil dieser Befehl interne Eigenschaften tief in der Programmstruktur von LilyPond verändern muss, ist seine Syntax nicht so einfach wie die der bisherigen Befehle. Man muss genau wissen, welche Eigenschaft welches Objektes in welchem Kontext geändert werder soll, und welches der neu zu setzende Wert dann ist. Schauen wir uns an, wie das vor sich geht.

Die allgemeine Syntax dieses Befehles ist:

```
\override Kontext.LayoutObjekt #'layout-eigenschaft =
#Wert
```

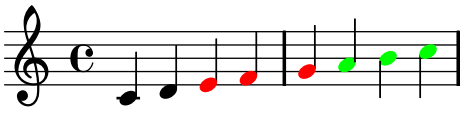
Damit wir die Eigenschaft mit der Bezeichnung *layout-property* das Layout-Objektes mit der Bezeichnung*LayoutObject*, welches ein Mitglied des *Kontext*-Kontextes ist, auf den Wert *value*.

Der *Kontext* kann (und wird auch normalerweise) ausgelassen werden, wenn der benötigte Kontext eindeutig impliziert ist und einer der untersten Kontexte ist, also etwa `Voice`, `ChordNames` oder `Lyrics`. Auch in diesem Text wird der Kontext oft ausgelassen werden. Später soll gezeigt werden, in welchen Fällen er ausdrücklich definiert werden muss.

Spätere Abschnitte behandeln umfassend Eigenschaften und ihre Werte, aber um ihre Funktion und ihr Format zu demonstrieren, werden wir hier nur einige einfache Eigenschaften und Werte einsetzen, die einfach zu verstehen sind.

Für den Moment könne Sie die `#'`-Zeichen ignorieren, die vor jeder Layout-Eigenschaft, und die `#`-Zeichen, die vor jedem Wert stehen. Sie müssen immer in genau dieser Form geschrieben werden. Das ist der am häufigsten gebrauchte Befehl für die Optimierung, und der größte Teil dieses Abschnittes wird dazu benutzt, seine Benutzung zu erläutern. Hier ein einfaches Beispiel, um die Farbe des Notenkopfes zu ändern:

```
c d
\override NoteHead #'color = #red
e f g
\override NoteHead #'color = #green
a b c
```



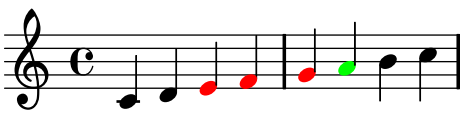
Der \revert-Befehl

Wenn eine Eigenschaft einmal überschrieben wurde, wird ihr neuer Wert so lange bewahrt, bis er noch einmal überschrieben wird oder ein `\revert`-Befehl vorkommt. Der `\revert`-Befehl hat die folgende Syntax und setzt den Wert der Eigenschaft zurück auf den Standardwert, nicht jedoch auf den vorigen Wert, wenn mehrere `\override`-Befehle benutzt wurden.

```
\revert Kontext.LayoutObjekt #'layout-eigenschaft
```

Wiederum, genauso wie der *Kontext* bei dem `\override`-Befehl, wird *Kontext* oft nicht benötigt. Er wird in vielen der folgenden Beispiele ausgelassen. Im nächsten Beispiel wird die Farbe des Notenkopfes wieder auf den Standardwert für die letzten zwei Noten gesetzt.

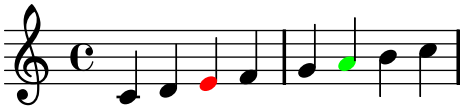
```
c d
\override NoteHead #'color = #red
e f g
\override NoteHead #'color = #green
a
\revert NoteHead #'color
b c
```



\once-Präfix

Sowohl der `\override`-Befehl als auch der `\set`-Befehl können mit dem Präfix `\once` (*einmal*) versehen werden. Dadurch wird der folgende `\override`- oder `\set`-Befehl nur für den aktuellen Musik-Moment wirksam, bevor sich wieder der Standard einstellt. Am gleichen Beispiel demonstriert, kann damit die Farbe eines einzelnen Notenkopfes geändert werden:

```
c d
\once \override NoteHead #'color = #red
e f g
\once \override NoteHead #'color = #green
a b c
```



Der \overrideProperty-Befehl
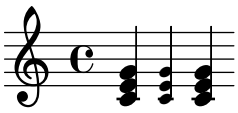
Es gibt eine andere Form des `override`-Befehls, `\overrideProperty` (überschreibe Eigenschaft), welcher ab und zu benötigt wird. Es wird hier nur der Vollständigkeit halber erwähnt, sein Einsatz wird demonstriert in Abschnitt “Schwierige Korrekturen” in *Notationsreferenz*.

Der \tweak-Befehl

Der letzte Optimierungsbefehl in LilyPond ist `\tweak` (engl. optimieren). Er wird eingesetzt um Eigenschaften von Objekten zu verändern, die zum selben Musik-Moment auftreten, wie etwa die Noten eines Akkordes. Ein `\override` würde alle Noten des Akkords beeinflussen, während mit `\tweak` nur das nächste Objekt der Eingabe geändert wird.

Hier ein Beispiel. Angenommen, die Größe des mittleren Notenkopfes (ein E) in einem C-Dur-Akkord soll geändert werden. Schauen wir zuerst, was wir mit `\once \override` erhalten:

```
<c e g>4
\once \override NoteHead #'font-size = #-3
<c e g>
<c e g>
```
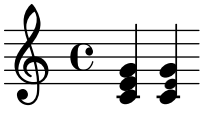


Wie man sehen kann, beeinflusst `override` *alle* Notenköpfe des Akkordes. Das liegt daran, dass alle die Noten eines Akkordes zum selben Musik-Moment auftreten und die Funktion von `\once` ist es, die Optimierung auf an allen Objekten auszuführen, die zum selben Musik-Moment auftreten wie der `\override`-Befehl.

Der `\tweak`-Befehl funktioniert anders. Er bezieht sich auf das direkt folgende Element in der Eingabe-Datei. Es wirkt aber auch nur mit Objekten, die direkt von der Eingabe kreirt werden, insbesondere Notenköpfe und Artikulationszeichen. Objekte wie etwa Hälse oder Versetzungszeichen werden erst später erstellt und lassen sich nicht auf diese Weise ändern. Zusätzlich *müssen* sich etwa Notenköpfe innerhalb eines Akkordes befinden, d. h. sie müssen von einfachen spitzen Klammern umschlossen sein. Um also eine einzelne Note mit `\tweak` zu verändern, muss der Befehl innerhalb der spitzen Klammern zusammen mit der Note eingegeben werden.

Um also zu unserem Beispiel zurückzukommen, könnte man die mittlere Note eines Akkordes auf diese Weise ändern:

```
<c e g>4
<c \tweak #'font-size #-3 e g>4
```
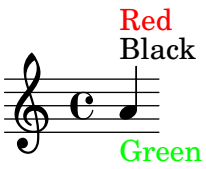


Beachten Sie, dass die Syntax des `\tweak`-Befehls sich von der des `\override`-Befehls unterscheidet. Weder Kontext noch Layout-Objekt konnen angegeben werden, denn das würde zu einem Fehler führen. Beide Angaben sind durch das folgende Element impliziert. Hier sollte auch kein Gleichheitzeichen vorhanden sein. Die verallgemeinerte Syntax des `\tweak`-Befehls ist also einfach

```
\tweak #'layout-eigenschaft #Wert
```

Ein `\tweak`-Befehl kann auch benutzt werden, um nur eine von mehreren Artikulationen zu ändern, wie im nächsten Beispiel zu sehen ist.

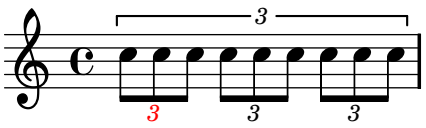
```
a ^Black
  -\tweak #'color #red ^Red
  -\tweak #'color #green _Green
```



Beachten Sie, dass dem `\tweak`-Befehl ein Artikulationsmodifikartor vorangestellt werden muss, ganz als ob er selbst ein Artikulationszeichen wäre.
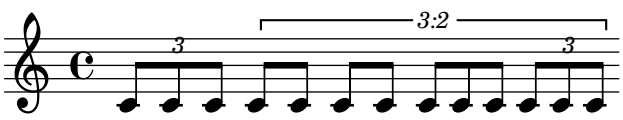
Der `\tweak`-Befehl muss auch benutzt werden, wenn das Aussehen einer vor mehreren geschachtelten Triolenklammern geändert werden soll, die zum selben Zeitpunkt beginnen. Im folgenden Beispiel beginnen die lange Klammer und die erste Triolenklammer zum selben Zeitpunkt, sodass ein `\override`-Befehl sich auf beide beziehen würde. In dem Beispiel wird `\tweak` benutzt, um zwischen ihnen zu unterscheiden. Der erste `\tweak`Befehl gibt an, dass die lange Klammer über den Noten gesetzt werden soll, und der zweite, dass die Zahl der rhythmischen Aufteilung für die erste der kurzen Klammern in rot gesetzt wird.

```
\tweak #'direction #up
\times 4/3 {
  \tweak #'color #red
  \times 2/3 { c8[ c8 c8] }
  \times 2/3 { c8[ c8 c8] }
  \times 2/3 { c8[ c8 c8] }
}
```



Wenn geschachtelte N-tolen nicht zum gleichen Zeitpunkt beginnen, kann ihr Aussehen auf die übliche Art mit dem `\override`-Befehl geändert werden:

```
\times 2/3 { c8[ c c]}
\once \override TupletNumber
  #'text = #tuplet-number::calc-fraction-text
\times 2/3 {
  c[ c]
  c[ c]
  \once \override TupletNumber #'transparent = ##t
  \times 2/3 { c8[ c c] }
\times 2/3 { c8[ c c]}
}
```



Siehe auch

Notationsreferenz: Abschnitt “Der tweak-Befehl” in *Notationsreferenz*.

4.2 Die Referenz der Programminterna
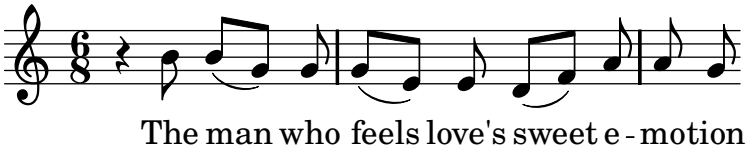
4.2.1 Eigenschaften von Layoutobjekten

Angenommen, in Ihrer Partitur tritt ein Legatobogen auf, der Ihrer Meinung nach zu dünn ausgefallen ist. Sie würden ihn gerne etwas schwerer gezeichnet sehen. Wie gehen Sie vor? Von den Anmerkungen in früheren Abschnitten wissen Sie schon, dass LilyPond sehr flexibel ist und eine derartige Modifikation möglich sein sollte, und Sie erraten vielleicht, dass ein `\override`-Befehl angebracht ist. Aber gibt es eine Eigenschaft für die Dicke eines Legatobogens (engl. slur), und wenn es sie gibt, auf welche Weise lässt sie sich verändern? Hier kommt die Referenz der Interna zur Geltung. Dort finden sich alle Informationen, um den beschriebenen und alle anderen `\override`-Befehle zu konstruieren.

Bevor Sie jetzt in die Referenz der Interna wechseln, ist eine Warnung angebracht. Es handelt sich um ein **Referenz**dokument, was heißt, dass es sehr wenig oder gar keine Erklärungen enthält: seine Aufgabe ist es, Information klar und genau darzustellen. Das bedeutet, dass es auf den ersten Blick entmutigend wirkt. Die Einführung und Erklärung in diesem Abschnitt wird Ihnen aber schnell ermöglichen, genau die Information aus der Referenz zu entnehmen, die Sie benötigen. Beachten Sie, dass die Referenz der Interna nur auf Englisch existiert. Um die Eigenschaftsbezeichnung eines bestimmten Objektes zu finden, können Sie das Glossar (siehe Abschnitt “Musikglossar” in *Glossar*) verwenden, in dem die englischen Begriffe in viele andere Sprachen übersetzt sind.

Das Vorgehen soll an einem konkreten Beispiel einer echten Komposition demonstriert werden. Hier das Beispiel:

```
{
  \time 6/8
  {
    r4 b8 b[( g]) g |
    g[( e]) e d[( f]) a |
    a g
  }
  \addlyrics {
    The man who feels love's sweet e -- mo -- tion
  }
}
```



Angenommen also, wir wollen die Legatobögen etwas dicker setzten. Ist das möglich? Die Legatobögen sind mit Sicherheit ein Layout-Objekt, die Frage muss also lauten: „Gibt es eine Eigenschaft von Legatobögen, die die Dicke bestimmt?“ Um diese Frage zu beantworten, müssen wir in der Referenz der Interna (kurz IR) nachschauen.

Die IR für die LilyPond-Version, die Sie benutzen, findet sich auf der LilyPond-Webseite unter der Adresse `http://lilypond.org`. Gehen Sie zur Dokumentationsseite und klicken Sie auf den Link zur Referenz der Interna. Die Sprache ändert sich ab hier nach englisch. Für diese Übung sollten Sie die HTML-Version benutzen, nicht die „auf einer großen Seite“ oder die PDF-Version. Damit Sie die nächsten Absätze verstehen können, müssen Sie genauso vorgehen, während Sie weiterlesen.

Unter der Überschrift **Top** befinden sich fünf Links. Wählen Sie den Link zum *Backend*, wo sich die Information über Layout-Objekte befindet. Hier, unter der Überschrift **Backend**, wählen

Sie den Link *All layout objects*. Die Seite, die sich öffnet, enthält ein Liste aller Layout-Objekte, die in Ihrerer LilyPond-Version benutzt werden, in alphabetischer Ordnung. Wählen Sie den Link *Slur* und die Eigenschaften der Legatobögen (engl. slur) werden aufgelistet.

Eine alternative Methode, auf diese Seite zu gelangen, ist von der Notationsreferenz aus. Auf einer der Seiten zu Legatobögen findet sich ein Link zur Referenz der Interna. Dieser Link führt Sie direkt auf diese Seite. Wenn Sie aber eine Ahnung haben, wie die Bezeichnung des Layout-Objektes lauten könnte, das sie ändern wollen, ist es oft schneller, direkt zur IR zu gehen und dort nachzuschlagen.

Aus der Slur-Seite in der IR könne wir entnehmen, dass Legatobögen (Slur-Objekte) durch den Slur_engraver erstellt werden. Dann werden die Standardeinstellungen aufgelistet. Beachten Sie, dass diese **nicht** in alphabetischer Reihenfolge geordnet sind. Schauen Sie sich die Liste an, ob sie eine Eigenschaft enthält, mit der die Dicke von Legatobögen kontrolliert werden kann. Sie sollten folgendes finden:

```
thickness (number)
     1.2
     Line thickness, generally measured in line-thickness
```

Das sieht ganz danach aus, als ob damit die Dicke geändert werden kann. Es bedeutet, dass der Wert von `thickness` einfach eine Zahl (*number*) ist, dass der Standardwert 1.2 ist, und dass die Einheit für die Dicke eine andere Eigenschaft mit der Bezeichnung `line-thickness` ist.

Wie schon früher gesagt, gibt es wenig bis gar keine Erklärungen in der IR, aber wir haben schon genug Informationen, um zu versuchen, die Dicke eines Legatobogens zu ändern. Die Bezeichnung des Layout-Objekts ist offensichtlich `Slur` und die Bezeichnung der Eigenschaft, die geändert werden soll `thickness`. Der neue Wert sollte etwas mehr als 1.2 sein, denn der Bogen soll ja dicker werden.
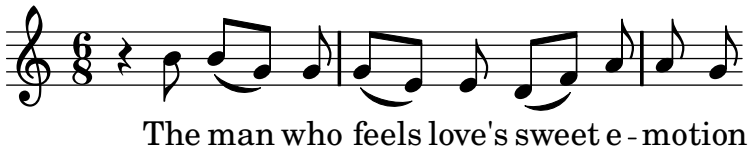
Den benötigten `\override`-Befehl können wir jetzt einfach konstruieren, indem wir die Werte für die Bezeichnungen in den Modellbefehl einfügen und den Kontext auslassen. Setzen wir einmal einen sehr großen Wert für die Dicke um zu sehen, ob der Befehl auch funktioniert. Also:

```
\override Slur #'thickness = #5.0
```

Vergessen Sie nicht das Rautenzeichen und Apostroph (`#'`) vor der Eigenschaftsbezeichnung und das Rautenzeichen vor dem neuen Wert!

Die nächste Frage ist nun: „Wohin soll dieser Befehl geschrieben werden?“ Solange wir uns noch im Lernstadium befinden, ist die beste Antwort: „Innerhalb der Noten, vor den ersten Legatobogen und nahe bei ihm.“ Also etwa so:

```
{
  \time 6/8
  {
    % Dicke aller folgenden Bögen von 1.2 zu 5.0 vergrößern
    \override Slur #'thickness = #5.0
    r4 b8 b[( g]) g |
    g[( e]) e d[( f]) a |
    a g
  }
  \addlyrics {
    The man who feels love's sweet e -- mo -- tion
  }
}
```



und wirklich wird der Legatobogen dicker.

Das ist also die grundlegende Herangehensweise, `\override`-Befehl zu formulieren. Es gibt einige zusätzliche Komplikationen, denen wir uns später widmen werden, aber Sie haben jetzt das Handwerkszeug, um Ihre eigenen Befehle zu konstruieren – wenn Sie auch noch etwas Übung benötigen. Die sollen Sie durch die folgenden Übungen erhalten.
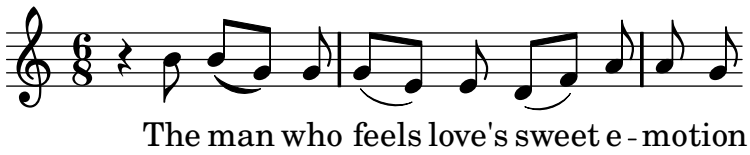
Den Kontext finden

Manchmal muss dennoch der Kontext spezifiziert werden. Welcher aber ist der richtige Kontext? Wir könnten raten, dass Legatobögen sich im `Voice`-Kontext befinden, denn sie sind immer einzelnen Melodielinien zugewiesen. Aber wir können uns dessen nicht sicher sein. Um unsere Annahme zu überprüfen, gehen wir wieder zu der Seite im IR, die die Legatobögen beschreibt und die Überschrift *Slur* hat. Dort steht: „Slur objects are created by: Slur engraver“. Legatobögen werden also in dem Kontext erstellt, in dem sich der `Slur_engraver` befindet. Folgen Sie dem Link zu der `Slur_engraver`-Seite. Unten auf der Seite steht, dass der `Slur_engraver` sich in fünf Stimmen-Kontexten befindet, unter anderem auch im normalen `Voice`-Kontext. Unsere Annahme war also richtig. Und weil `Voice` einer der Kontexte der untersten Ebene ist, welcher eindeutig schon dadurch definiert ist, dass wir Noten eingeben, kann er an dieser Stelle auch weggelassen werden.

Nur einmal mit \override verändern

Im Beispiel oben wurden *alle* Legatobögen dicker gesetzt. Vielleicht wollen Sie aber nur den ersten Bogen dicker haben. Das können Sie mit dem `\once`-Befehl erreichen. Er wird direkt vor den `\override`-Befehl gesetzt und bewirkt, dass nur der Bogen geändert wird, der **unmittelbar an der nächsten Note beginnt.** Wenn die nächste Note keinen Bogenbeginn hat, dann passiert gar nichts – der Befehl wird nicht gespeichert, sondern einfach vergessen. Der Befehl, mit `\once` zusammen benutzt, muss also wie folgt positioniert werden:

```
{
  \time 6/8
  {
    r4 b8
    % Nur die Dicke des direkt folgenden Bogens vergrößern
    \once \override Slur #'thickness = #5.0
    b[( g]) g |
    g[( e]) e d[( f]) a |
    a g
  }
  \addlyrics {
    The man who feels love's sweet e -- mo -- tion
  }
}
```
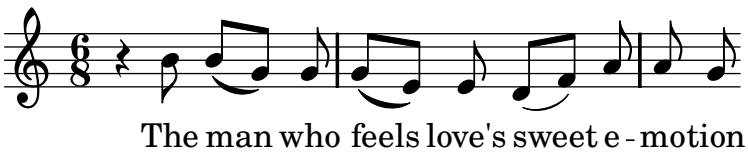


Jetzt bezieht er sich nur noch auf den ersten Legatobogen.

Der `\once`-Befehl kann übrigens auch vor einem `\set`-Befehl eingesetzt werden.

Rückgängig machen
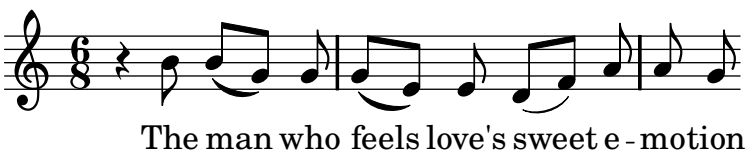
Eine weitere Möglichkeit: nur die beiden ersten Legatobögen sollen dicker gesetzt werden. Gut, wir könnten jetzt zwei Befehle benutzen, jeden mit dem `\once`-Präfix und direkt vor die entsprechende Note gestellt, an welcher der Bogen beginnt:

```
{
  \time 6/8
  {
    r4 b8
    % Nur die Dicke des direkt folgenden Bogens vergrößern
    \once \override Slur #'thickness = #5.0
    b[( g]) g |
    % Nur die Dicke des direkt folgenden Bogens vergrößern
    \once \override Slur #'thickness = #5.0
    g[( e]) e d[( f]) a |
    a g
  }
  \addlyrics {
    The man who feels love's sweet e -- mo -- tion
  }
}
```



Wir könnten aber auch den `\once`-Befehl weglassen und anstelle dessen später den `\revert`-Befehl einsetzen, um die `thickness`-Eigenschaft wieder auf ihren Standardwert zurückzusetzen:

```
{
  \time 6/8
  {
    r4 b8
    % Dicke aller folgenden Bögen von 1.2 zu 5.0 vergrößern
    \override Slur #'thickness = #5.0
    b[( g]) g |
    g[( e])
    % Die Dicke aller folgenden Bögen zurücksezten auf 1.2
    \revert Slur #'thickness
    e d[( f]) a |
    a g
  }
  \addlyrics {
    The man who feels love's sweet e -- mo -- tion
  }
}
```



Der `\revert`-Befehl kann benutzt werden, um eine beliebige Eigenschaft, die mit `\override` geändert worden ist, wieder in ihre Standardeinstellungen zurückzuversetzen. In unserem Beispiel können Sie die Methode benutzen, die Ihnen lieber ist, beide haben das gleiche Resultat.

Damit endet die Einleitung in die Referenz der Interna (IR) und die grundlegenden Optimierungsmethoden. Einige Beispiele folgen in späteren Abschnitten dieses Kapitel, einerseits um Sie mit weiteren Möglichkeiten der IR bekanntzumachen, andererseits um Ihnen mehr Übungsmöglichkeiten zu geben, die relevante Information dort zu finden. Die Beispiele werden Schritt für Schritt immer weniger Erklärungen beinhalten.

4.2.2 Eigenschaften, die Schnittstellen besitzen können

Der Text unseres Beispiels soll jetzt kursiv gesetzt werden. Was für ein `\override`-Befehl wird dazu benötigt? Schauen wir uns zunächst das Inhaltsverzeichnis in der IR an: „All layout objects“, wie auch schon zuvor. Welches Objekt könnte die Darstellung des Textes (engl. lyrics) beeinflussen? Es gibt den Eintrag `LyricText`, das hört sich schon sehr gut an. Ein Klick hierauf zeigt alle Eigenschaften an, die verändert werden können. Dazu gehört `font-series` und `font-size`, aber nichts, womit man kursiven Text erreichen könnte. Das liegt daran, dass die Schnitteigenschaft allen Schrift-Objekten gemeinsam ist. Sie findet sich also nicht in jedem einzelnen Layout-Objekt aufgelistet, sondern ist mit anderen ähnlichen Eigenschaften zusammen in einem **Interface** – einer Schnittstelle – verortet; in diesem Fall das `font-interface`.

Jetzt müssen wir also lernen, wie wir Eigenschaften von Schnittstellen finden und wie wir herausfinden, welche Objekte diese Schnittstelleneigenschaften benutzen.

Schauen Sie sich noch einmal die Seite in der IR an, die `LyricText` beschreibt. Unten auf der Seite ist eine klickbare Liste (in der HTML-Version der IR) an Eigenschaften, die von `LyricText` unterstützt werden. Diese Liste enthält sieben Einträge, darunter auch `font-interface`. Ein Klick hierauf bringt uns zu den Eigenschaften, die mit dieser Schnittstelle verbunden sind, also auch `LyricText`.

Jetzt sehen wir alle die Eigenschaften, die der Benutzer verändern kann, um die Schriftartendarstellung zu beeinflussen. Dazu gehört nun auch `font-shape(symbol)`, wobei `symbol` auf die Werte `upright` (gerade), `italics` (kursiv) oder `caps` (Kapitälchen) gesetzt werden kann.

Sie werden gemerkt haben, dass `font-series` und `font-size` hier auch aufgelistet sind. Es stellt sich die Frage, warum diese allgemeinen Schriftarteigenschaften `font-series` und `font-size` sowohl unter der Überschrift `LyricText` als unter dem `font-interface` aufgelistet sind, aber `font-shape` befindet sich nur im `font-interface`? Die Antwort ist: Die globalen Einstellungen von `font-series` und `font-size` werden geändert, wenn ein `LyricText`-Objekt erstellt wird, aber `font-shape` wird davon nicht beeinflusst. Die zusätzlichen Einträge unter der Überschrift `LyricText` beinhalten dann die Werte der Standardeinstellungen dieser zwei Eigenschaften, wenn es sich um ein `LyricText`-Objekt handelt. Andere Objekte, die auch das `font-interface` unterstützen, setzen diese Eigenschaften anders, wenn sie erstellt werden.

Versuchen wir nun einen `\override`-Befehl zu konstruieren, der den Gesantext kursiv setzt. Das Objekt hat die Bezeichnung `LyricText`, die Eigenschaft ist `font-shape` und der Wert `italic`. Wie vorher schon lassen wir den Kontext aus.
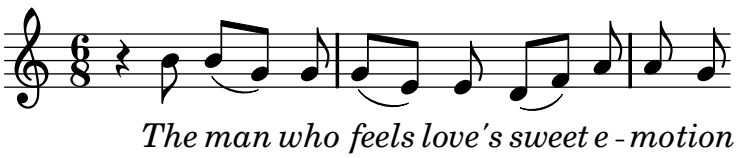
Am Rande sei angemerkt, dass die Werte der `font-shape`-Eigenschaft mit einem Apostroph (`'`) gekennzeichnet werden müssen, weil es sich um Symbole handelt. Aus dem gleichen Grund mussten auch für `thickness` weiter oben im Text ein Apostroph gesetzt werden. Symbole sind besondere Bezeichnungen, die LilyPond intern bekannt sind. Einige sind Bezeichnungen von Eigenschaften, wie eben `thickness` oder `font-shape`. Andere sind besondere Werte, die an Eigenschaften übergeben werden können, wie `italic`. Im Unterschied hierzu gibt es auch beliebige Zeichenketten, die immer mit Anführungszeichen, also als `"Zeichenkette"` auftreten. Für weitere Einzelheiten zu Zeichenketten und Werten, siehe Anhang B [Scheme-Übung], Seite 181.

Gut, der `\override`-Befehl, mit dem der Gesangstext kursiv gesetzt wird, lautet:

```
\override LyricText #'font-shape = #'italic
```

und er muss direkt vor den Text gesetzt werden, auf den er sich bezieht, etwa so:

```
{
  \time 6/8
  {
    r4 b8 b[( g]) g |
    g[( e]) e d[( f]) a |
    a g
  }
  \addlyrics {
    \override LyricText #'font-shape = #'italic
    The man who feels love's sweet e -- mo -- tion
  }
}
```



Jetzt wird der Text kursiv gesetzt.

Den Kontext im Liedtextmodus bestimmen

Bei Gesangstexten funktioniert der `\override`-Befehl nicht mehr, wenn Sie den Kontext im oben dargestellten Format angeben. Eine Silbe wird im Gesangtextmodus (lyricmode) entweder von einem Leerzeichen, einer neuen Zeile oder einer Zahl beendet. Alle anderen Zeichen werden als Teil der Silbe integriert. Aus diesem Grund muss auch vor der schließenden Klammer `}` ein Leerzeichen gesetzt oder eine neue Zeile begonnen werden. Genauso müssen Leerzeichen vor und nach einem Punkt benutzt werden, um die Kontext-Bezeichnung von der Objekt-Bezeichnung zu trennen, denn sonst würden beide Bezeichnungen als ein Begriff interpretiert und von LilyPond nicht verstanden werden. Der Befehl muss also lauten:

```
\override Lyrics . LyricText #'font-shape = #'italic
```

Achtung: Innerhalb von Gesangstext muss immer ein Leerzeichen zwischen der letzten Silbe und der schließenden Klammer gesetzt werden.

Achtung: Innerhalb von `\override`-Befehlen in Gesangstexten müssen Leerzeichen um Punkte zwischen Kontext- und Objektbezeichnungen gesetzt werden.

4.2.3 Typen von Eigenschaften

Bis jetzt hatten wir es mit zwei Arten von Eigenschaften zu tun: `number` (Zahl) und `symbol`. Damit ein Befehl funktioniert, muss der Wert einer Eigenschaft vom richtigen Typ sein und die Regeln befolgen, die für diesen Typ gelten. Der Eigenschaftstyp ist in der IR in Klammern hinter der Eigenschaftsbezeichnung angegeben. Hier eine Liste der Typen, die Sie vielleicht benötigen werden, mit den Regeln, die für den jeweiligen Typ gelten und einigen Beispielen. Sie müssen immer ein Rautenzeichen (`#`) vor den Typeintrag setzen, wenn sie in einem `\override`-Befehl benutzt werden.

Eigenschaftstyp	Regeln	Beispiele
Boolesch	Entweder wahr oder falsch, dargestellt als #t oder #f	`#t`, `#f`
Dimension (in Notenlinienabständen)	Eine positive Dezimalzahl (in Notenlinienabstand-Einheiten)	`2.5`, `0.34`
Richtung	Eine gültige Richtungskonstante oder das numerische Äquivalent	`LEFT`, `CENTER`, `UP`, `1`, `-1`
Integer	Eine positive ganze Zahl	`3`, `1`
Liste	Eine eingeklammerte Anzahl von Einträgen, mit Klammern getrennt und angeführt von einem Apostroph	`'(left-edge staff-bar)`, `'(1)`, `'(1.0 0.25 0.5)`
Textbeschriftung (markup)	Beliebige gültige Beschriftung	`\markup { \italic "cresc." }`
Moment	Ein Bruch einer ganzen Note, mit der make-moment-Funktion konstruiert	`(ly:make-moment 1 4)`, `(ly:make-moment 3 8)`
Zahl	Eine beliebige positive oder negative Dezimalzahl	`3.5`, `-2.45`
Paar (Zahlenpaar)	Zwei Zahlen getrennt von „Leerzeichen . Leerzeichen“, eingeklammert und angeführt von einem Apostroph	`'(2 . 3.5)`, `'(0.1 . -3.2)`
Symbol	Eine beliebige Anzahl von Symbolen, die für die Eigenschaft gültig sind, angeführt von einem Apostroph	`'italic`, `'inside`
Unbekannt	Eine Prozedur oder `#f` (um keine Aktion hervorzurufen)	`bend::print`, `ly:text-interface::print`, `#f`
Vektor	Eine Liste mit drei Einträgen, eingeklammert und mit Apostroph-Raute (`'#`) angeführt.	`'#(#t #t #f)`

Siehe auch

Handbuch zum Lernen: Anhang B [Scheme-Übung], Seite 181.

4.3 Erscheinung von Objekten

In diesem Abschnitt wollen wir zeigen, wie die Kenntnisse der vorigen Abschnitte in der Praxis angewandt werden können, um das Aussehen des Musiksatzes zu beeinflussen.

4.3.1 Sichtbarkeit und Farbe von Objekten

In Unterrichtsmaterial für den Musikunterricht wird oft eine Partitur dargestellt, in der bestimmte Notationselemente fehlen, so dass der Schüler die Aufgabe bekommt, die nachzutragen. Ein einfaches Beispiel ist etwa, die Taktlinien zu entfernen, damit der Schüler sie selber zeichnen kann. Aber die Tatklinien werden normalerweise automatisch eingefügt. Wie verhindern wir, dass sie ausgegeben werden?

Bevor wir uns hieran machen, sei daran erinnert, dass Objekteigenschaften in sogenannten *Schnittstellen* – engl. interface – gruppiert sind, siehe auch Abschnitt 4.2.2 [Eigenschaften], Seite 95. Das dient ganz einfach dazu, die Eigenschaften zusammenzufassen, die üblicherweise zusammen benötigt werden – wenn eine davon für ein Objekt gilt, dann auch die anderen. Manche Objekte brauchen die Eigenschaften von der einen Schnittstelle, andere von einer anderen. Die
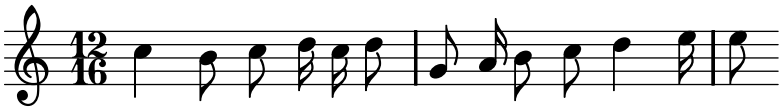
Schnittstellen, die die Eigenschaften von einem bestimmten Grob beinhalten, sind in der IR unten auf der Seite aufgelistet, die dieses Grob beschreibt. Die Eigenschaften können betrachtet werden, indem die Seite der entsprechenden Schnittstelle geöffnet wird.

Zu Information, wie man Eigenschaften von Grobs findet, siehe Abschnitt 4.2.1 [Eigenschaften von Layoutobjekten], Seite 91. Wir benutzen also jetzt die selbe Methode um in der IR das Layout-Objekt zu finden, dass für die Tatklinien zuständig ist. Über die Überschriften *Backend* und *All layout objects* kommen wir zu einem Layout-Objekt mit der Bezeichnung `BarLine` (engl. TaktLinie). Seine Eigenschaften beinhalten zwei, die über die Sichtbarkeit entscheiden: `break-visibility` und `stencil`. `BarLine` unterstützt auch einige Schnittstellen, unter anderem `grob-interface`, wo wir eine `transparent` und eine `color`-Eigenschaft finden. Alle können die Sichtbarkeit von Taktlinien (und natürlich auch die Sichtbarkeit von vielen anderen Objekten) beeinflussen. Schauen wir uns diese Eigenschaften eine nach der anderen an.

stencil (Matrize)
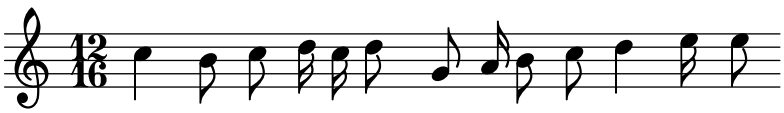
Diese Eigenschaft kontrolliert die Erscheinung der Taktlinien, indem sie das Symbol bestimmt, das ausgegeben werden soll. Wie bei vielen anderen Eigenschaften auch, kann sie so eingestellt werden, dass sie nichts ausgibt, indem ihr Wert auf `#f` (falsch) gesetzt wird. Ein Versuch also, wie vorher, indem wir den impliziten Kontext (`Voice`) auslassen:

```
{
  \time 12/16
  \override BarLine #'stencil = ##f
  c4 b8 c d16 c d8 |
  g, a16 b8 c d4 e16 |
  e8
}
```



Die Taktlinien werden aber immer noch angezeigt. Was ist da falsch gelaufen? Gehen Sie zurück zur IR und schauen Sie auf die Seite, die die Eigenschafter für `BarLine` angibt. Oben auf der Seite steht: „Barline objects are created by: Bar_engraver“. Schauen Sie sich die `Bar_engraver`-Seite an. Unten auf der Seite steht eine Liste der Kontexte, in denen der Takt-Engraver funktioniert. Alle Kontexte sind `Staff`-Typen (also Notensystem-Typen). Der Grund, warum der `\override`-Befehl nicht funktioniert hat, liegt also darin, dass das Taktlinie-Objekt (`BarLine`) sich nicht im `Voice`-Kontextbefindet. Wenn der Kontext falsch angegeben wird, bewirkt der Befehl einfach gar nichts. Keine Fehlermeldung wird ausgegeben und auch nichts in die Log-Datei geschrieben. Versuchen wir also, den richtigen Kontext mitanzugeben:

```
{
  \time 12/16
  \override Staff.BarLine #'stencil = ##f
  c4 b8 c d16 c d8 |
  g, a16 b8 c d4 e16 |
  e8
}
```
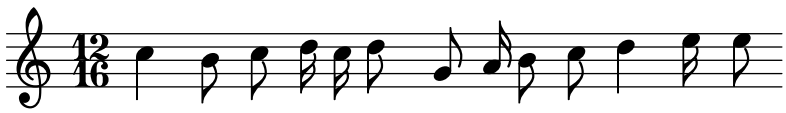


Jetzt sind die Taktlinien wirklich verschwunden.

break-visibility (unsichtbar machen)

Aus der Beschreibung der Eigenschaften für `BarLine` in der IR geht hervor, dass die `break-visibility`-Eigenschaft einen Vektor mit drei Booleschen Werten benötigt. Diese kontrollieren jeweils, ob die Taktlinien am Ende einer Zeile, in der Mitte einer Zeile und am Anfang einer Zeile ausgegeben werden. Wenn also alle Taktlinien unsichtbar sein sollen, wie in unserem Beispiel, brauchen wir den Wert `'#(#f #f #f)`. Versuchen wir es also, und berücksichtigen wir auch den `Staff`-Kontext. Beachten Sie auch, dass Sie `#'#` vor der öffnenden Klammer schreiben müssen: `'#` wird benötigt als Teil des Wertes, um einen Vektor zu signalisieren, und das erste `#` wird benötigt, um den Wert in einem `\override`-Befehl anzuführen.

```
{
  \time 12/16
  \override Staff.BarLine #'break-visibility = #'#(#f #f #f)
  c4 b8 c d16 c d8 |
  g, a16 b8 c d4 e16 |
  e8
}
```
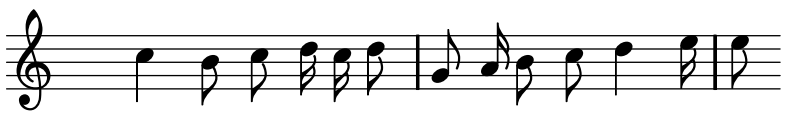


Auch auf diesem Weg gelingt es, die Taktlinien unsichtbar zu machen.

transparent (durchsichtig)

Aus den Eigenschaftsdefinitionen auf der `grob-interface`-Seite in der IR geht hervor, dass die `transparent`-Eigenschaft boolesch ist. Mit `#t` (wahr) wird also ein Grob durchsichtig gemacht. Im unserem Beispiel soll jetzt die Taktart durchsichtig gemacht werden, anstatt die Taktlinien durchsichtig zu machen. Wir brauchen also wieder die Grob-Bezeichnung für die Taktart. Auf der „All layout objects“-Seite in der IR müssen wir die Eigenschaften des `TimeSignature`-Layout-Objekts suchen Das Objekt wird vom `Time_signature_engraver` erstellt, der sich auch im `Staff`-Kontext befindet und genauso das `grob-interface` unterstützt, wie Sie sich überzeugen können. Der Befehl, um die Taktangabe unsichtbar zu machen, ist also:
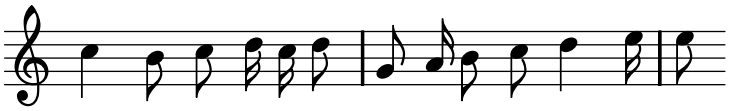
```
{
  \time 12/16
  \override Staff.TimeSignature #'transparent = ##t
  c4 b8 c d16 c d8 |
  g, a16 b8 c d4 e16 |
  e8
}
```



Die Taktangabe ist verschwunden, aber mit diesem Befehl wird ein freier Platz gelassen, wo sich die Taktangabe eigentlich befinden würde. Das braucht man vielleicht für eine Schulaufgabe, in der die richtige Taktangabe eingefügt werden soll, aber in anderen Fällen ist diese Lücke nicht schön. Um auch die Lücke zu entfernen, muss die Matrize (stencil) der Taktangabe auf `#f` (falsch) gesetzt werden:

```
{
  \time 12/16
```

```
  \override Staff.TimeSignature #'stencil = ##f
  c4 b8 c d16 c d8 |
  g, a16 b8 c d4 e16 |
  e8
}
```
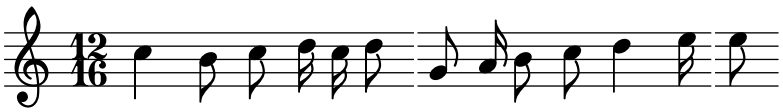


Und der Unterschied wird deutlich: hiermit wird das gesamte Objekt entfernt, während man mit `transparent` ein Objekt unsichtbar machen kann, es aber an seinem Platz gelassen wird.

color (Farbe)

Abschließend wollen wir die Taktlinien unsichtbar machen, indem wir sie weiß einfärben. (Es gibt hier eine Schwierigkeit: die weiße Taktlinie übermalt manchmal die Taktlinienen, wo sie sie kreuzt, manchmal aber auch nicht. Sie können in den Beispielen unten sehen, dass das nicht vorhersagbar ist. Die Einzelheiten dazu, warum das passiert und wie sie es kontrollieren können, werden dargestellt in Abschnitt "Objekte weiß malen" in *Notationsreferenz*. Im Moment wollen wir lernen, wie man mit Farbe arbeitet, akzeptieren Sie bitte an dieser Stelle die Beschränkung.)

Das `grob-interface` bestimmt, dass der Wert der Farb-Eigenschaft eine Liste ist, aber es gibt keine Erklärung, was für eine Liste das sein soll. Die Liste, die benötigt wird, ist eine Liste mit Werten in internen Einheiten, aber damit Sie nicht wissen müssen, wie diese aussehen, gibt es mehrere Wege, Farben anzugeben. Der erste Weg ist es, „normale" Farben zu benutzen, wie sie in der Tabelle in Abschnitt "Liste der Farben" in *Notationsreferenz* aufgelistet sind. Beachten Sie, dass die Bezeichnungen auf English sind. Um die Taktlinien auf weiß zu setzen, können Sie schreiben:

```
{
  \time 12/16
  \override Staff.BarLine #'color = #white
  c4 b8 c d16 c d8 |
  g, a16 b8 c d4 e16 |
  e8
}
```
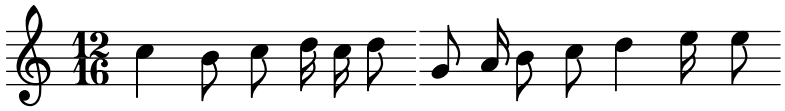


und die Taktlinien verschwinden in der Tat. Beachten Sie, dass *white* nicht mit einem Apostroph angeführt wird – es ist kein Symbol, sondern eine *Funktion*. Wenn sie aufgerufen wird, stellt sie eine Liste mit internen Werten zu Verfügung, mit welcher die Farbe auf weiß gestellt wird. Die anderen Farben in der Liste sind auch Funktionen. Um sich zu überzeugen, dass der Befehl auch wirklich funktioniert, können Sie die Farbe auf eine der anderen Funktionen dieser Liste abändern.

Die zweite Art die Farbe zu ändern geschieht, indem die Liste der X11-Farbbezeichnungen einzusetzen, siehe die zweite Liste in Abschnitt "Liste der Farben" in *Notationsreferenz*. Diesen Farben muss jedoch eine andere Funktion vorangestellt werden, die die X11-Farbbezeichnungen in interne Werte konvertiert: `x11-color`. Das geschieht wie folgt:

```
{
  \time 12/16
```
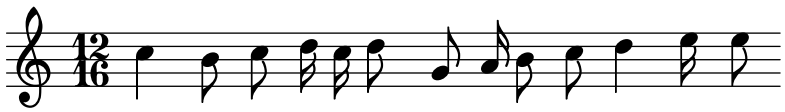
```
  \override Staff.BarLine #'color = #(x11-color 'white)
  c4 b8 c d16 c d8 |
  g, a16 b8 c d4 e16 |
  e8
}
```



In diesem Fall hat die Funktion `x11-color` ein Symbol als Argument, darum muss dem Symbol ein Apostroph vorangestellt und beide zusammen in Klammern gesetzt werden.

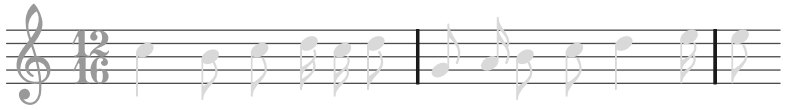
Es gibt noch eine dritte Funktion, die RGB-Werte in die internen Werte übersetzt – die `rgb-color`-Funktion. Sie braucht drei Argumente, um die Stärke von Rot, Grün und Blau darzustellen. Die Werte befinden sich zwischen 0 und 1. Um also die Farbe Rot darzustellen, muss der Wert der Funktion lauten: `(rgb-color 1 0 0)`, weiß würde sein: `(rgb-color 1 1 1)`.

```
{
  \time 12/16
  \override Staff.BarLine #'color = #(rgb-color 1 1 1)
  c4 b8 c d16 c d8 |
  g, a16 b8 c d4 e16 |
  e8
}
```



Schließlich gibt es noch eine Grauskala, die zu den X11-Farben gehört. Sie reicht von schwarz (`'grey0'`) bis weiß (`'grey100`), in Einserschritten. Wir wollen das illustrieren, indem alle Layout-Objekte im Beispiel verschiede Grauschattierungen erhalten:

```
{
  \time 12/16
  \override Staff.StaffSymbol   #'color = #(x11-color 'grey30)
  \override Staff.TimeSignature #'color = #(x11-color 'grey60)
  \override Staff.Clef          #'color = #(x11-color 'grey60)
  \override Voice.NoteHead      #'color = #(x11-color 'grey85)
  \override Voice.Stem          #'color = #(x11-color 'grey85)
  \override Staff.BarLine       #'color = #(x11-color 'grey10)
  c4 b8 c d16 c d8 |
  g, a16 b8 c d4 e16 |
  e8
}
```
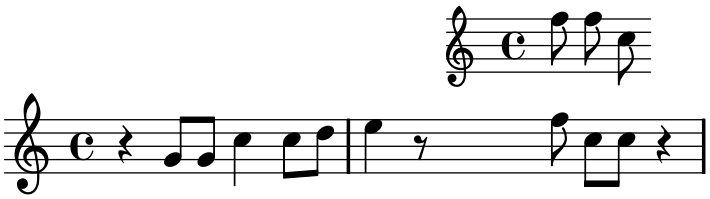


Beachten Sie die Kontexte, die mit jedem einzelnen Layout-Objekt verbunden sind. Es ist wichtig, den richtigen Kontext einzusetzen, damit die Befehle funktionieren. Denken Sie daran,

dass der Kontext sich daran orientiert, wo sich der entsprechende Engraver befindet. Den Standardkontext für Engraver finden Sie, indem Sie beim Layout-Objekt beginnen, zum Engraver gehen, der es produziert und auf der Seite des Engravers in der IR finden Sie Information, in welchem Kontext sich der Engraver normalerweise befindet.

4.3.2 Größe von Objekten
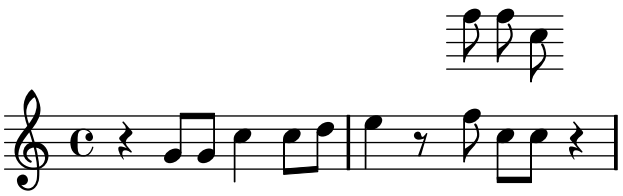
Als Startpunkt wollen wir wieder ein früheres Beispiel wählen, siehe Abschnitt 3.1.3 [Musikalische Ausdrücke ineinander verschachteln], Seite 46. Hier wurde ein neues Notensystem erstellt, wie man es für ein Abschnitt “Ossia” in *Glossar* braucht.

```
\new Staff ="main" {
       \relative g' {
         r4 g8 g c4 c8 d |
         e4 r8
         <<
           { f c c }
           \new Staff \with {
             alignAboveContext = #"main" }
           { f8 f c }
         >>
         r4 |
       }
     }
```



Ossia-Systeme werden normalerweise ohne Schlüssel und Taktangabe geschrieben, und sie werden etwas kleiner als das Hauptsystem gesetzt. Wie man Schlüssel und Taktangabe entfernt, wissen wir schon: wir setzen den Stencil von beiden auf `#f`:

```
\new Staff ="main" {
  \relative g' {
    r4 g8 g c4 c8 d |
    e4 r8
    <<
      { f c c }
      \new Staff \with {
        alignAboveContext = #"main"
      }
      {
        \override Staff.Clef #'stencil = ##f
        \override Staff.TimeSignature #'stencil = ##f
        { f8 f c }
      }
    >>
    r4 |
  }
}
```



wobei ein zusätzliches Klammerpaar nach der `\with`-Konstruktion erforderlich ist um sicherzugehen, dass die Modifikation und die Noten sich auch auf das Ossia-System beziehen.
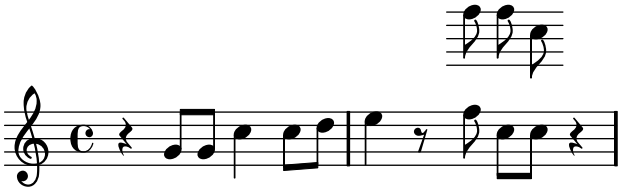
Was für einen Unterschied macht es, ob man den `Staff`-Kontext mit `\with` verändert, oder ob man die Stencils mit `\override` beeinflusst? Der größte Unterschied liegt darin, dass Änderungen, die mit `\with` eingeführt werden, während der Erstellung des Kontextes miterzeugt werden und als **Standardeinstellungen** für diesen Kontext während seiner gesamten Dauer gelten, während `\set`- oder `\override`-Befehle dynamisch in die Noten eingebettet werden – sie führen die Änderungen synchron mit einem bestimmten Zeitpunkt in der Musik aus. Wenn die Änderungen mit `\unset` oder `\revert` rückgängig gemacht werden, werden wieder die Standardwerte eingesetzt, die also die sind, die mit einer `\with`-Konstruktion definiert wurden, oder wenn hier keine definiert worden sind, die normalen Standardwerte.

Manche Kontexteigenschaften können nur ein einer `\with`-Konstruktion verändert werden. Das sind Eigenschaften, die nicht sinnvoll mitten im System geändert werden können. `alignAboveContext` (Orientierung über dem Kontext) und die Parallele, `alignBelowContext` (Orientierung unter dem Kontext) sind zwei derartige Eigenschaften – wenn das Notensystem einmal erstellt wurde, ist die Orientierung schon bestimmt und es wäre nicht sinnvoll, sie später zu ändern.

Die Standardwerte für Layout-Objekt-Eigenschaften können auch in der `\with`-Konstruktion gesetzt werden. Benutzen Sie einfach den normalen `\override`-Befehl ohne den Kontext, denn der Kontext ist eindeutig definiert durch die Stelle, an welcher sich `\with` befindet. Wenn an dieser Stelle ein Kontext angegeben wird, produziert LilyPond eine Fehlermeldung.

Das obige Beispiel könnte also auch so aussehen:

```
\new Staff ="main" {
  \relative g' {
    r4 g8 g c4 c8 d |
    e4 r8
    <<
      { f c c }
      \new Staff \with {
        alignAboveContext = #"main"
        % Keine Schlüssel in diesem System
        \override Clef #'stencil = ##f
        % Keine Taktangabe in diesem System
        \override TimeSignature #'stencil = ##f
      }
        { f8 f c }
    >>
    r4 |
  }
}
```



Nun können wir daran gehen, auch wirklich die Größe der Objekte zu ändern.

Manche Layout-Objekte werden aus Glyphen erstellt, die sich in einer Schriftartdatei befinden. Dazu gehören die Notenköpfe, Versetzungszeichen, Text, Schlüssel, Taktbezeichnung, Dynamik und Gesangstext. Ihre Größe wird verändert, indem die `font-size-` (Schriftgröße)-Eigenschaft geändert wird, wie wir bald sehen werden. Andere Layout-Objekte, wie Bögen – oder allgemein Strecker-Objekte – werden individuell gezeichnet, es gibt dazu also keine `font-size`, die mit ihnen verknüpft wäre. Weitere Eigenschaften wie die Länge von Hälsen und Taktlinien, Dicke von Balken und anderen Linien und der Abstand der Notenlinien voneinander müssen auf spezielle Weise verändert werden.

In unserem Ossia-Beispiel wollen wir zuerst die Schriftgröße verändern. Das ist auf zwei Arten möglich. Entweder wir ändern die Schriftgröße für jede Objektart mit einem eigenen Befehl, etwa:

```
\override NoteHead #'font-size = #-2
```
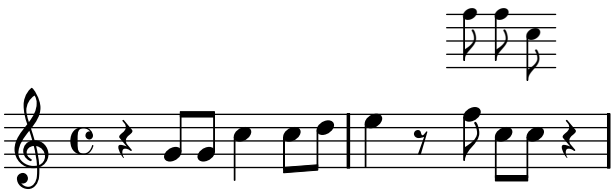
oder wir ändern die Größe aller Schriftobjekte, indem wir den Wert einer besonderen Eigenschaft, `fontSize`, mit dem `\set`-Befehl bestimmen oder sie in eine `\with`-Konstruktion (ohne `\set` einschließen.

```
\set fontSize = #-2
```

Beide Beispiele reduzieren die Schriftgröße um zwei Schritte im Vergleich zum vorigen Wert, wobei jeder Schritt die Schriftgröße um etwa 12% verändert.

Setzen wir das also in unserem Ossia-Beispiel ein:

```
\new Staff ="main" {
  \relative g' {
    r4 g8 g c4 c8 d |
    e4 r8
    <<
      { f c c }
      \new Staff \with {
        alignAboveContext = #"main"
        \override Clef #'stencil = ##f
        \override TimeSignature #'stencil = ##f
        % Alle Schriftgrößen um ~24% verkleinern
        fontSize = #-2
      }
        { f8 f c }
    >>
    r4 |
  }
}
```
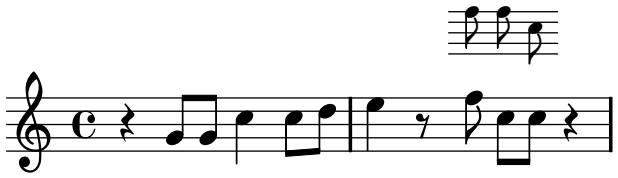


Das sieht aber immer noch nicht richtig aus. Die Notenköpfe und Fähnchen sind kleiner, aber die Hälse im Vergleich dazu zu lang und die Notenlinien zu weit auseinander. Sie müssen auch proportional zur Schriftart verkleinert werden. Der nächste Abschnitt behandelt diese Anpassung.

4.3.3 Länge und Dicke von Objekten

Abstände und Längen werden in LilyPond üblicherweise in Notenlinienabständen (engl. staff-spaces) gemessen. Das ist der Abstand zwischen zwei Notenlinien im System. Die meisten Dicken (engl. thickness) dagegen werden in einer internen Einheit Linien-Dicke (engl. line-thickness) gemessen. Die Linien von Dynamikklammern zum Beispiel haben standardmäßig eine Dicke von einer Einheit `line-thickness`, während die Dicke eines Notenhalses 1,3 ist. Beachten Sie jedoch, dass sich manche Dicken anders verhalten: die Dicke von Balken etwa wird in Notenlinienabständen gemessen.

Wie also werden Längen skaliert um der Schriftgröße zu entsprechen? Das kann mit einer besonderen Funktion `magstep` vorgenommen werden, die genau für diesen Zweck vorhanden ist. Sie nimmt ein Argument auf, die Änderung der Schriftgröße (#-2 im obigen Beispiel) und gibt einen Skalierungsfaktor aus, der dazu dient, Objekte proportionell zueinander zu verändern. So wird sie benutzt:

```
\new Staff ="main" {
  \relative g' {
    r4 g8 g c4 c8 d |
    e4 r8
    <<
      { f c c }
      \new Staff \with {
        alignAboveContext = #"main"
        \override Clef #'stencil = ##f
        \override TimeSignature #'stencil = ##f
        fontSize = #-2
        % Die Halslänge und Linienabstand anpassen
        \override StaffSymbol #'staff-space = #(magstep -2)
      }
        { f8 f c }
    >>
    r4 |
  }
}
```



Da die Länge eines Halses und viele andere Längeneigenschaften relativ zum Wert des Notenlinienabstands (`staff-space`) errechnet werden, werden sie auch automatisch verkleinert. Das wirkt sich jedoch nur auf die vertikale Skalierung des Ossias aus – die horizontale Skala ist durch das Layout des Haupsystems bestimmt und wird also von diesen Größenänderungen nicht betroffen. Wenn natürlich die Größe der gesamten Noten reduziert würde, würde sich auch die horizontalen Abstände ändern. Dass wird später im Layout-Abschnitt betrachtet.

Mit dieser Änderung ist unser Ossia fertig. Die Größen und Längen aller anderen Objekte können auf analoge Weise geändert werden.

Für kleine Größenänderungen, wie in dem obigen Beispiel, braucht die Dicke der verschiedenen Linien, wie Taktlinien, Notenlinien, Balken, Dynamikklammern usw. normalerweise keine spezielle Anpassung. Wenn die Dicke eines bestimmten Layout-Objektes angepasst werden

muss, kann man das erreichen, indem die entsprechende `thickness`-Eigenschaft des Objekts mit `\override` verändert wird. Ein Beispiel, wie man die Dicke von Bögen ändert, wurde schon gezeigt, siehe Abschnitt 4.2.1 [Eigenschaften von Layoutobjekten], Seite 91. Die Dicke aller gezeichneten Objekte (die also nicht aus einer Schriftart stammen) können auf gleiche Weise geändert werden.

4.4 Positionierung von Objekten

4.4.1 Automatisches Verhalten

Es gibt Objekte der Notation, die zum Notensystem gehören, und andere, die außerhalb des Systems gesetzt werden müssen. Sie werden `within-staff`-Objekte bzw. `outside-staff`-Objekte genannt.

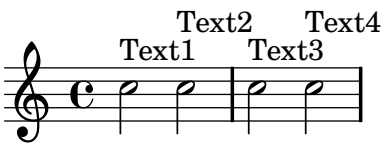
`within-staff`-Objekte werden innerhalb des Notensystems (engl. staff) gesetzt: Notenköpfe, Hälse, Versetzungszeichen usw. Ihre Position ist üblicherweise durch die notierte Musik bestimmt – sie werden vertikal auf bestimmten Linien notiert oder sind an andere Objekte gebunden, die vertikal festgelegt sind. Kollisionen von Notenköpfen, Hälsen und Versetzungszeichen werden normalerweise automatisch vermieden. Es gibt Befehle, um dieses automatische Verhalten zu verändern, wie unten gezeigt werden soll.

Objekte, die außerhalb des Notensystems gesetzt werden, sind unter Anderem Übungsmarkierungen, Text und Dynamikzeichen. LilyPonds Regel für ihre vertikale Positionierung lautet, sie so nah wie möglich am Notensystem zu setzen, aber nicht so nah, dass sie mit anderen Objekten kollidieren. Dabei wird die `outside-staff-priority`-(Priorität außerhalb des Notensystems)-Eigenschaft eingesetzt, um die Reihenfolge zu bestimmen, in denen Objekte gesetzt werden sollen.

Zuerst werden alle Innersystemobjekte von LilyPond gesetzt. Dann werden die Objekte außerhalb des Systems nach ihrer `outside-staff-priority` geordnet. Die `outside-staff`-Objekte werden dann nacheinander gesetzt, mit der niedrigsten Priorität beginnend, und so gesetzt, dass sie nicht mit anderen Objekten kollidieren, die schon gesetzt wurden. Wenn also zwei `outside-staff`-Objekte um den selben Platz streiten, wird das mit der geringeren `outside-staff-priority` näher am System gesetzt werden. Wenn zwei Objekte die selbe Priorität haben, wird das näher am System gesetzt, welches zuerst auftritt.

Im folgenden Beispiel haben alle Textbeschriftungen die gleiche Priorität (weil sie nicht explizit gesetzt worden ist). Beachten Sie, dass „Text3“ wieder dicht am System gesetzt wurde, weil er unter „Text2“ passt.

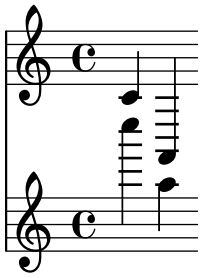
```
c2^"Text1"
c^"Text2"
c^"Text3"
c^"Text4"
```



Notensysteme werden in den Standardeinstellungen auch so dicht beeinander gesetzt wie es möglich ist (mit einem minimalen Abstand). Wenn Noten sehr weit aus einem System herausragen, zwingen sie das nächste System weiter weg, wenn eine Kollision drohen würde. Im nächsten Beispiel sehen Sie, wie Noten auf zwei Systemen „ineinander greifen“.

```
<<
  \new Staff {
```

```
    \relative c' { c a, }
  }
  \new Staff {
    \relative c'''' { c a, }
  }
>>
```
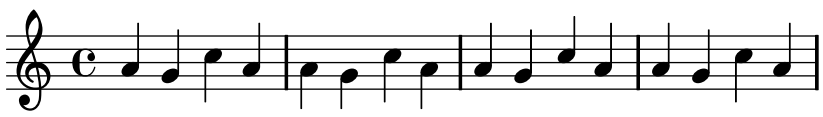


4.4.2 within-staff (Objekte innerhalb des Notensystems)

Es wurde schon gezeigt, wie die Befehle `\voiceXXX` die Richtung von Bögen, Fingersatz und allen anderen Objekten beeinflusst, die von der Richtung der Notenhälsen abhängen. Diese Befehle sind nötig, wenn polyphone Musik geschrieben wird, damit sich die einzelnen Melodielinien klar abzeichnen. Es kann aber von Zeit zu Zeit nötig sein, dieses automatische Verhalten zu verändern. Das kann entweder für ganze Abschnitte, aber genauso auch nur für eine einzelne Note vorgenommen werden. Die Eigenschaft, die die Richtung bestimmt, ist die `direction`-Eigenschaft jedes Layout-Objekts. Es soll erst erklärt werden, was sie bewirkt und dann eine Anzahl an fertigen Befehlen für die üblicheren Situationen präsentiert werden, mit denen Sie gleich loslegen können.

Manche Layout-Objekte, wie Legato- und Bindebögen, biegen sich oder zeigen entweder nach oben oder nach unten, andere, wie Hälse und Fähnchen, verändern auch die Position rechts oder links, je nach der Richtung, in die sie zeigen. Das wird automatisch berücksichtigt, wenn die `direction`-Eigenschaft verändert wird.

Das folgende Beispiel zeigt im ersten Takt die Standardeinstellung für Hälse, die bei hohen Noten nach unten zeigen und bei tiefen noten nach oben. Im nächsten Takt werden alle Hälse nach unten gezwungen, im dritten Takt nach oben, und im vierten wird wieder der Standard eingestellt.

```
a4 g c a
\override Stem #'direction = #DOWN
a g c a
\override Stem #'direction = #UP
a g c a
\revert Stem #'direction
a g c a
```



Hier werden die Konstanten `DOWN` und `UP` eingesetzt. Sie haben die Werte `-1` bwz. `+1`, und diese numerischen Werte können ebenso benutzt werden. Auch der Wert `0` kann in manchen Fällen benutzt werden. Er bedeutet für die Hälse das gleiche wie `UP`, für einige andere Objekte jedoch „zentiert“. Es gibt hierzu die Konstante `CENTER`, die den Wert `0` hat.

Es gibt aber einfachere Befehle, die normalerweise benutzt werden. Hier eine Tabelle der häufigsten. Die Bedeutung des Befehls wird erklärt, wenn sie nicht selbsverständlich ist.

Runter/Links	**Rauf/Rechts**	**Rückgängig**	**Wirkung**
`\arpeggioArrowDown`	`\arpeggioArrowUp`	`\arpeggioNormal`	Arpeggio mit Pfeil nach unten, oben oder ohne Pfeil
`\dotsDown`	`\dotsUp`	`\dotsNeutral`	Richtung der Verschiebung eines Punktes, um Notenlinien zu vermeiden
`\dynamicDown`	`\dynamicUp`	`\dynamicNeutral`	Position der Dynamik-Bezeichnung relativ zum System
`\phrasingSlurDown`	`\phrasingSlurUp`	`\phrasingSlurNeutral`	Befehl für Richtung von Phrasierungsbögen
`\slurDown`	`\slurUp`	`\slurNeutral`	Befehl für Richtung von Legatobögen
`\stemDown`	`\stemUp`	`\stemNeutral`	Befehl für Richtung von Hälsen
`\textSpannerDown`	`\textSpannerUp`	`\textSpannerNeutral`	Position von Textbeschrifungen, die als Strecker eingegeben werden
`\tieDown`	`\tieUp`	`\tieNeutral`	Befehl für Richtung von Bindebögen
`\tupletDown`	`\tupletUp`	`\tupletNeutral`	Befehl für Richtung von Klammern/Zahlen der N-tolen

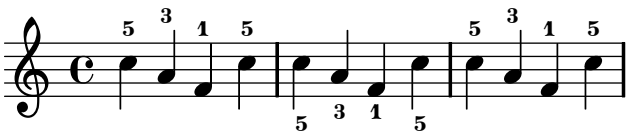
Diese vordefinierten Befehl können allerdings **nicht** zusammen mit `\once` benutzt werden. Wenn Sie die Wirkung eines Befehl auf eine einzige Noten begrenzen wollen, müssen Sie den entsprechenden `\once \override`-Befehl benutzen oder den definierten Befehl, gefolgt von dem entsprechenden neutralisierenden `xxxNeutral`-Befehl nach der Note.

Fingersatz

Die Positionierung von Fingersatz kann auch durch den Wert seiner `direction`-Eigenschaft beeinflusst werden, aber eine Veränderung von `direction` hat keinen Einfluss auf Akkorde. es gibt auch hier besondere Befehle, mit denen der Fingersatz von einzelnen Noten in Akkorden kontrolliert werden kann, wobei mögliche Positionen über, unter der Note und rechts bzw. links von ihr sind.

Zunächst die Wirkungsweise von `direction` auf den Fingersatz: im ersten Takt der Standard, dann die Wirkung von `DOWN` (runter) und `UP` (hinauf).

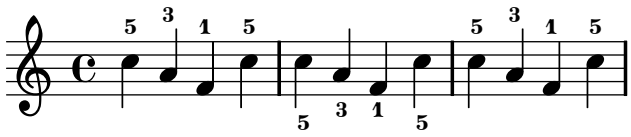
```
c-5 a-3 f-1 c'-5
\override Fingering #'direction = #DOWN
c-5 a-3 f-1 c'-5
\override Fingering #'direction = #UP
c-5 a-3 f-1 c'-5
```



Eine Beeinflussung der `direction`-Eigenschaft ist jedoch nicht die einfachste Art, Fingersatzbezeichnungen manuell über oder unter das System zu setzen. Normalerweise bietet es sich

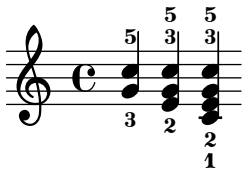
an, _ oder ^ anstelle von - vor der Fingersatz-Zahl zu benutzen. Hier das vorherigen Beispiel mit dieser Methode:

```
c-5 a-3 f-1 c'-5
c_5 a_3 f_1 c'_5
c^5 a^3 f^1 c'^5
```



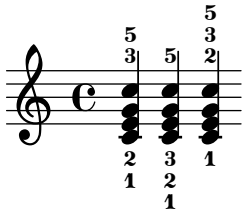
Die `direction`-Eigenschaft wirkt sich nicht auf Akkorde aus, während die Präfixe _ und ^ funktionieren. Standardmäßig wird der Fingersatz automatisch entweder über oder unter dem Akkord gesetzt:

```
<c-5 g-3>
<c-5 g-3 e-2>
<c-5 g-3 e-2 c-1>
```



aber das kann manuell geändert werden, um einzelne Fingersatzanweisungen nach oben oder unten zu zwingen:

```
<c-5 g-3 e-2 c-1>
<c^5 g_3 e_2 c_1>
<c^5 g^3 e^2 c_1>
```



Noch bessere Kontrolle über die Positionierung von Fingersatz für einzelne Noten in einem Akkord ist mit dem `\set fingeringOrientations`-Befehl möglich. Die Syntax lautet:

```
\set fingeringOrientations = #'([up] [left/right] [down])
```

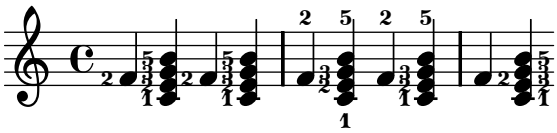
`\set`wird benutzt, weil `fingeringOrientations` eine Eigenschaft des `Voice`-Kontextes ist, erstellt und eingesetzt vom `New_fingering_engraver`.

Die Eigenschaft kann als Wert eine Liste mit einem bis drei Einträgen haben. Damit wird bestimmt, ob Fingersatz oberhalb gesetzt werden kann (wenn `up` in der Liste auftaucht), darunter (wenn`down` auftaucht), links (wenn `left` auftaucht) oder rechts (wenn `right` auftaucht). Wenn andererseits ein Wert nicht auftaucht, wir auch kein Fingersatz in dieser Richtung gesetzt. LilyPond nimmt diese Beschränkung als Bedingung und errechnet die besten Positionen für die Noten des nächsten Akkordes. Die seitliche Positionierung kann nur auf einer Seite des Akkordes geschehen, nicht auf beiden gleichzeitig, es kann also nur entweder `left` oder `right` auftreten, nicth beide gleichzeitig.

> **Achtung:** Damit eine einzelne Note mit diesem Befehl beeinflusst werden kann, muss sie als ein „Ein-Noten-Akkord“ geschrieben werden, indem einfache spitze Klammern um die Note positioniert werden.

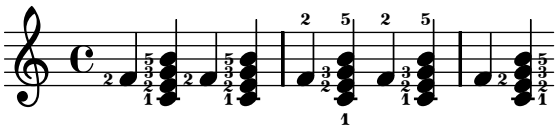
Hier ein paar Beispiele:

```
\set fingeringOrientations = #'(left)
<f-2>
< c-1  e-2 g-3 b-5 > 4
\set fingeringOrientations = #'(left)
<f-2>
< c-1  e-2 g-3 b-5 > 4
\set fingeringOrientations = #'(up left down)
<f-2>
< c-1  e-2 g-3 b-5 > 4
\set fingeringOrientations = #'(up left)
<f-2>
< c-1  e-2 g-3 b-5 > 4
\set fingeringOrientations = #'(right)
<f-2>
< c-1  e-2 g-3 b-5 > 4
```



Wenn die Fingersatzbezeichnung zu gedrungen aussieht, kann auch die Schriftgröße (`font-size`) verringert werden. Der Standardwert kann aus dem `Fingering`-Objekt in der IR entnommen werden, er ist `-5`, versuchen wir es also mit `-7`.

```
\override Fingering #'font-size = #-7
\set fingeringOrientations = #'(left)
<f-2>
< c-1  e-2 g-3 b-5 > 4
\set fingeringOrientations = #'(left)
<f-2>
< c-1  e-2 g-3 b-5 > 4
\set fingeringOrientations = #'(up left down)
<f-2>
< c-1  e-2 g-3 b-5 > 4
\set fingeringOrientations = #'(up left)
<f-2>
< c-1  e-2 g-3 b-5 > 4
\set fingeringOrientations = #'(right)
<f-2>
< c-1  e-2 g-3 b-5 > 4
```



4.4.3 Objekte außerhalb des Notensystems

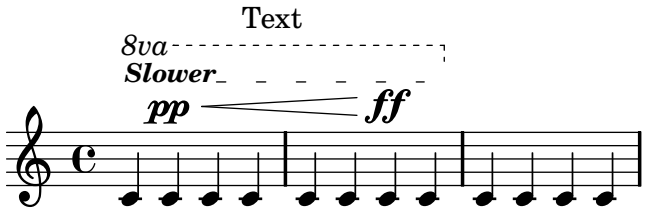
Objekte außerhalb des Notensystems werden automatisch gesetzt, um Kollisionen zu vermeiden. Objekten mit einem geringeren Prioritätswert der Eigenschaft `outside-staff-priority` werden

näher an das System gesetzt, und andere Objekte außerhalb des Systems werden dann soweit vom System entfernt gesetzt, dass Zusammenstöße vermieden werden. Die `outside-staff-priority`-Eigenschaft ist im `grob-interface` definiert und ist also eine Eigenschaft von allen Layout-Objekten. Standardmäßig ist sie für alle Objekte auf falsch (`#f`) gesetzt; dieser Wert wird in einen numerischen Wert dem Objekt entsprechend geändert, wenn das Objekt für die Notenausgabe erstellt wird. Die Tabelle unten zeigt die Standardwerte für die meistbenutzten `outside-staff`-Objekte, die den Voreinstellungen nach im `Staff`- oder `Voice`-Kontext gesetzt werden.

Layout-Objekt	**Priorität**	**Kontrolliert Position von:**
`MultiMeasureRestText`	`450`	Text über Ganztaktpausen
`TextScript`	`450`	Textbeschriftung
`OttavaBracket`	`400`	Ottava (Oktavierungsklammern)
`TextSpanner`	`350`	Text-Strecker
`DynamicLineSpanner`	`250`	Alle Dynamik-Bezeichnungen
`VoltaBracketSpanner`	`100`	Volta-Klammern
`TrillSpanner`	`50`	Triller-Strecker

Hier ein Beispiel, das die Standardpositionierung von einigen Objekten zeigt.

```
% Details für späteren Text-Spanner setzen
\override TextSpanner #'(bound-details left text)
    = \markup { \small \bold Slower }
% Dynamik-Zeichen über System setzen
\dynamicUp
% Beginn der Oktavierungsklammer
\ottava #1
c' \startTextSpan
% Dynamik-Text hinzufügen
c\pp
% Dynamic Line Spanner hinzufügen
c\<
% Textbeschriftung hinzufügen
c^Text
c c
% Dynamik-Text hinzufügen
c\ff c \stopTextSpan
% Ende der Oktavierungsklammer
\ottava #0
c, c c c
```



Dieses Beispiel zeigt auch, wie man Text-Strecker erstellt, d.h. Text mit Bindestrichen, der sich über eine bestimmte Länge erstreckt. Der Strecker beginnt mit dem `\startTextSpan`-Befehl

und endet mit dem `\stopTextSpan`-Befehl, und das Format des Textes wird mit dem `\override TextSpanner`-Befehl bestimmt. Mehr Einzelheiten siehe Abschnitt "Text mit Verbindungslinien" in *Notationsreferenz*.

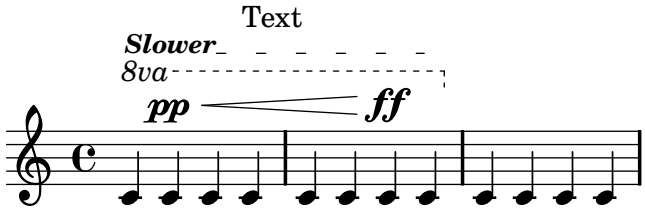
Im Beispiel wird auch gezeigt, wie Oktavierungsklammern (Ottava) erstellt werden.

Beachten Sie, dass Taktnummern, Metronombezeichnungen und Übungszeichen nicht gezeigt werden. Sie werden standardmäßig im `Score`-(Partitur)-Kontext erstellt und ihre `outside-staff-priority` wird in Bezug auf die Layout-Objekte, die im `Staff`-Kontext erstellt werden, ignoriert. Wenn Sie Taktnummern, Metronombezeichnungen oder Übungszeichen entsprechend ihrer Außersystem-Priorität setzen wollen, müssen Sie die entsprechenden Engraver (`Bar_number_engraver`, `Metronome_mark_engraver` oder `Mark_engraver`) vom `Score`-Kontext entfernen und dem `Staff`-Kontext hinzufügen. Wenn die Engraver so geändert werden, erhalten sie folgenden Werte für `outside-staff-priority`:

Layout-Objekt	**Priorität**
`RehearsalMark`	1500
`MetronomeMark`	1000
`BarNumber`	100

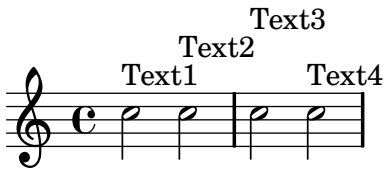
Wenn die Standardwerte der `outside-staff-priority` nicht die Positionierung hervorrufen, die Sie wünschen, kann die Priorität eines jeden Objektes geändert werden. Als Beispiel wollen wir zeigen, wie sich die Oktavierungsklammer unter den Textstrecker des vorigen Beispiels setzen lässt. Wir müssen nur die Priorität des `OttavaBracket`-Objektes in der IR oder der Tabelle oben herausfinden und einen kleineren Wert angeben als der Wert, den das `TextSpanner`-(Strecker)-Objekt bekommt, wobei noch daran zu denken ist, dass `OttavaBracket` im `Staff`-Kontext erstellt wird:

```
% Details für späteren Text-Spanner setzen
\override TextSpanner #'(bound-details left text)
    = \markup { \small \bold Slower }
% Dynamik-Zeichen über System setzen
\dynamicUp
%Nächste Ottava-Klammer unter Text-Spanner setzen
\once \override Staff.OttavaBracket #'outside-staff-priority = #340
% Beginn der Oktavierungsklammer
\ottava #1
c' \startTextSpan
% Dynamik-Text hinzufügen
c\pp
% Dynamic Line Spanner hinzufügen
c\<
% Textbeschriftung hinzufügen
c^Text
c c
% Dynamik-Text hinzufügen
c\ff c \stopTextSpan
% Ende der Oktavierungsklammer
\ottava #0
c, c c c
```



Eine Änderung der `outside-staff-priority` kann auch dazu benutzt werden, die vertikale Plazierung von individuellen Objekten zu kontrollieren, auch wenn das Ergebnis nicht immer optimal ist. Im nächsten Beispiel soll „Text3“ oberhalb von „Text4“ gesetzt werden, das Beispiel wurde behandelt in Abschnitt 4.4.1 [Automatisches Verhalten], Seite 106. Der Wert der Priorität muss also für die Eigenschaft `TextScript` entweder in der IR oder in der Tabelle oben festgestellt werden und dann die Priorität für „Text3“ höher eingestellt werden:

```
c2^"Text1"
c^"Text2"
\once \override TextScript #'outside-staff-priority = #500
c^"Text3"
c^"Text4"
```
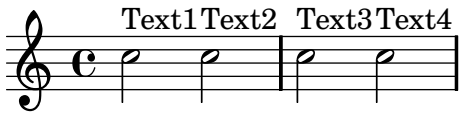


Damit wird zwar „Text3“ ganz richtig über „Text4“ platziert, aber auch über „Text2“, und „Text4“ wird jetzt weiter unten gesetzt. Eigentlich sollten ja alle diese Anmerkungen gleichweit vom System entfernt sein. Dazu muss offensichtlich horizontal etwas Platz gemacht werden. Das kann erreicht werden mit dem `textLengthOn`-(Textlänge an)-Befehl.

\textLengthOn (Textlänge berücksichtigen)

Standardmäßig wird Text, der mit dem Beschriftungsbefehl `\markup` bzw. Äquivalenten erstellt wird, kein zusätzlicher Platz in Bezug auf die Positionierung der Noten zugestanden. Der `\textLengthOn`-Befehl ändert dieses Verhalten, so dass die Noten gespreizt werden, wenn die Breite des Textes es erfordert:

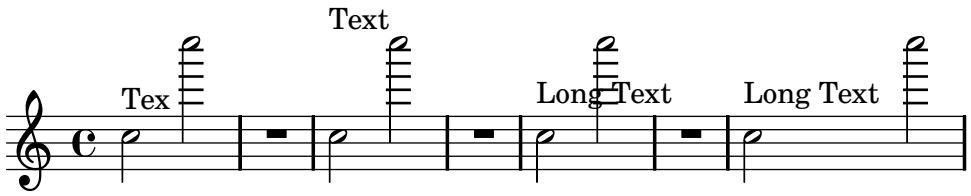
```
\textLengthOn  % Noten spreizen um dem Text Platz zu machen
c2^"Text1"
c^"Text2"
c^"Text3"
c^"Text4"
```



Dieses Verhalten wird mit dem `\textLengthOff`-Befehl rückgängig gemacht. Erinnern Sie sich, dass `\once` nur mit `\override`, `\set`, `\revert` oder `unset` funktioniert, der Befehl kann also nicht zusammen mit `\textLengthOn` benutzt werden.

Textbeschriftung vermeidet auch Noten, die über das System hinausstehen. Wenn das nicht gewünscht ist, kann die automatische Verschiebung nach oben hin auch vollständig ausgeschaltet werden, indem die Priorität auf `#f` gesetzt wird. Hier ein Beispiel, wie eine Textbeschriftung mit diesen Noten reagiert:

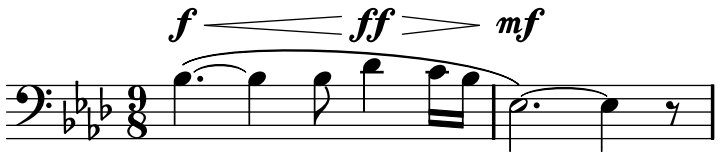
```
% Dieser Text ist kurz genug um ohne Kollision zu passen
c2^"Tex"
c''2
R1
% Dieser ist zu lang, darum wird der nach oben verschoben
c,,2^"Text"
c''2
R1
% Abschalten der automatischen Vermeidung von Zusammenstößen
\once \override TextScript #'outside-staff-priority = ##f
c,,2^"Long Text   "
c''2
R1
% Abschalten der automatischen Vermeidung von Zusammenstößen
\once \override TextScript #'outside-staff-priority = ##f
\textLengthOn  % und Textlänge berücksichtigen
c,,2^"Long Text   "  % Leerzeichen am Ende werden beachtet
c''2
```



Dynamik

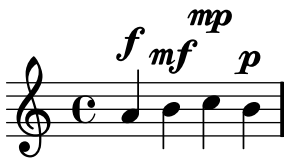
Dynamikbezeichnung wird üblicherweise unter dem System gesetzt, kann aber auch nach oben mit dem Befehl `dynamicUp` gezwungen werden. Die Bezeichnung wird vertikal relativ zu der Note positioniert, an die sie angefügt wurde. Sie wird vertikal variabel gesetzt in Bezug zu Innersystemobjekten wie Bögen oder Taktnummern. Damit können oft recht gute Resultate erreicht werden, wie im folgenden Beispiel:

```
\clef "bass"
\key aes \major
\time 9/8
\dynamicUp
bes4.~\f\< \( bes4 bes8 des4\ff\> c16 bes\! |
ees,2.~\)\mf ees4 r8 |
```



Wenn aber Noten und Dynamikzeichen sehr dicht beieinander stehen, positioniert die automatische Kollisionsvermeidung später kommende Dynamikzeichen weiter weg, was allerdings nicht immer die beste Möglichkeit ist, wie in dem folgenden, etwas gewollten Beispiel zu sehen ist:

```
\dynamicUp
a4\f b\mf c\mp b\p
```



Wenn eine ähnliche Situation in „echter“ Musik auftaucht, kann es nötig sein, die Noten etwas zu spreizen, damit die Dynamikzeichen alle auf der selben vertikalen Position gesetzt werden können. Dieses Verhalten war im Falle von Textbeschriftungen möglich mit dem `\textLengthOn`-Befehl, aber es gibt keinen entsprechenden Befehl für Dynamik. Wir müssen also unsere eigenen Befehle mit `\override` konstruieren.

Verändern der Größe von grobs
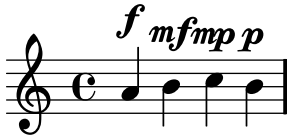
Zuallererst müssen wir lernen, wie die Größe von Grobs verändert wird. Alle Grobs besitzen einen Referenzpunkt, der benutzt wird, um ihre Position in Relation zu ihnen übergeordneten Objekten zu bestimmen. Dieser Punkt innerhalb des Grobs wird dann auf einer horizontalen Position (`X-offset`) und einer vertikalen Position (`Y-offset`) ausgerichtet, immer bezüglich des übergeordneten Objektes. Eine horizontale Strecke wird durch ein Zahlenpaar angegeben (`X-extent`), welche die linke und rechte Grenze relativ zum übergeordneten Objekt bezeichnen. Die vertikale Strecke wir genauso durch ein Zahlenpaar (`Y-extent`) definiert. Diese Eigenschaften gehören zu allen Grobs, die das `grob-interface` unterstützen.

Standardmäßig haben Außersystemobjekte eine Länge von Null, so dass sie sich in horizontaler Richtung überlappen können. Das geschieht, indem dem linken Rand Unendlich zugewiesen wird und dem rechten Rand minus Undendlich (der Code der `extra-spacing-width`-(zusätzliche Positionierungslänge)-Eigenschaft lautet: `'(+inf.0 . -inf.0)`). Damit sich diese Objekte also horizontal nicht überschneiden, muss der Wert von `extra-spacing-width` auf `'(0 . 0)` gesetzt werden, sodass die wirkliche Länge eines Objektes zur Geltung kommt. Mit diesem Befehl wird das für Dynamik-Zeichen erledigt:

```
\override DynamicText #'extra-spacing-width = #'(0 . 0)
```
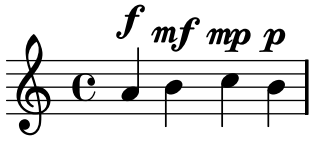
Schauen wir uns an, wie es mit dem vorigen Beispiel funktioniert:

```
\dynamicUp
\override DynamicText #'extra-spacing-width = #'(0 . 0)
a4\f b\mf c\mp b\p
```



Damit werden die Dynamik-Zeichen also wirklich nebeneinander gesetzt, aber es gibt noch zwei Probleme. Die Zeichen sollten etwas weiter außeinander stehen und es wäre gut, wenn sie alle den gleichen Abstand zum System hätte. Das erste Problem ist einfach behoben. Anstatt der `extra-spacing-width`-Eigenschaft Null zuzuweisen, können wir auch einen etwas größeren Wert wählen. Die Einheit wird gemessen in dem Abstand zwischen zwei Notenlinien, es scheint also gut, den rechten und linken Rand eine halbe Einheit zu vergrößern:

```
\dynamicUp
% Breite um einen Linienabstand vergrößern
\override DynamicText #'extra-spacing-width = #'(-0.5 . 0.5)
a4\f b\mf c\mp b\p
```



Das sieht schon besser aus, aber es wäre noch besser, wenn die Dynamik-Zeichen alle an einer Linie ausgerichtet wären, anstatt höher und tiefer zu sitzen. Das kann mit der `staff-padding`-Eigenschaft erreicht werden, die wir uns im folgenden Abschnitt genauer anschauen werden.

4.5 Kollision von Objekten

4.5.1 Verschieben von Objekten

Es wird vielleicht eine Überraschung sein, aber LilyPond ist nicht perfekt. Einige Notationselemente können sich überschneiden. Das ist nicht schön, aber zum Glück sehr selten. Normalerweise müssen die Objekte zur Klarheit oder aus ästhetischen Gründen verschoben werden – sie könnten besser aussehen, wenn sie etwas zusätzlichen Platz erhalten.

Es gibt im Grunde drei Herangehensweisen, überlappende Notation zu verbessern. Man sollte sie in der folgenden Reihenfolge anwenden:

1. Die **Richtung** eines der überlappenden Objekte kann geändert werden, indem die vordefinierten Befehle für Innersystemobjekte verwendet werden, wie beschrieben in Abschnitt 4.4.2 [within-staff (Objekte innerhalb des Notensystems)], Seite 107. Hälse, Bögen, Balken, Dynamik-Zeichen und Triolen können auf diese Weise einfach umgeordnet werden. Beschränkt ist diese Methode insofern, als es nur zwei Möglichkeiten zur Veränderung gibt: oben oder unten.
2. Die **Objekteigenschaft**, die LilyPond benutzt um die Layout-Objekte zu platzieren, können mit dem `\override`-Befehl positioniert werden. Die Vorteile von Änderungen dieser Art sind a) dass einige Objekte automatisch verschoben werden, wenn es nötig ist Platz zu schaffen und b) ein einziges `\override` sich auf alle Fälle eines Objekttyps bezieht. Zu diesen Eigenschaften gehören:
   - `direction` (Richtung)

     Das wurde schon detailliert behandelt, siehe Abschnitt 4.4.2 [within-staff (Objekte innerhalb des Notensystems)], Seite 107.
   - `padding`, `left-padding`, `right-padding`, `staff-padding` (Verschiebung)

     Wenn ein Objekt platziert wird, bestimmt der Wert seiner `padding`-(Füllungs)-Eigenschaft die Größe des Abstandes, der zwischen dem Objekt selber und dem Objekt, relativ zu welchem es positioniert wird, gelassen werden muss. Dabei zählt der `padding`-Wert des Objektes, das platziert werden soll, der `padding`-Wert des Objektes, das schon gesetzt wurde, wird hingegegen ignoriert. Abstände mit `padding` können zu allen Objekten hinzugefügt werden, die das `side-position-interface` unterstützen.

     Anstelle von `padding` wird die Position von Versetzungszeichengruppen durch die Eigenschaften `left-padding` und `right-padding` bestimmt. Diese Eigenschaften werden im `AccidentalPlacement`-(Versetzungszeichen-Positionierungs)-Objekt gefunden, das sich innerhalb des **Staff**-Kontexts befindet. Während des Notensatzes werden die Notenköpfe zuerst gesetzt und dann die Versetzungszeichen, wenn denn welche gesetzt werden, durch die `right-padding`-Eigenschaft auf die linke Seite der Notenköpfe positioniert, um die Entfernung von den Notenköpfen zu bestimmen. Also nur die `right-padding`-(Verschiebung nach rechts)-Eigenschaft des `AccidentalPlacement`-Objekts hat Einfluss auf die Positionierung der Versetzungszeichen.

     Die `staff-padding`-(Verschiebung zum System)-Eigenschaft ist sehr ähnlich wie die `padding`-Eigenschaft: `padding` bestimmt den Minimalabstand zwischen einem Objekt, das das `side-position-interface` unterstützt, und dem nächsten anderen Objekt (normalerweise die Note oder Notenlinie); `staff-padding` dagegen wirkt nur auf Objekte die immer außerhalb des Notensystems sind – damit wird der minimale Abstand bestimmt, der zwischen dem Objekt und dem Notensystem gelassen werden soll. `staff-padding` hat also **keinen Einfluss** auf Objekte, die relativ zu einer Note positioniert

werden, sondern nur auf solche, die zum System relativ stehen. Wenn es mit einem anderen Objekt eingesetzt wird, erhält man keine Fehlermeldung, aber der Befehl hat auch keine Auswirkungen.

Um herauszufinden, welche `padding`-Eigenschaft für das bestimmte Objekt nötig, ist, das Sie verschieben wollen, müssen Sie in der IR nach den Objekt-Eigenschaften schauen. Dabei sollten Sie bedenken, dass sich die `padding`-Eigenschaften nicht unbedingt in dem Objekt selber befinden, schauen Sie also auch in Objekten nach, die offensichtlich Ähnlichkeiten haben.

Alle `padding`-Werte werden in Notenlinienabständen gemessen. Für die meisten Objekte ist der Wert ungefähr auf 1.0 oder weniger gesetzt (das variiert von Objekt zu Objekt). Der Wert kann geändert werden, wenn ein größerer (oder kleinerer) Abstand gewünscht wird.

- `self-alignment-X` (Selbstpositionierung)

  Diese Eigenschaft kann benutzt werden, um ein Objekt nach links, rechts oder zentriert an dem Referenzpunkt des Objekts auszurichten, an das es verknüpft ist. Es kann bei allen Objekten benutzt werden, die das `self-alignment-interface` unterstützen. Das sind üblicherweise Objekte, die Text enthalten. Die möglichen Werte der Eigenschaft sind `LEFT`, `RIGHT` oder `CENTER`. Alternativ kann ein numerischer Wert zwischen `-1` und `+1` bestimmt werden: `-1` heißt linksbündig, `+1` rechtsbündig und Zahlen dazwischen bewegen den Text schrittweise von links nach rechts. Zahlen größer als `1` können angegeben werdne, um den Text noch weiter nach links zu bewegen, oder weniger als `-1`, um ihn weiter nach rechts zu schieben. Eine Änderung um `1` des Wertes entspricht einer Bewegung um die halbe Textbreite.

- `extra-spacing-width` (zusätzliche Breite)

  Diese Eigenschaft steht für alle Objekte zur Verfügung, die das `item-interface` unterstützen. Es braucht zwei Zahlen als Argument, die erste wird zur rechten Ausdehnung, die zweite zur linken Ausdehnung hinzugerechnet. Negative Zahlen verschieben die Ausdehnung nach rechts, positive nach links, um also ein Objekt zu verbreitern, muss die erste Zahl negativ und die zweite positiv sein. Allerdings beachten nicht alle Objekte beide Zahlen. Das `accidental`-(Versetzungszeichen)-Objekt etwa beachtet nur erste Zahl für die linke Ausdehnung.

- `staff-position` (Notensystempositionierung)

  `staff-position` ist eine Eigenschaft des `staff-symbol-referencer-interface`, die von Objekten unterstützt wird, die relativ zum Notensystem (engl. staff) positioniert werden. Hiermit wird die vertikale Position eines Objekts relativ zur Mittellinie des Systems in halben Notenlinienabständen angegeben. Das ist sehr nützlich, um Zusammenstöße zwischen Layout-Objekten wie Ganztaktpausen, Bögen und Noten in verschiedenen Stimmen zu lösen.

- `force-hshift` (vertikale Verschiebung erzwingen)

  Eng beeinander stehende Noten in einem Akkord oder Noten, die zum gleichen Zeitpunkt in unterschiedlichen Stimmen stehen, werden in zwei oder manchmal auch mehr Kolumnen gesetzt, um Kollisionen zu umgehen. Diese Kolumnen werden Notenkolumnen genannt; ein `NoteColumn`-Objekt wird erstellt um die Noten in den Kolumnen zu setzen.

  Die `force-hshift`-(erzwinge horizontale Verschiebung)-Eigenschaft ist eine Eigenschaft von `NoteColumn` (bzw. vom `note-column-interface`). Eine Veränderung dieser Eigenschaft macht es möglich, eine Notenkolumne zu verschieben, dabei gilt als Einheit die Breite einer Kolumne, also die Breite des Notenkopfes der ersten Stimme. Diese Eigenschaft kann in Situationen benutzt werden, in denen die normalen `\shiftOn`-Befehle (siehe auch Abschnitt 3.2.2 [Stimmen explizit beginnen], Seite 54) das Prob-

lem nicht beseitigen. Diese Eigenschaft ist besser in solchen Fällen zu verwenden als die `extra-offset`-Eigenschaft, weil man die richtige Entfernung nicht in Notenlinienabständen ausrechnen muss. Wenn eine Note in eine Notenkolumne oder aus ihr heraus geschoben wird, werden auch andere Funktionen beeinflusst, wie etwa die Verschmelzung von Notenköpfen.

3. Zu guter Letzt, wenn alles andere nicht funktioniert, können Objekte auch manuell positioniert werden, entweder vertikal in Bezug auf die Mittellinie des Systems, oder indem sie einen beliebigen Abstand weit auf eine neue Position verschoben werden. Der Nachteil ist, dass die richtigen Werte für eine gute Position manuell ausprobiert werden müssen, meistens durch Herantasten an den richtigen Wert, und das für jedes einzelne Objekt extra. Und weil diese Verschiebungen erst vorgenommen werden, wenn LilyPond alle anderen Objekte gesetzt hat, ist man als Notensetzer selber dafür verantwortlich, ob es Zusammenstöße gibt. Am schwerwiegendsten ist aber die Tatsache, dass die Verschiebungskoordinaten wahrscheinlich neu errechnet oder ausprobiert werden müssen, wenn sich an den Noten und deren Layout später irgend etwas ändert. Die Eigenschaften, die für diese Arte der manuellen Verschiebung verwendet werden können, sind:

   `extra-offset (zusätzlicher Abstand)`
   : Diese Eigenschaft gehört zu jedem Layout-Objekt, das das `grob-interface` unterstützt. Sie braucht ein Zahlenpaar, das die exakte Verschiebung in horizontaler und vertikaler Richtung bezeichnet. Negative Zahlen verschieben das Objekt nach links oder unten. Die Einheit sind Notenlinienabstände. Die zusätzliche Positionierung wird vorgenommen, nachdem alle anderen Objekte platziert sind, weshalb ein Objekt irgendwohin verschoben werden kann, ohne den restlichen Satz zu beeinflussen.

   `positions (Position)`
   : Diese Eigenschaft ist am sinnvollsten, um die Steigung und die Höhe von Balken, Bögen und Triolenklammern anzupassen. Sie braucht ein Zahlenpaar, das die Position des rechten und linken Endes relativ zur Mittellinie des Notensystems bestimmt. Die Einheit sind Notenlinienabstände. Bögen allerdings können nicht beliebig weit weg positioniert werden. LilyPond erstellt zunächst eine Liste an möglichen Positionen für den Bogen und findet normalerweise die Version, die „am besten aussieht“. Wenn die `positions`-Eigenschaft verändert worden ist, wird der Bogen aus der Liste gewählt, der der gewünschten Position am nächsten kommt.

Ein bestimmtes Objekt hat vielleicht nicht alle dieser Eigenschaften. Darum ist es nötig, in der IR nachzuschlagen, welche Eigenschaften ein bestimmtes Objekt unterstützt.

Hier ist eine Liste an Objekten, die am wahrscheinlichsten an einer Kollision beteiligt sind, daneben findet sich die Bezeichnung des Objektes, mit der Sie es in der IR finden, um zu bestimmen, welche Eigenschaften benutzt werden können, um es zu verschieben.

Objekttyp	**Objektbezeichnung**
Articulationszeichen	`Script`
Balken	`Beam`
Dynamikzeichen (vertikal)	`DynamicLineSpanner`
Dynamikzeichen (horizontal)	`DynamicText`
Fingersatz	`Fingering`
Übungs-/Textmarken	`RehearsalMark`
Legatobögen	`Slur`
Text z. B. `^"text"`	`TextScript`
Bindebögen	`Tie`

N-tolen TupletBracket

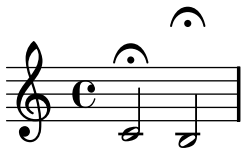
4.5.2 Überlappende Notation in Ordnung bringen

Hier soll nun gezeigt werden, wie die Eigenschaften, die im vorigen Abschnitt vorgestellt wurden, bei der Problemlösung mit sich überschneidenden Notationselementen eingesetzt werden können.

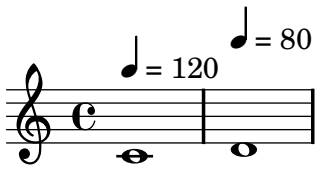
padding (Fülleigenschafte)

Die `padding`-(Verschiebungs-)Eigenschaft kann benutzt werden, um den Abstand zwischen Symbolen zu vergößern (oder zu verkleinern), die über oder unter den Noten gesetzt werden.

```
c2\fermata
\override Script #'padding = #3
b2\fermata
```



```
% Das funktioniert nicht, siehe unten
\override MetronomeMark #'padding = #3
\tempo 4=120
c1
% Das funktioniert:
\override Score.MetronomeMark #'padding = #3
\tempo 4=80
d1
```



Im zweiten Beispiel können Sie sehen, wie wichtig es ist den richtigen Kontext anzugeben. Weil das `MetronomeMark`-Objekt sich im `Score`-Kontext befindet, werden Eigenschaftsänderungen im `Voice`-Kontext einfach ignoriert. Für mehr Einzelheiten siehe Abschnitt “Eignschaften verändern” in *Notationsreferenz*.
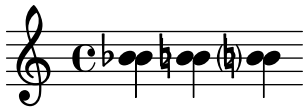
Wenn die `padding`-Eigenschaft eines Objektes erhöht wird, das sich in einem Stapel von Objekten befindet, die nach ihrer Außersystempriorität (`outside-staff-priority`) positioniert werden, werden das Objekt und alle, die sich außerhalb davon befinden, entsprechend verschoben.

left-padding (Verschieben nach rechts) und right-padding (Verschieben nach links)

Die `right-padding`-Eigenschaft wirkt sich auf den Abstand zwischen einem Versetzungszeichen und der Note, auf das sie sich bezieht, aus. Sie wird nicht sehr oft benötigt, aber das folgende Beispiel zeigt eine Situation, wo man sie braucht. Das Beispiel stellt eine Situation dar, in der in einem Akkord sowohl H als auch B vorkommen sollen. Damit keine Ambiguität ensteht, sollen beide Noten ein Zeichen haben, also ein B und ein Auflösungszeichen. Hier einige Notationsversuche:
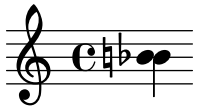
```
<b bes>
<b! bes>
```

```
<b? bes>
```



Keiner davon funktioniert, und der zweite und dritte weist hässliche Zusammenstöße zwischen den Zeichen auf.

Eine Möglichkeit, das Gewünschte zu erhalten, ist es den Stencil des Versetzungszeichens zu ersetzen mit einer Textbeschriftung (Markup), die sowohl das B als auch das Aulösungszeichen enthält:

```
AuflösungB = \markup { \natural \flat }
\relative c'' {
  \once \override Accidental
    #'stencil = #ly:text-interface::print
  \once \override Accidental #'text = #AuflösungB
  \once \override Score.AccidentalPlacement #'right-padding = #1.5
  <b bes>
}
```
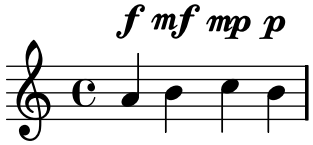


Dazu ist aber ein `\override`-Befehl für den Stencil des Versetzungszeichens nötig, der bisher nicht behandelt wurde. Der Typ des Stencils muss eine Prozedur sein, die hier geändert wurde, um den Inhalt der `text`-Eigenschaft des `Accidental` (Versetzungszeichen)-Objekts zu setzen, die dann so definiert wird, dass sie ein Auflösungszeichen gefolgt von einem B enthält. Diese werden dann mit `right-padding` weiter nach rechts verschoben.

staff-padding (Systemfüllungseigenschaft)

`staff-padding` (Verschiebung zum Notensystem) kann verwendet werden um Objekte wie Dynamikzeichen an einer Grundlinie auf einer bestimmten Höhe über dem System auszurichten, sodass sie nicht von der Position der Note abhängen, an die sie angehängt sind. Diese Verschiebung ist keine Eigenschaft von `DynamicText`, sondern von `DynamicLineSpanner`. Das liegt daran, dass die Grundlinie sich gleicherweise auf **alle** Dynamikzeichen beziehen soll, also auch auf die, die als Strecker erstellt wurden. Hier also die Lösung, die Dynamikzeichen aus dem Beispiel des vorigen Abschnitts auszurichten:

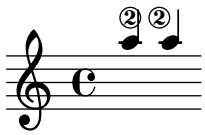
```
\dynamicUp
% Breite um eine Einheit vergrößern
\override DynamicText #'extra-spacing-width = #'(-0.5 . 0.5)
% Dynamik-Zeichen an einer Linie ausrichten, die 2 Einheiten über dem System ist
\override DynamicLineSpanner #'staff-padding = #2
a4\f b\mf c\mp b\p
```



self-alignment-X (Selbstausrichtung-X-Eigenschaft)

Das nächste Beispiel zeigt, wie man den Zusammenstoß einer Fingersatzbezeichnung mit einem Notenhals verhindern kann, indem die rechte Ecke an dem Referenzpunkt der abhängigen Note angeordnet wird:
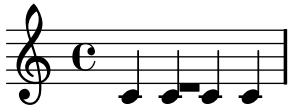
```
\voiceOne
< a \2 >
\once \override StringNumber #'self-alignment-X = #RIGHT
< a \2 >
```



staff-position (Position innerhalb des Systems)

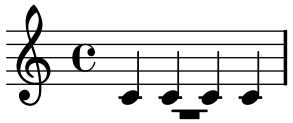
Vieltaktpausen in einer Stimmen können mit Noten in anderen Stimmen kollidieren. Da diese Pausen zentriert zwischen den Taktlinien gesetzt werden, würde es für LilyPond eine recht große Anstrengung bedeuten herauszufinden, welche Noten mit ihnen zusammenstoßen könnten, denn alle Kollisionsvermeidung für Noten und Pausen funktioniert nur für Noten bzw. Pausen, die zur selben Zeit auftreten. Hier ein typisches Beispiel für eine Kollision dieser Art:

```
<< {c c c c} \\ {R1} >>
```



Die beste Lösung ist es, die Ganztaktpause nach unten zu schieben, denn die Pause ist in der zweiten Stimme. Per Standardeinstellung für die zweite Stimme (`\voiceTwo`, also die zweite Stimme in der `<<{...} \\ {...}>>`-Konstruktion) wird die Position auf dem System (`staff-position`) auf -4 für `MultiMeasureRest`, in unserem Beispiel muss es also bspw. auf die Position `-8` gesetzt werden, d.h. vier halbe Notenlinienabstände weiter nach unten:

```
<<
  {c c c c}
\\
  \override MultiMeasureRest #'staff-position = #-8
  {R1}
>>
```



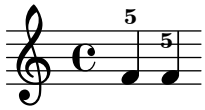
Das ist besser, als etwa `extra-offset` zu benutzen, denn in unserem Fall wird die Hilfslinie der Pause automatisch gesetzt.

extra-offset (Genaues Positionieren)

Die `extra-offset`-Eigenschaft bietet vollständige Kontrolle über die Positionierung von Objekten in horizontaler und vertikaler Richtung.

Im Beispiel unten ist das zweite Fingersatzzeichen (`Fingering`) etwas nach links und 1,8 Notenlinienabstände nach unten verschoben:

```
\stemUp
f-5
\once \override Fingering
    #'extra-offset = #'(-0.3 . -1.8)
f-5
```



Ausrichtungseigenschaft
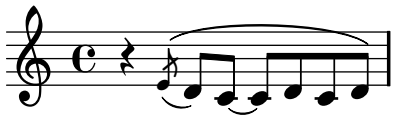
Die `positions`-Eigenschaft erlaubt die Kontrolle von Position und Steigung von Balken, Legato- und Phrasierungsbögen sowie Triolenklammern. Hier ein Beispiel, in der ein unschöner Phrasierungsbogen auftritt, weil er den Bogen des Vorschlags vermeidet:

```
r4  \acciaccatura e8\( d8 c ~c d c d\)
```
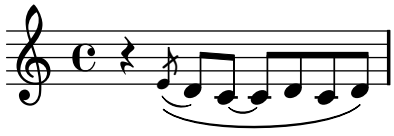


Man könnte einfach den Phrasierungsbogen oberhalb der Noten setzen, und das wäre auch die beste Lösung:

```
r4
\phrasingSlurUp
\acciaccatura e8\( d8 c ~c d c d\)
```



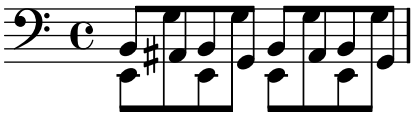
aber wenn es einen Grund geben sollte, warum das nicht geht, könnte man das linke Ende des Phrasierungsbogens etwas nach unten verschieben, indem man die `positions`-Eigenschaft einsetzt. Damit verschwindet auch die etwas unschöne Form:

```
r4
\once \override PhrasingSlur #'positions = #'(-4 . -3)
\acciaccatura
e8\( d8 c ~c d c d\)
```



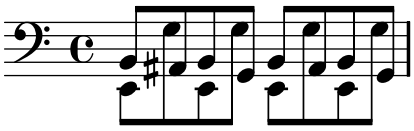
Hier noch ein weiteres Beispiel aus der Einleitung von Chopins Prelude Op. 28 Nr. 2, das untere System. Wie zu sehen ist, stößt der Balken mit den oberen Noten zusammen:

```
{
\clef "bass"
<< {b,8 ais, b, g,} \\ {e, g e, g} >>
<< {b,8 ais, b, g,} \\ {e, g e, g} >>
}
```



Das kann manuell gelöst werden, indem beide Enden des Balkens von ihrer Position 2 Notenlinienabstände über der Mittellinie hochgeschoben werden, etwa auf 3:

```
{
  \clef "bass"
  <<
    \override Beam #'positions = #'(3 . 3)
    {b,8 ais, b, g,}
  \\
    {e, g e, g}
  >>
  << {b,8 ais, b, g,} \\ {e, g e, g} >>
}
```
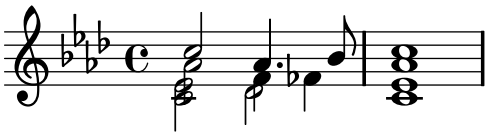


Hier ist zu beobachten, dass die Veränderung sich auch auf die weiteren Achtelbalken der ersten Stimme auwirkt, während sie keine Auswirkung auf die Balken der zweiten Stimme hat.

force-hshift (vertikale Verschiebunseigenschaft)

An diesem Punkt können wir den letzten Feinschliff an unserem Chopin-Beispiel vornehmen, das wir behandelt haben in Abschnitt 3.2.1 [Ich höre Stimmen], Seite 49. Wir hatten es in folgende Form gebracht:
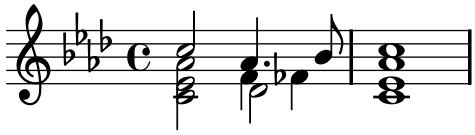
```
\new Staff \relative c'' {
  \key aes \major
  <<
    { c2 aes4. bes8 } \\
    { aes2 f4 fes   } \\
    { \voiceFour
      <ees c>2
      des2
    }
  >> |
  <c ees aes c>1 |
}
```



Die unteren zwei Noten des ersten Akkords (also diein der dritten Stimme) sollten nicht aus der Notenkolumne der oberen zwei Noten weggeschoben werden. Um das zu korrigieren, setzen wir `force-hshift`, das eine Eigenschaft von `NoteColumn` ist, für diese Noten auf Null. Die untere Note des zweiten Akkordes wird am besten direkt rechts von den oberen Noten gesetzt. Das erreichen wir, indem wir `force-hshift` für diese Note auf 0.5 setzen, also eine halbe Notenkopfbreite nach rechts von der Kolumne der oberen Noten aus.

Hier das Endergebnis:

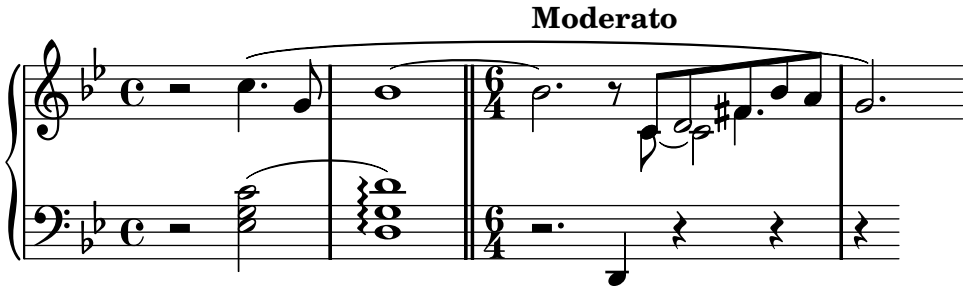
```
\new Staff \relative c'' {
  \key aes \major
  <<
    { c2 aes4. bes8 } \\
    { aes2 f4 fes   } \\
    { \voiceFour
      \once \override NoteColumn #'force-hshift = #0 <ees c>2
      \once \override NoteColumn #'force-hshift = #0.5 des2
    }
  >> |
  <c ees aes c>1 |
}
```



4.5.3 Beispiele aus dem Leben

Das Kapitel zu Optimierungen soll mit einem komplizierten Beispiel beendet werden, in dem verschiedene Optimierungen vorgenommen werden müssen, bis das Ergebnis gut ausssieht. Das Beispiel wurde ganz bewusst gewählt um die Benutzung der Notationsreferenz zu zeigen, wenn ungewöhnliche Notationsprobleme gelöst werden müssen. Es ist nicht repräsentativ für normale Notationsprojekte, lassen Sie sich also nicht durch dieses Beispiel entmutigen! Zum Glück sind Probleme wie die hier gezeigten nicht sehr häufig.

Das Beispiel stammt aus Chopins Premiére Ballade, Op. 23, Takte 6–9, der Übergang vom Lento der Einleitung zum Moderato. Hier zunächst der Satz, wie er aussehen soll, allerdings ohne Dynamik, Fingersatz und Pedalbezeichnung, um das Beispiel nicht zu kompliziert zu machen.



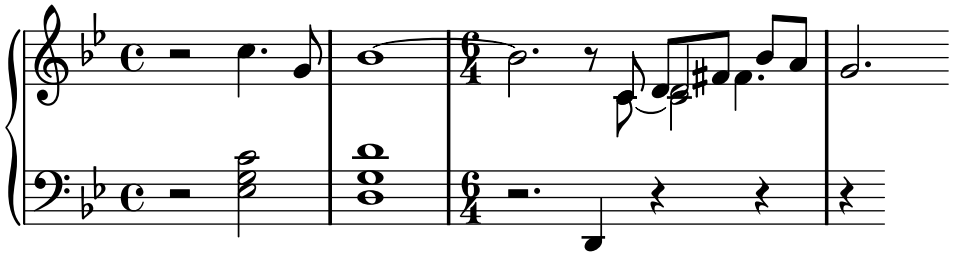
Die erste Überlegung ist, dass das System für die rechte Hand im dritten Takt vier Stimmen braucht. Das sind die fünf Achtelnoten mit Balken, das übergebundene C, die Halbe D, die mit der Achtel D verschmolzen ist, und die punktierte Viertel Fis, die auch mit einer Achtelnote verschmolzen ist. Alles andere ist eine einzige Stimme, es ist also am einfachsten, die Stimmen nur zeitweise zu erstellen, wenn sie auftreten. Wenn Sie vergessen haben, wie man das anstellt, schauen Sie sich nochmal den Abschnitt Abschnitt 3.2.1 [Ich höre Stimmen], Seite 49 an. Wir wollen anfange, indem wir die Noten in zwei Variablen notieren und dann die Systemstruktur in einer `\score`-Umgebung erstellen. Das ist, was LilyPond erstellt:

```
rhNoten = \relative c'' {
  r2 c4. g8 |
  bes1~ |
  \time 6/4
  bes2. r8
```

```
  % Beginn des polyphonen Abschnitts mit vier Stimmen
  <<
    {c,8 d fis bes a | }
  \\
    {c,8~ c2 | }
  \\
    {s8 d2 | }
  \\
    {s4 fis4. | }
  >>
  g2.
}

lhNoten = \relative c' {
  r2 <c g ees>2 |
  <d g, d>1 |
  r2. d,,4 r4 r |
  r4
}

\score {
  \new PianoStaff <<
    \new Staff = "RH"  <<
      \key g \minor
      \rhNoten
    >>
    \new Staff = "LH" <<
      \key g \minor
      \clef "bass"
      \lhNoten
    >>
  >>
}
```
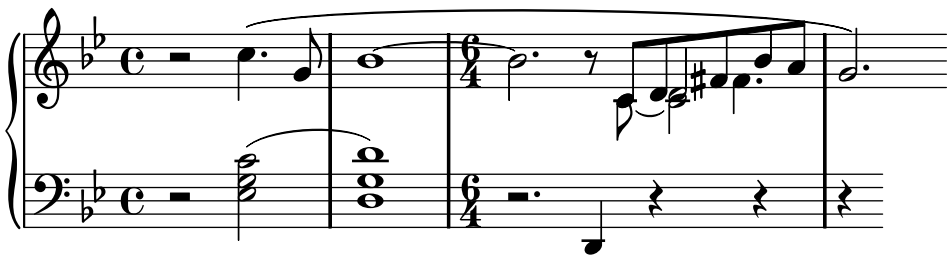


Alle Noten sind richtig, aber die Positionierung sehr verbesserungsbedürftig. Der Bindebogen stößt mit der veränderten Taktart zusammen, die Balkung im dritten Takt ist falsch, die Noten werden nicht verschmolzen und einige Notationselemente fehlen ganz. Behandeln wir zunächst die einfacheren Dinge. Der Balken kann durch eine manuelle Begrenzung einfach korrigiert werden, und auch der Legatobogen der linken Hand und der Phrasierungsbogen der rechten Hand sind schnell gesetzt, denn sie wurden schon in der Übung erklärt. Damit haben wir folgendes Notenbild:

```
rhNoten = \relative c'' {
  r2 c4.\( g8 |
  bes1~ |
```

```
  \time 6/4
  bes2. r8
  % Beginn des polyphonen Abschnitts mit vier Stimmen
  <<
    {c,8[ d fis bes a] | }
  \\
    {c,8~ c2 | }
  \\
    {s8 d2 | }
  \\
    {s4 fis4. | }
  >>
  g2.\)
}

lhNoten = \relative c' {
  r2 <c g ees>2( |
  <d g, d>1) |
  r2. d,,4 r4 r |
  r4
}

\score {
  \new PianoStaff <<
    \new Staff = "RH"  <<
      \key g \minor
      \rhNoten
    >>
    \new Staff = "LH" <<
      \key g \minor
      \clef "bass"
      \lhNoten
    >>
  >>
}
```



Der erste Takt stimmt jetzt schon. Der zweite Takt enthält ein Arpeggio und wird mit einer doppelten Taktlinie beschlossen. Wie können wir diese notieren, denn sie sind im Handbuch zum Lernen nicht vorgekommen? Hier brauchen wir jetzt die Notationsreferenz. Ein Blick in den Index zeigt uns die Einträge für „Arpeggio“ und „Taktlinien“: ein Arpeggio also erstellt man mit dem Befehl `\arpeggio` hinter einem Akkord und eine doppelte Taktlinie wird mit dem Befehl `\bar "||"` erstellt. Das ist einfach. Als nächstes muss der Zusammenstoß des Bindebogens mit der Taktartbezeichnung gelöst werden. Das geht am besten, indem wir den Bogen nach oben verschieben. Wie man Objekte verschiebt wurde schon behandelt in Abschnitt 4.5.1

[Verschieben von Objekten], Seite 116, wo stand, dass Objekte die relativ zum System positioniert werden, verschoben werden können, indem ihre `staff-position`-Eigenschaft geändert wird, die in halben Notenlienienabständen relativ zur Mittellinie angegeben wird. Dieser `\override`-Befehl also, direkt vor die erste übergebundene Note gestellt, verschiebt den Bindebogen (`tie`) 3,5 halbe Notenlinienabstände über die Mittellinie:

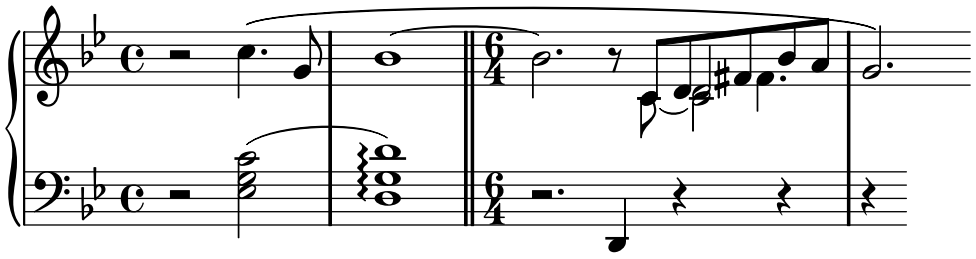
```
\once \override Tie #'staff-position = #3.5
```

Damit ist auch der zweite Takt vollständig:

```
rhNoten = \relative c'' {
  r2 c4.\( g8 |
  \once \override Tie #'staff-position = #3.5
  bes1~ |
  \bar "||"
  \time 6/4
  bes2. r8
  % Beginn des polyphonen Abschnitts mit vier Stimmen
  <<
    {c,8[ d fis bes a] | }
  \\
    {c,8~ c2 | }
  \\
    {s8 d2 | }
  \\
    {s4 fis4. | }
  >>
  g2.\)
}

lhNoten = \relative c' {
  r2 <c g ees>2( |
  <d g, d>1)\arpeggio |
  r2. d,,4 r4 r |
  r4
}

\score {
  \new PianoStaff <<
    \new Staff = "RH"  <<
      \key g \minor
      \rhNoten
    >>
    \new Staff = "LH" <<
      \key g \minor
      \clef "bass"
      \lhNoten
    >>
  >>
}
```



In Takt drei beginnt der Moderato-Abschnitt. In der Übung wurde behandelt, wie man fetten Text mit dem `\markup`-Befehl eingibt, es ist also einfach, das „Moderato“ hinzuzufügen. Wie aber werden Noten verschmolzen? Hier nehmen wir wieder die Notationsreferenz zu Hilfe. Die Suche nach „Verschmelzen“ (engl. merge) im Index führt uns zu den Befehlen um Noten mit unterschiedlichen Köpfen und unterschiedlichen Punkten zu verschmelzen in Abschnitt “Auflösung von Zusammenstößen” in *Notationsreferenz*. In unserem Beispiel müssen sowohl unterschiedliche Köpfe also auch unterschiedliche Punktierung verschmolzen werden, wir brauchen also die Befehle

```
\mergeDifferentlyHeadedOn
\mergeDifferentlyDottedOn
```

aus der Notationsreferenz, die wir an den Beginn unseres Abschnittes stellen und

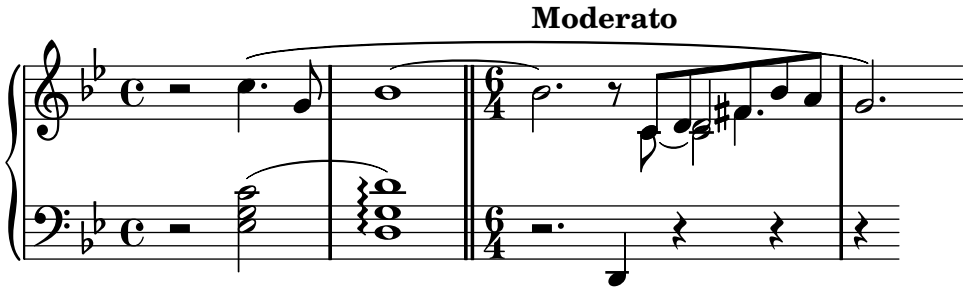
```
\mergeDifferentlyHeadedOff
\mergeDifferentlyDottedOff
```

um das Verhalten wieder auszuschalten. Das sieht so aus:

```
rhNoten = \relative c'' {
  r2 c4.\( g8 |
  \once \override Tie #'staff-position = #3.5
  bes1~ |
  \bar "||"
  \time 6/4
  bes2.^\markup {\bold "Moderato"} r8
  \mergeDifferentlyHeadedOn
  \mergeDifferentlyDottedOn
  % Beginn des polyphonen Abschnitts mit vier Stimmen
  <<
    {c,8[ d fis bes a] | }
  \\
    {c,8~ c2 | }
  \\
    {s8 d2 | }
  \\
    {s4 fis4. | }
  >>
  \mergeDifferentlyHeadedOff
  \mergeDifferentlyDottedOff
  g2.\)
}

lhNoten = \relative c' {
  r2 <c g ees>2( |
  <d g, d>1)\arpeggio |
  r2. d,,4 r4 r |
  r4
}
```

```
\score {
  \new PianoStaff <<
    \new Staff = "RH"  <<
      \key g \minor
      \rhNoten
    >>
    \new Staff = "LH" <<
      \key g \minor
      \clef "bass"
      \lhNoten
    >>
  >>
}
```



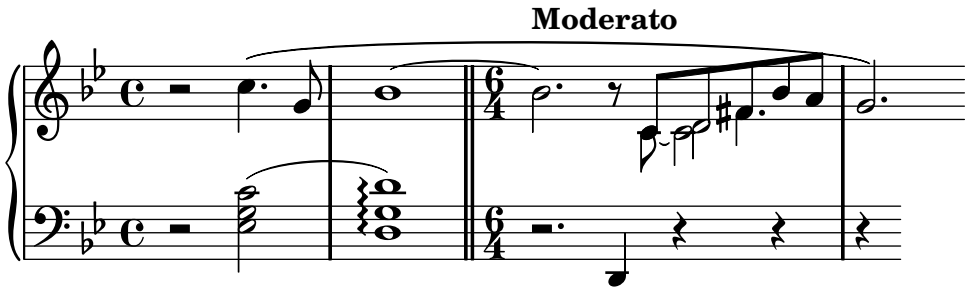
Mit diesen Veränderungen wurden die beiden Fis-Noten verschmolzen, aber nicht die zwei Ds. Warum nicht? Die Antwort befindet sich im gleicher Abschnitt der Notationsreferenz: Noten, die verschmolzen werden, müssen Hälse in entgegengesetzte Richtungen aufweisen und zwei Noten können nicht verschmolzen werden, wenn eine dritte Noten in der gleichen Kolumne stört. In unserem Fall weisen beide Hälse nach oben und es befindet sich zur gleichen Zeit auch noch eine dritte Note, das C. Wie die Richtung von Hälsen geändert wird, wissen wir schon: mit `\stemDown`, und in der Notationsreferenz findet sich auch Information, wie das C verschoben werden kann: mit dem `\shift`-Befehl. Aber welcher von ihnen? Das C befindet sich in der zweiten Stimme, die „shift off“ hat, die zwei Ds sind in den Stimmen eins und drei, die „shift off“ bzw. „shift on“ haben. Das C muss also noch eine Stufe weiter verschoben werden mit `\shiftOnn`, damit es die Verschmelzung der Ds nicht stört. Das sieht jetzt so aus:

```
rhNoten = \relative c'' {
  r2 c4.\( g8 |
  \once \override Tie #'staff-position = #3.5
  bes1~ |
  \bar "||"
  \time 6/4
  bes2.^\markup {\bold "Moderato"} r8
  \mergeDifferentlyHeadedOn
  \mergeDifferentlyDottedOn
  % Beginn des polyphonen Abschnitts mit vier Stimmen
  <<
    {c,8[ d fis bes a] | }
  \\
    % Verschiebe das c2 aus der Hauptnotenkolumne, damit Verschmelzung funktioniert
    {c,8~ \shiftOnn c2 | }
  \\
    % Hals vom d2 muss nach unten, damit Verschmelzung gelingt
    {s8 \stemDown d2 | }
```

```
  \\
    {s4 fis4. | }
  >>
  \mergeDifferentlyHeadedOff
  \mergeDifferentlyDottedOff
  g2.\)
}

lhNoten = \relative c' {
  r2 <c g ees>2( |
  <d g, d>1)\arpeggio |
  r2. d,,4 r4 r |
  r4
}

\score {
  \new PianoStaff <<
    \new Staff = "RH"  <<
      \key g \minor
      \rhNoten
    >>
    \new Staff = "LH" <<
      \key g \minor
      \clef "bass"
      \lhNoten
    >>
  >>
}
```



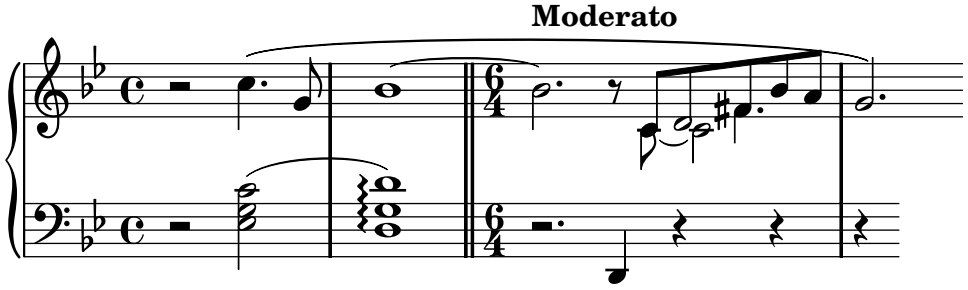
Fast schon geschafft. Nur noch ein Problem ist übrig: Der Hals nach unten des verschmolzenen sollte nicht da sein, und das C sähe besser auf der rechten Seite des Ds aus. Beides können wir mit den gelernten Optimierungsmethoden erreichen. Den Hals machen wir durchsichtig und das C verschieben wir mit der `force-hshift`-Eigenschaft. Hier ist das Endergebnis:

```
rhNoten = \relative c'' {
  r2
  c4.\( g8 |
  \once \override Tie #'staff-position = #3.5
  bes1~ |
  \bar "||"
  \time 6/4
  bes2.^\markup {\bold "Moderato"} r8
  \mergeDifferentlyHeadedOn
  \mergeDifferentlyDottedOn
```

```
  <<
    {c,8[ d fis bes a] | }
  \\
    % c2 neu positionieren rechts von der verschmolzenen Note
    {c,8~ \once \override NoteColumn #'force-hshift = #1.0
    % Verschiebe das c2 aus der Hauptnotenkolumne, damit Verschmelzung funktioniert
    \shiftOnn c2}
  \\
    % Hals vom d2 muss nach unten, damit Verschmelzung gelingt
    {s8 \stemDown \once \override Stem #'transparent = ##t d2}
  \\
    {s4 fis4.}
  >>
  \mergeDifferentlyHeadedOff
  \mergeDifferentlyDottedOff
  g2.\)
}

lhNoten = \relative c' {
  r2 <c g ees>2( |
  <d g, d>1)\arpeggio |
  r2. d,,4 r4 r |
  r4
}

\score {
  \new PianoStaff <<
    \new Staff = "RH"  <<
      \key g \minor
      \rhNoten
    >>
    \new Staff = "LH" <<
      \key g \minor
      \clef "bass"
      \lhNoten
    >>
  >>
}
```
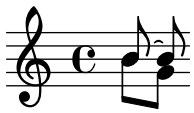


4.6 Weitere Optimierungen

4.6.1 Andere Benutzung von Optimierungen
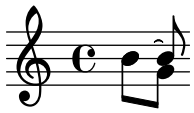
Noten zwischen unterschiedlichen Stimmen überbinden

Das nächste Beispiel zeigt, wie man Noten von verschiedenen Stimmen miteinander verknüpfen kann, indem man Bindebögen für Überbindungen benutzt. Normalerweise können nur zwei Noten der gleichen Stimme übergebunden werden. Wenn man zwei Stimmen benutzt, wobei die überbundenen Noten sich in der selben befinden,
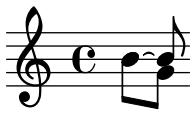


und dann den ersten Hals nach oben unsichtbar macht, sieht es so aus, als ob die Überbindung zwischen den Stimmen stattfindet:

```
<<
  {
    \once \override Stem #'transparent = ##t
    b8~ b8\noBeam
  }
\\
  { b[ g8] }
>>
```



Um sicherzugehen, dass der unsichtbare Hals den Bindebogen nicht zu sehr verkleinert, kann er verlängert werden, indem seine Länge (`length`) auf den Wert `8` gesetzt wird:

```
<<
  {
    \once \override Stem #'transparent = ##t
    \once \override Stem #'length = #8
    b8~ b8\noBeam
  }
\\
  { b[ g8] }
>>
```



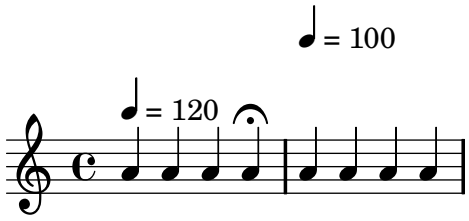
Eine Fermate in MIDI simulieren

Für Objekte außerhalb des Notensystems ist es normalerweise besser, die `stencil`-Eigenschaft anstelle der `transparent`-Eigenschaft zu verändern, wenn man sie vom fertigen Notensatz entfernen will. Indem die `stencil`-Eigenschaft auf falsch (`#f`) gesetzt wird, wird das entsprechende Objekt vollständig entfernt. Das bedeutet, dass es die Positionierung der anderen Objekte nicht beeinflusst.
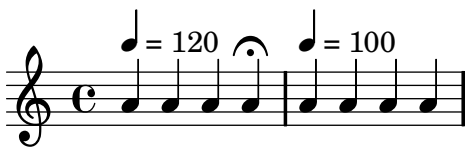
Auf diese Art kann etwa das Tempo geändert werden, damit in der MIDI-Ausgabe eine Fermate zu hören ist, ohne dass im Notensatz etwas von diesen Tempoänderungen zu sehen ist. Die Metronombezeichnung soll auch nicht die Position von Text an der gleichen Stelle oder

die Abstände zwischen zwei Systemen beeinflussen. Darum ist es am besten, `stencil` auf `#f` zu setzen. Im Beispiel wird der Unterschied zwischen einem unsichtbaren Objekt und einem entfernten Objekt gezeigt:

```
\score {
  \relative c'' {
    % Sichtbare Tempo-Bezeichnung
    \tempo 4=120
    a4 a a
    \once \override Score.MetronomeMark #'transparent = ##t
    % Unsichtbare Tempo-Bezeichnung um Fermate im MIDI zu verlängern
    \tempo 4=80
    a\fermata
    % Neues Tempo im nächsten Abschnitt
    \tempo 4=100
    a a a a
  }
  \layout { }
  \midi { }
}
```




```
\score {
  \relative c'' {
    % Sichtbare Tempo-Bezeichnung
    \tempo 4=120
    a4 a a
    \once \override Score.MetronomeMark #'stencil = ##f
    % Unsichtbare Tempo-Bezeichnung um Fermate im MIDI zu verlängern
    \tempo 4=80
    a\fermata
    % Neues Tempo im nächsten Abschnitt
    \tempo 4=100
    a a a a
  }
  \layout { }
  \midi { }
}
```



Mit beiden Methoden wird die Tempobezeichnung entfernt, mit der die Fermate verlängert wird, und beide beeinflussen die MIDI-Ausgabe wie gewünscht. Die unsichtbare Metronombezeichnung schiebt aber die folgende Bezeichnung in die Höhe, während das im zweiten Beispiel, in dem der `stencil` entfernt wurde, nicht passiert.

4.6.2 Variablen für Optimierungen einsetzen

`\override`-Befehle sind oft lang und mühsam zu tippen, und sie müssen immer absolut richtig sein. Wenn derselbe Befehl mehrere Male benutzt werden muss, lohnt es sich oft schon, eine Variable zu definieren, in der er sich befindet.

Als Beispiel sollen einige Worte im Gesangstext fett und kursiv hervorgehoben werden. Die Befehle `\italic` und `\bold` funktionieren im Gesangstext-Kontext nur, wenn sie gleichzeitig mit den Wörtern, auf die sie angewendet werden sollen, zusätzlich in eine `\markup`-Umgebung eingeschlossen werden. Durch diese Einbettung können einzelne Wörter nicht einfach zu einer Variable umgeformt werden. Als Alternative versuchen wir, einen Befehl mit `\override` und `\revert` zu konstruieren.

```
\override Lyrics . LyricText #'font-shape = #'italic
\override Lyrics . LyricText #'font-series = #'bold

\revert Lyrics . LyricText #'font-shape
\revert Lyrics . LyricText #'font-series
```
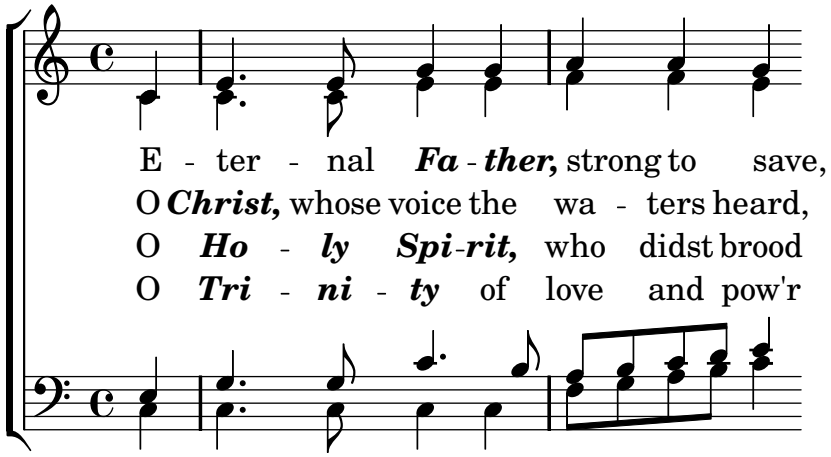
Das wäre natürlich noch viel mühsamer, wenn viele Wörter eine Hervorhebung benötigen. Anstelle dieser Befehlsketten *können* wir jedoch zwei Variablen definieren. Mit ihnen und dem entsprechenden Wort in geschweiften Klammern erreichen wir den gewünschten Effekt. Ein weiterer Vorteil ist, dass in diesem Fall die Leerzeichn um die Punkte herum nicht benötigt werden, weil sie nicht innerhalb des `lyricmode`-Kontextes interpretiert werden. Hier ein Beispiel; die Bezeichnungen können natürlich auch kürzer sein, um noch weniger schreiben zu müssen:

```
emphasize = {
  \override Lyrics.LyricText #'font-shape = #'italic
  \override Lyrics.LyricText #'font-series = #'bold
}
normal = {
  \revert Lyrics.LyricText #'font-shape
  \revert Lyrics.LyricText #'font-series
}

global = { \time 4/4 \partial 4 \key c \major}
SopranNoten   = \relative c' { c4 | e4. e8 g4  g  | a a g }
AltNoten  = \relative c' { c4 | c4. c8 e4  e  | f f e }
TenorNoten = \relative c  { e4 | g4. g8 c4. b8 | a8 b c d e4 }
BassNoten  = \relative c  { c4 | c4. c8 c4  c  | f8 g a b c4 }
StropheEins   = \lyrics { E -- | ter -- nal \emphasize Fa -- ther, \normal | strong to
StropheZwei   = \lyricmode { O | \emphasize Christ, \normal whose voice the | wa -- te
StropheDrei = \lyricmode { O | \emphasize Ho -- ly Spi -- rit, \normal | who didst bro
StropheVier  = \lyricmode { O | \emphasize Tri -- ni -- ty \normal of | love and pow'r

\score {
  \new ChoirStaff <<
    \new Staff <<
      \clef "treble"
      \new Voice = "Sopran"  { \voiceOne \global \SopranNoten }
      \new Voice = "Alt" { \voiceTwo \AltNoten }
      \new Lyrics \lyricsto "Sopran" { \StropheEins   }
      \new Lyrics \lyricsto "Sopran" { \StropheZwei   }
      \new Lyrics \lyricsto "Sopran" { \StropheDrei }
      \new Lyrics \lyricsto "Sopran" { \StropheVier  }
    >>
```

```
    \new Staff <<
      \clef "bass"
      \new Voice = "Tenor" { \voiceOne \TenorNoten }
      \new Voice = "Bass"  { \voiceTwo \BassNoten  }
    >>
  >>
}
```



4.6.3 Mehr Information

Die Programmreferenz enthält sehr viel Information über LilyPond, aber noch mehr Information findet sich in den internen LilyPond-Dateien. Um sie erforschen zu können, müssen Sie erst das richtige Verzeichnis auf Ihrem System finden. Die Position hängt a) davon ab, ob Ihre LilyPond-Installation mit der vorkompilierten Version von der LilyPond-Internetseite vorgenommen wurde oder Sie die Version durch Ihren Paketmanager installiert haben (also z. B. in einer Linux-Distribution oder unter fink oder cygwin installiert), und b) auf welchem Betriebssystem Sie das Programm benutzen:

Von lilypond.org heruntergeladen

- Linux

  Wechseln Sie in das Verzeichnis '*INSTALL_VERZ*`/lilypond/usr/share/lilypond/current/`'
- MacOS X

  Wechseln Sie in das Verzeichnis '*INSTALL_VERZ*`/LilyPond.app/Contents/Resources/share/lilypond/c`
  indem Sie entweder mit dem Befehl `cd` vom Terminal aus in das Verzeichnis wechseln, oder mit Control-Klick auf das LilyPond-Programmsymbol gehen und „Show Package Contents“ auswählen.
- Windows

  Wechseln Sie mit dem Windows Explorer ins Verzeichnis '*INSTALL_VERZ*`/LilyPond/usr/share/lilypond/current/`'

Mit einem Paket-Manager installiert oder selber aus den Quellen kompiliert

Wechseln Sie in das Verzeichnis '*PREFIX*`/share/lilypond/`*X.Y.Z*`/`', wobei *PREFIX* bei Ihrem Paket-Manager oder dem `configure`-Skript gesetzt wird, und *X.Y.Z* die LilyPond-Versionsnummer.

In diesem Ordner sind die zwei interessanten Unterordner:

- '`ly/`' - beinhaltet Dateien im LilyPond-Format
- '`scm/`' - beinhaltet Dateien im Scheme-Format

Schauen wir uns zuerst einige Dateien in ‘ly/’ an. Öffnen Sie ‘ly/property-init.ly’ in einem Texteditor. Der, den Sie normalerweise für .ly-Dateien benutzen, genügt. Diese Datei enthält die Definitionen aller vordefinierten Befehle für LilyPond, wie etwa \stemUp und \slurDotted. Sie können sehen, dass es sich um nichts mehr handelt als Definitionen von Variablen, die eine oder mehrere \override-Befehle enthalten. Der Befehl /tieDotted etwa wird folgendermaßen definiert:

```
tieDotted = {
  \override Tie #'dash-period = #0.75
  \override Tie #'dash-fraction = #0.1
}
```

Wenn Sie diese Voreinstellungen der vordefinierten Befehl nicht mögen, können Sie sie ganz einfach umdefinieren, genauso wie jede andere Variable auch, indem Sie sie an den Anfang Ihrer Quelldatei schreiben.

Hier sind die wichtisgsten Dateien, die sich im Ordner ‘ly/’ befinden:

Dateiname	**Inhalt**
‘ly/engraver-init.ly’	Definitionen von Engraver-Kontexten
‘ly/paper-defaults-init.ly’	Spezifikationen von Voreinstellungen für Papiermaße
‘ly/performer-init.ly’	Definitionen von Performer-Kontexten
‘ly/property-init.ly’	Definitionen aller vordefinierten Befehle
‘ly/spanner-init.ly’	Definitionen aller vordefinierten Strecker-Befehle

Andere Einstellungen (wie die Definitionen von Beschriftungsbefehlen) sind in .scm-(Scheme)-Dateien gespeichert. Die Scheme-Programmiersprache wird benutzt, um eine programmierbare Schnittstelle zu den internen Operationen von LilyPond zu haben. Eine weitere Erklärung dieser Dateien ist im Moment außerhalb des Rahmens dieses Handbuchs, denn sie erfordern einige Kenntnis der Scheme-Sprache. Die Warnung ist hier angebracht, dass des ein gutes technisches Verständnis oder sehr viel Zeit braucht, um Scheme und diese Dateien zu verstehen (siehe auch Anhang B [Scheme-Übung], Seite 181).

Wenn Sie sich mit Scheme auskennen, sind hier mögliche interessante Dateien:

Dateiname	**Inhalt**
‘scm/auto-beam.scm’	Sub-Balken-Voreinstellungen
‘scm/define-grobs.scm’	Voreinstellungen für Grob-Eigenschaften
‘scm/define-markup-commands.scm’	Definition aller Markup-Beschriftungsbefehle
‘scm/midi.scm’	Voreinstellung für die MIDI-Ausgabe
‘scm/output-lib.scm’	Einstellungen mit Einfluss auf die Darstellung von Bunddiagrammen, Farben, Versetzungszeichen, Taktlinien usw.
‘scm/parser-clef.scm’	Definitionen der unterstützten Schlüssel
‘scm/script.scm’	Voreinstellungen for Artikulationszeichen

4.6.4 Vermeiden von Optimierungen durch langsamere Übersetzung

LilyPond kann einige zusätzliche Tests durchführen, während die Noten gesetzt werden. Dadurch braucht das Programm länger, um den Notensatz zu produzieren, aber üblicherweise werden weniger nachträgliche Anpassungen nötig sein. Wenn eine Textsilbe oder eine Beschriftung aus dem Rand der Partitur ragt, wird durch diese Tests die Zeile gerade so weit komprimiert, dass sie sich innerhalb der Ränder befindet.

```
\new Score \with {
 %  Um sicher zu gehen, dass Texte und Liedtext
 %  innerhalb der Papierränder bleiben
```

```
 \override PaperColumn #'keep-inside-line = ##t
 \override NonMusicalPaperColumn #'keep-inside-line = ##t
} {
   ...
}
```

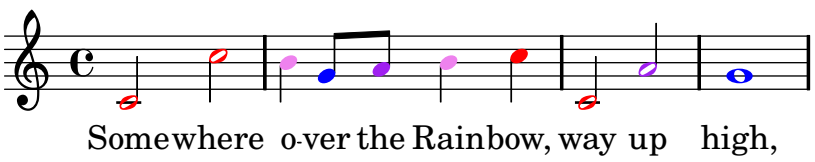
4.6.5 Fortgeschrittene Optimierungen mit Scheme

Auch wenn viele Sachen mit `\override` und `\tweak` möglich sind, gibt es eine sehr viel mächtigere Möglichkeit, die Arbeitsweise von LilyPond mit Hilfe der programmierbaren Schnittstelle zu beeinflussen. Code, der in der Scheme-Programmiersprache geschrieben ist, kann direkt in die interne Satzmaschine von LilyPond eingefügt werden. Natürlich brauchen Sie dazu wenigstens ein grundlegendes Verständnis von Scheme. Eine Einleitung finden Sie in der Anhang B [Scheme-Übung], Seite 181.

Zur Illustration der vielen Möglichkeiten soll gezeigt werden, dass eine Eigenschaft nicht nur auf eine Konstante, sondern auch auf eine Scheme-Prozedur gesetzt werden kann, die dann jedes Mal aufgerufen wird, wenn die Eigenschaft von LilyPond benutzt wird. Die Eigenschaft kann damit dynamisch auf einen Wert gesetzt werden, der durch die Prozedur jedes Mal neu bestimmt wird. In diesem Beispiel wird die Farbe der Notenköpfe entsprechend zu ihrer Position innerhalb der Tonleiter gesetzt.

```
#(define (color-notehead grob)
  "Color the notehead according to its position on the staff."
  (let ((mod-position (modulo (ly:grob-property grob 'staff-position) 7)))
    (case mod-position
      ;;   Return rainbow colors
      ((1) (x11-color 'red    ))  ; for C
      ((2) (x11-color 'orange ))  ; for D
      ((3) (x11-color 'yellow ))  ; for E
      ((4) (x11-color 'green  ))  ; for F
      ((5) (x11-color 'blue   ))  ; for G
      ((6) (x11-color 'purple ))  ; for A
      ((0) (x11-color 'violet ))  ; for B
    )
  )
)

\relative c' {
  % Anordnungen um Farbe von der color-notehead-Prozedur zu erhalten
  \override NoteHead #'color = #color-notehead
  c2 c' |
  b4 g8 a b4 c |
  c,2 a' |
  g1 |
}
\addlyrics {
  Some -- where o -- ver the Rain -- bow, way up high,
}
```



Weiter Beispiele, die die Benutzung dieser programmierbaren Schnittstelle zeigen, finden sich in Abschnitt B.1 [Optimierungen mit Scheme], Seite 183.

5 An LilyPond-Projekten arbeiten

Dieses Kapitel erklärt, wie bestimmte häufige Probleme zu lösen oder ganz zu vermeiden sind. Wenn Sie schon Programmiererfahrung mitbringen, erscheinen diese Hinweise vielleicht überflüssig, aber es wird dennoch empfohlen, dieses Kapitel zu lesen.

5.1 Vorschläge, wie LilyPond-Eingabe-Dateien geschrieben werden sollen

Jetzt sind Sie so weit, größere Stücke mit LilyPond zu schreiben – nicht nur die kleinen Beispiele aus der Übung, sondern ganze Stücke. Aber wie geht man das am besten an?

Solange LilyPond Ihre Dateien versteht und die Noten so setzt, wie Sie das wollen, spielt es eigentlich keine Rolle, wie Ihre Dateien aussehen. Es gibt aber trotzdem ein paar Dinge, die man beim Schreiben von LilyPond-Code berücksichtigen sollte.

- Was ist, wenn Sie einen Fehler machen? Die Struktur einer LilyPond-Datei kann es erleichtern (oder erschweren), bestimmte Fehler zu finden.
- Was ist, wenn Sie Ihre Dateien mit jemandem austauschen wollen? Oder Ihre Dateien nach einige Jahren noch einmal überarbeiten wollen? Manche LilyPond-Dateien versteht man auf den ersten Blick, über anderen muss man eine Stunde grübeln, um die Struktur zu ahnen.
- Was ist, wenn sie Ihre Dateien auf eine neuere LilyPond-Version aktualisieren wollen? Die Syntax der Eingabesprache verändert sich allmählich mit Verbesserungen im Programm. Die meisten Veränderungen können automatisch durch `convert-ly` gelöst werden, aber bestimmte Änderungen brauchen Handarbeit. LilyPond-Dateien können strukturiert werden, damit sie einfacher aktualisierbar sind.

5.1.1 Allgemeine Vorschläge

Hier einige Vorschläge, wie Sie Probleme vermeiden oder lösen können:

- **Schreiben Sie immer mit `\version` die Versionsnummer in jede Datei**. Beachten Sie, dass in allen Vorlagen die Versionsnummer `\version "2.12.0"` eingetragen ist. Es empfiehlt sich, in alle Dateien, unabhängig von ihrer Größe, den `\version`-Befehl einzufügen. Persönliche Erfahrung hat gezeigt, dass es ziemlich frustrierend sein kann zu erinnern, welche Programmversion man etwa vor einem Jahr verwendet hat. Auch `convert-ly` benötigt die Versionsnummer.
- **Benutzen Sie Überprüfungen**: Abschnitt “Oktavenüberprüfung” in *Notationsreferenz*, und Abschnitt “Takt- und Taktzahlüberprüfung” in *Notationsreferenz*. Wenn Sie hier und da diese Überprüfungen einfügen, finden Sie einen möglichen Fehler weit schneller. Wie oft aber ist „hier und da“? Das hängt von der Komplexität der Musik ab. ei einfachen Stücken reicht es vielleicht ein- oder zweimal, in sehr komplexer Musik sollte man sie vielleicht in jeden Takt einfügen.
- **Ein Takt pro Textzeile**. Wenn irgendetwas kompliziertes vorkommt, entweder in der Musik selber oder in der Anpassung der Ausgabe, empfiehlt es sich oft, nur einen Takt pro Zeile zu schreiben. Bildschirmplatz zu sparen, indem Sie acht Takte in eine Zeile zwängen, hilft nicht weiter, wenn Sie ihre Datei „debuggen“ müssen.
- **Kommentieren Sie ihre Dateien**. Benutzen Sie entweder Taktnummern (in regelmäßigen Abständen) oder Verweise auf musikalische Themen („Zweites Thema in den Geigen“, „vierte Variation“ usw.). Sie brauchen diese Kommentare vielleicht noch nicht, wenn Sie das Stück notieren, aber spätestens wenn Sie nach ein paar Jahren etwas verändern wollen oder Sie den Quelltext an einen Freund weitergeben wollen, ist es weitaus komplizierter, die Dateistruktur ohne Kommentare zu verstehen, als wenn Sie sie rechtzeitig eingefügt hätten.
- **Schreiben Sie Klammern mit Einrückung**. Viele Probleme entstehen durch ungerade Anzahl von `{` and `}`-Klammern.

- **Schreiben Sie Tondauerangaben** am Anfang von Abschnitten und Bezeichnern. Wenn Sie beispielsweise `c4 d e` am Anfang eines Abschnittes schreiben, ersparen Sie sich viele Probleme, wenn Sie ihre Musik eines Tages umarrangieren wollen.
- **Trennen Sie Einstellungen** von den eigentlichen Noten. Siehe auch Abschnitt 5.1.4 [Tipparbeit sparen durch Bezeichner und Funktionen], Seite 141 und Abschnitt 5.1.5 [Stil-Dateien], Seite 142.

5.1.2 Das Kopieren von existierender Musik

Wenn Sie Musik aus einer fertigen Partitur kopieren (z. B. die LilyPond-Eingabe einer gedruckten Partitur):

- Schreiben Sie ein System ihrer Quelle nach dem anderen (aber trotzdem nur einen Takt pro Textzeile) und überprüfen Sie jedes System, nachdem Sie es fertig kopiert haben. Mit dem `showLastLength`- oder `showFirstLenght`-Befehl können Sie den Übersetzungsprozess beschleunigen. Siehe auch Abschnitt “Korrigierte Musik überspringen” in *Notationsreferenz*.
- Definieren Sie `mBreak = { \break }` und schreiben Sie `\mBreak` in der Quelldatei immer dann, wenn im Manuskript ein Zeilenumbruch vorkommt. Das macht es einfacher, die gesetzte Zeile mit den ursprünglichen Noten zu vergleichen. Wenn Sie die Partitur fertig gestellt haben, könne Sie `mBreak = { }`, also leer definieren, um diese manuellen Zeilenumbrüche zu entfernen. Damit kann dann LilyPond selber entscheiden, wohin es passende Zeilenumbrüche platziert.

5.1.3 Große Projekte

Besonders wenn Sie an größeren Projekten arbeiten, ist es unumgänglich, dass Sie ihre LilyPond-Dateien klar strukturieren.

- **Verwenden Sie Variablen für jede Stimme**, innerhalb der Definition sollte so wenig Struktur wie möglich sein. Die Struktur des `\score`-Abschnittes verändert sich am ehesten, während die `violine`-Definition sich wahrscheinlich mit einer neuen Programmversion nicht verändern wird.

```
violine = \relative c'' {
g4 c'8. e16
}
...
\score {
  \new GrandStaff {
    \new Staff {
      \violine
    }
  }
}
```
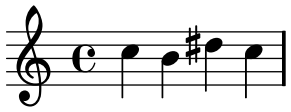
- **Trennen Sie Einstellungen von den Noten**. Diese Empfehlung wurde schon im Abschnitt Abschnitt 5.1.1 [Allgemeine Vorschläge], Seite 139 gegeben, aber für große Projekte ist es unumgänglich. Muss z. B. die Definition für `fdannp` verändert werden, so braucht man es nur einmal vorzunehmen und die Noten in der Geigenstimme, `violin`, bleiben unberührt.

```
fdannp = _\markup{
  \dynamic f \italic \small { 2nd } \hspace #0.1 \dynamic p }
violin = \relative c'' {
g4\fdannp c'8. e16
}
```

5.1.4 Tipparbeit sparen durch Bezeichner und Funktionen
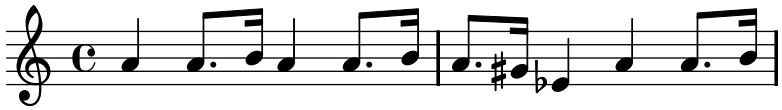
Bis jetzt haben Sie immer etwa solche Noten gesehen:

```
HornNoten = \relative c'' { c4 b dis c }
\score {
  {
    \HornNoten
  }
}
```
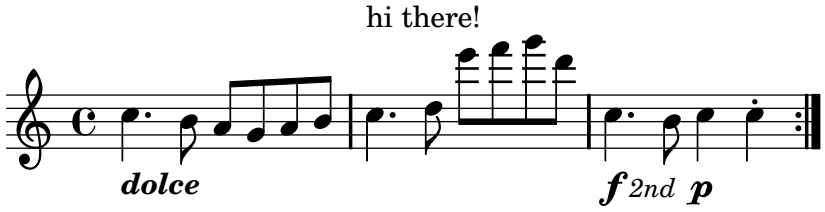


Das könnte auch nützlich in Minimal-Music sein:

```
FramentA = \relative c'' { a4 a8. b16 }
FragmentB = \relative c'' { a8. gis16 ees4 }
Geige = \new Staff { \FramentA \FramentA \FragmentB \FramentA }
\score {
  {
    \Geige
  }
}
```



Sie können diese Bezeichner oder Variablen aber auch für (eigene) Einstellungen verwenden:

```
dolce = \markup{ \italic \bold dolce }
AbstandText = { \once \override TextScript #'padding = #5.0 }
FdannP=_\markup{ \dynamic f \italic \small { 2nd } \hspace #0.1 \dynamic p }
Geige = \relative c'' {
  \repeat volta 2 {
    c4._\dolce b8 a8 g a b |
    \AbstandText
    c4.^"hi there!" d8 e' f g d |
    c,4.\FdannP b8 c4 c-. |
  }
}
\score {
  {
    \Geige
  }
\layout{ragged-right=##t}
}
```



Die Variablen haben in diesem Beispiel deutlich die Tipparbeit erleichtert. Aber es lohnt sich, sie zu einzusetzen, auch wenn man sie nur einmal anwendet, denn sie vereinfachen die Struktur. Hier ist das vorangegangene Beispiel ohne Variablen. Es ist sehr viel komplizierter zu lesen, besonders die letzte Zeile.
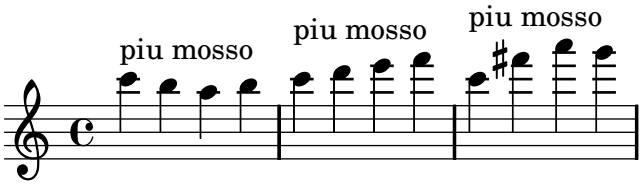
```
violin = \relative c'' {
  \repeat volta 2 {
    c4._\markup{ \italic \bold dolce } b8 a8 g a b |
    \once \override TextScript #'padding = #5.0
    c4.^"hi there!" d8 e' f g d |
    c,4.\markup{ \dynamic f \italic \small { 2nd }
      \hspace #0.1 \dynamic p } b8 c4 c-. |
  }
}
```

Bis jetzt wurde nur statische Substitution vorgestellt – wenn LilyPond den Befehl `\padText` findet, wird er ersetzt durch durch unsere vorherige Definition (alles, was nach dem `padtext =` kommt).

LilyPond kennt aber auch nicht-statische Substitutionen (man kann sie sich als Funktionen vorstellen).

```
AbstandText =
#(define-music-function (parser location padding) (number?)
  #{
    \once \override TextScript #'padding = #$padding
  #})

\relative c''' {
  c4^"piu mosso" b a b
  \AbstandText #1.8
  c4^"piu mosso" d e f
  \AbstandText #2.6
  c4^"piu mosso" fis a g
}
```



Die Benutzung von Variablen hilft auch, viele Schreibarbeit zu vermeiden, wenn die Eingabesyntax von LilyPond sich verändert (siehe auch Abschnitt 5.2.1 [Alte Dateien aktualisieren], Seite 146). Wenn nur eine einzige Definition (etwa `\dolce`) für alle Dateien verwendet wird (vgl. Abschnitt 5.1.5 [Stil-Dateien], Seite 142), muss nur diese einzige Definition verändert werden, wenn sich die Syntax ändert. Alle Verwendungen des Befehles beziehen sich dann auf die neue Definition.
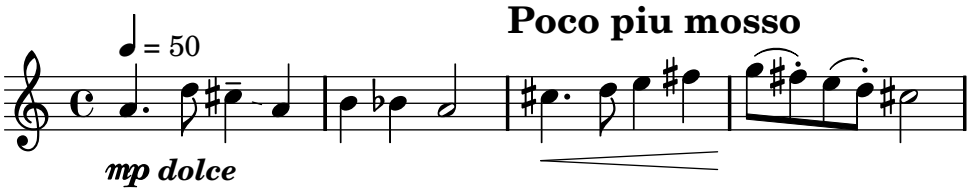
5.1.5 Stil-Dateien

Die Ausgabe, die LilyPond erstellt, kann sehr stark modifiziert werden, siehe Kapitel 4 [Die Ausgabe verändern], Seite 86 für Einzelheiten. Aber wie kann man diese Änderungen auf eine ganze Serie von Dateien anwenden? Oder die Einstellungen von den Noten trennen? Das Verfahren ist ziemlich einfach.

Hier ist ein Beispiel. Es ist nicht schlimm, wenn Sie nicht auf Anhieb die Abschnitte mit den ganzen `#()` verstehen. Das wird im Kapitel Abschnitt 4.6.5 [Fortgeschrittene Optimierungen mit Scheme], Seite 137 erklärt.

```
mpdolce = #(make-dynamic-script (markup #:hspace 1 #:translate (cons 5 0)
  #:line(#:dynamic "mp" #:text #:italic "dolce" )))
tempoZeichen = #(define-music-function (parser location markp) (string?)
#{
  \once \override Score . RehearsalMark #'self-alignment-X = #left
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $markp }
#})

\relative c'' {
  \tempo 4=50
  a4.\mpdolce d8 cis4--\glissando a | b4 bes a2
  \tempoZeichen "Poco piu mosso"
  cis4.\< d8 e4 fis | g8(\! fis)-. e( d)-. cis2
}
```



Es treten einige Probleme mit überlappenden Symbolen auf. Sie werden beseitigt mit den Tricks aus dem Kapitel Abschnitt 4.5.1 [Verschieben von Objekten], Seite 116. Aber auch die `mpdolce` und `tempoMark`-Definitionen können verbessert werden. Sie produzieren das Ergebnis, das gewünscht ist, aber es wäre schön, sie auch in anderen Stücken verwenden zu können. Man könnte sie natürlich einfach kopieren und in die anderen Dateien einfügen, aber das ist lästig. Die Definitionen verbleiben auch in der Notendatei und diese `#()` sehen nicht wirklich schön aus. Sie sollen in einer anderen Datei versteckt werden:
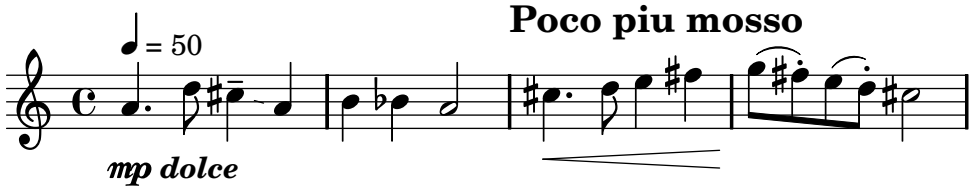
```
%%% speichern in einer Datei "definitions.ly"
mpdolce = #(make-dynamic-script (markup #:hspace 1 #:translate (cons 5 0)
  #:line(#:dynamic "mp" #:text #:italic "dolce" )))
tempoMark = #(define-music-function (parser location markp) (string?)
#{
  \once \override Score . RehearsalMark #'self-alignment-X = #left
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $markp }
#})
```

Jetzt muss natürlich noch die Notendatei angepasst werden (gespeichert unter dem Namen ‘`"music.ly"`’).

```
\include "definitions.ly"

\relative c'' {
  \tempo 4=50
  a4.\mpdolce d8 cis4--\glissando a | b4 bes a2
  \once \override Score.RehearsalMark #'padding = #2.0
  \tempoMark "Poco piu mosso"
  cis4.\< d8 e4 fis | g8(\! fis)-. e( d)-. cis2
```
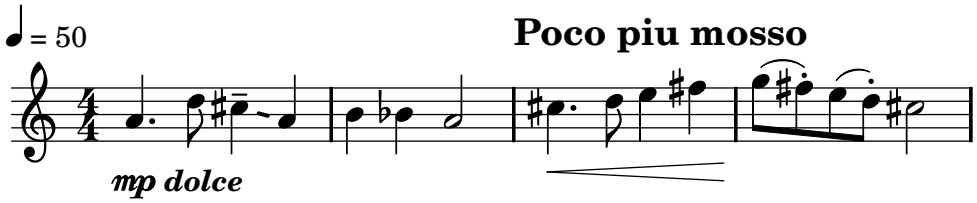
```
}
```



Das sieht schon besser aus, aber es sind noch einige Verbesserungen möglich. Das Glissando ist schwer zu sehen, also soll es etwas dicker erscheinen und dichter an den Notenköpfen gesetzt werden. Das Metronom-Zeichen soll über dem Schlüssel erscheinen, nicht über der ersten Note. Und schließlich kann unser Kompositionsprofessor „C“-Taktangaben überhaupt nicht leiden, also müssen sie in „4/4“ verändert werden.

Diese Veränderungen sollten Sie aber nicht in der ‘`music.ly`’-Datei vornehmen. Ersetzen Sie die ‘`definitions.ly`’-Datei hiermit:

```
%%%  definitions.ly
mpdolce = #(make-dynamic-script (markup #:hspace 1 #:translate (cons 5 0)
  #:line( #:dynamic "mp" #:text #:italic "dolce" )))
tempoMark = #(define-music-function (parser location markp) (string?)
#{
  \once \override Score . RehearsalMark #'self-alignment-X = #left
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $markp }
#})
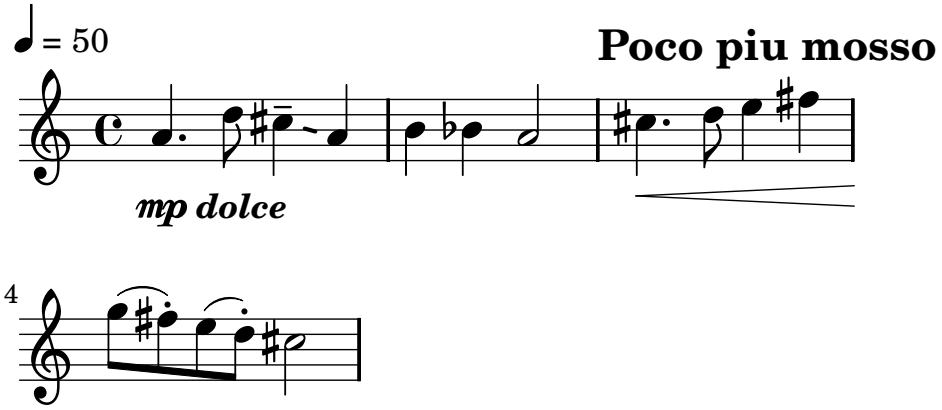
\layout{
  \context { \Score
    \override MetronomeMark #'extra-offset = #'(-9 . 0)
    \override MetronomeMark #'padding = #'3
  }
  \context { \Staff
    \override TimeSignature #'style = #'numbered
  }
  \context { \Voice
    \override Glissando #'thickness = #3
    \override Glissando #'gap = #0.1
  }
}
```



Das sieht schon besser aus! Aber angenommen Sie möchten dieses Stück jetzt veröffentlichen. Ihr Kompositionsprofessor mag die „C“-Taktangaben nicht, aber Sie finden sie irgendwie schöner. Also kopieren Sie die Datei ‘`definitions.ly`’ nach ‘`web-publish.ly`’ und verändern diese. Weil die Noten in einer PDF-Datei auf dem Bildschirm angezeigt werden sollen, bietet es sich auch an, die gesamte Ausgabe zu vergrößern.

```
%%%  definitions.ly
mpdolce = #(make-dynamic-script (markup #:hspace 1 #:translate (cons 5 0)
  #:line( #:dynamic "mp" #:text #:italic "dolce" )))
tempoMark = #(define-music-function (parser location markp) (string?)
#{
  \once \override Score . RehearsalMark #'self-alignment-X = #left
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $markp }
#})

#(set-global-staff-size 23)
\layout{
  \context { \Score
    \override MetronomeMark #'extra-offset = #'(-9 . 0)
    \override MetronomeMark #'padding = #'3
  }
  \context { \Staff
  }
  \context { \Voice
    \override Glissando #'thickness = #3
    \override Glissando #'gap = #0.1
  }
}
```



In der Notendatei muss jetzt nur noch `\include "definitions.ly"` durch `\include "web-publish.ly"` ausgetauscht werden. Das könnte man natürlich noch weiter vereinfachen. Also eine Datei '`definitions.ly`', die nur die Definitionen von `mpdolce` und `tempoMark` enthält, eine Datei '`web-publish.ly`', die alle die Änderungen für den `\layout`-Abschnitt enthält und eine Datei '`university.ly`' für eine Ausgabe, die den Wünschen des Professors entspricht. Der Anfang der '`music.ly`'-Datei würde dann so aussehen:

```
\include "definitions.ly"

%%%  Nur eine der beiden Zeilen auskommentieren!
\include "web-publish.ly"
%\include "university.ly"
```

Durch diese Herangehensweise kann auch bei der Erstellung von nur einer Ausgabeversion Arbeit gespart werden. Ich benutze ein halbes Dutzend verschiedener Stilvorlagen für meine Projekte. Jede Notationsdatei fängt an mit `\include "../global.ly"`, welches folgenden Inhalt hat:

```
%%%   global.ly
\version "2.12.0"
#(ly:set-option 'point-and-click #f)
\include "../init/init-defs.ly"
\include "../init/init-layout.ly"
\include "../init/init-headers.ly"
\include "../init/init-paper.ly"
```

5.2 Wenn etwas nicht funktioniert

5.2.1 Alte Dateien aktualisieren

Die Syntax von LilyPond verändert sich ab und zu. Wenn LilyPond besser wird, muss auch die Syntax (Eingabesprache) entsprechend angepasst werden. Teilweise machen diese Veränderungen die Eingabesprache einfacher lesbar, teilweise dienen sie dazu, neue Eigenschaften des Programmes benutzbar zu machen.

LilyPond stellt ein Programm bereit, das Aktualisierungen vereinfacht: `convert-ly`. Einzelheiten zur Programmbenutzung finden sich in Abschnitt “Dateien mit convert-ly aktualisieren” in *Anwendungsbenutzung*.

Leider kann `convert-ly` nicht alle Veränderungen der Syntax berücksichtigen. Hier werden einfache „Suchen und Ersetzen“-Veränderungen vorgenommen (wie etwa `raggedright` zu `ragged-right`), aber einige Veränderungen sind zu kompliziert. Die Syntax-Veränderungen, die das Programm nicht berücksichtigt, sind im Kapitel Abschnitt “Dateien mit convert-ly aktualisieren” in *Anwendungsbenutzung* aufgelistet.

Zum Beispiel wurden in LilyPond 2.4 und früheren Versionen Akzente und Umlaute mit LaTeX-Befehlen eingegeben, ein „No\"el“ etwa ergäbe das französische Wort für Weihnachten. In LilyPond 2.6 und höher müssen diese Sonderzeichen direkt als utf-8-Zeichen eingegeben werden, in diesem Fall also „ë“. `convert-ly` kann nicht alle dieser LaTeX-Befehle verändern, das muss manuell vorgenommen werden.

5.2.2 Fehlersuche (alles auseinandernehmen)

Früher oder später werden Sie in die Lage kommen, dass LilyPond Ihre Datei nicht kompilieren will. Die Information, die LilyPond während der Übersetzung gibt, können Ihnen helfen, den Fehler zu finden, aber in vielen Fällen müssen Sie nach der Fehlerquelle auf die Suche gehen.

Die besten Hilfsmittel sind in diesem Fall das Zeilen- und Blockkommentar (angezeigt durch `%` bzw. `%{ ... %}`). Wenn Sie nicht bestimmen können, wo sich das Problem befindet, beginnen Sie damit, große Teile des Quelltextes auszukommentieren. Nachdem Sie einen Teil auskommentiert haben, versuchen Sie, die Datei erneut zu übersetzen. Wenn es jetzt funktioniert, muss sich das Problem innerhalb der Kommentare befinden. Wenn es nicht funktioniert, müssen Sie weitere Teile auskommentieren bis sie eine Version haben, die funktioniert.

In Extremfällen bleibt nur noch solch ein Beispiel übrig:

```
\score {
  <<
    % \melody
    % \harmony
    % \bass
  >>
  \layout{}
}
```

(also eine Datei ohne Noten).

Geben Sie nicht auf, wenn das vorkommen sollte. Nehmen Sie das Kommentarzeichen von einem Teil wieder weg, sagen wir der Bassstimme, und schauen Sie, ob es funktioniert. Wenn nicht, dann kommentieren Sie die gesamte Bassstimme aus, aber nicht den `\bass`-Befehl in dem `\score`-Abschnitt:

```
bass = \relative c' {
%{
  c4 c c c
  d d d d
%}
}
```

Jetzt beginnen Sie damit, langsam Stück für Stück der Bassstimme wieder hineinzunehmen, bis Sie die problematische Zeile finden.

Eine andere nützliche Technik zur Problemlösung ist es, Abschnitt 5.2.3 [Minimalbeispiele], Seite 147 zu konstruieren.

5.2.3 Minimalbeispiele

Ein Minimalbeispiel ist eine Beispieldatei, die so klein wie möglich ist. Diese Beispiele sind sehr viel einfacher zu verstehen als die langen Originaldateien. Minimalbeispiele werden benutzt, um

- Fehlerberichte zu erstellen,
- eine Hilfeanfrage an die E-Mail-Liste zu schicken,
- Ein Beispiel zur LilyPond Schnipselsammlung hinzuzufügen.

Um ein Beispiel zu konstruieren, das so klein wie möglich ist, gibt es eine einfache Regel: Alles nicht Notwendige entfernen. Wenn Sie unnötige Teile einer Datei entfernen, bietet es sich an, sie auszukommentieren und nicht gleich zu löschen. Auf diese Weise können Sie eine Zeile leicht wieder mit aufnehmen, sollten Sie sie doch brauchen, anstatt sie von Anfang an neu zu schreiben.

Es gibt zwei Ausnahmen dieser „So klein wie möglich“-Regel:

- Fügen Sie immer einen `\version`Befehl ein.
- Wenn es möglich ist, benutzen Sie `\paper{ ragged-right = ##t }` am Beginn des Beispiels.

Der Sinn der Minimalbeispiele ist, dass sie einfach lesbar sind:

- Vermeiden Sie es, komplizierte Noten, Schlüssel oder Taktangaben zu verwenden, es sei denn, Sie wollen genau an diesen Elementen etwas demonstrieren.
- Benutzen Sie keine `\override`-Befehle, wenn sie nicht der Zweck des Beispieles sind.

5.3 Partituren und Stimmen

Orchesternoten werden alle zweimal gesetzt. Erstens als Stimmen für die Musiker, und dann als große Partitur für den Dirigenten. Mit Variablen kann hier doppelte Arbeit erspart werden. Die Musik muss nur einmal eingegeben werden und wird in einer Variable abgelegt. Der Inhalt dieser Variable wird dann benutzt, um sowohl die Stimme als auch die Partitur zu erstellen.

Es bietet sich an, die Noten in eigenen Dateien zu speichern. Sagen wir beispielsweise, dass in der Datei ‘`Horn-Noten.ly`’ die folgenden Noten eines Duetts für Horn und Fagott gespeichert sind:

```
HornNoten = \relative c {
  \time 2/4
  r4 f8 a cis4 f e d
}
```

Daraus wird dann eine eigene Stimme gemacht, indem folgende Datei erstellt wird:
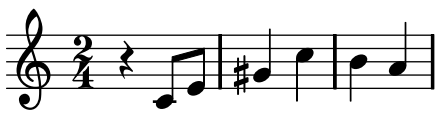
```
\include "Horn-Noten.ly"
\header {
  instrument = "Horn in F"
}

{
 \transpose f c' \HornNoten
}
```

Die Zeile

```
\include "Horn-Noten.ly"
```

setzt den Inhalt der Datei ‘`Horn-Noten.ly`’ an die Stelle des Befehls in die aktuelle Datei. Damit besteht also eine Definition für `HornNoten`, so dass die Variable verwendet werden kann. Der Befehl `\transpose f c'` zeigt an, dass das Argument, also `\HornNoten`, um eine Quinte nach oben transponiert wird. Klingendes ‚f‘ wird also als `c'` notiert. Das entspricht der Notation eines Waldhorns in F. Die Transposition zeigt die folgende Ausgabe:
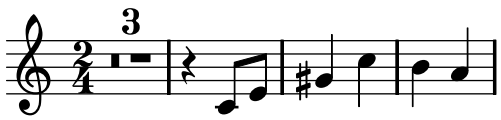


In der Musik für mehrere Instrumente kommt es oft vor, dass eine Stimme für mehrere Takte nicht spielt. Das wird mit einer besonderen Pause angezeigt, dem Pausenzeichen für mehrere Takte (engl. multi-measure rest). Sie wird mit dem *großen* Buchstaben ‘`R`’ eingegeben, gefolgt von einer Dauer (`1` für eine Ganze, `2` für eine Halbe usw.). Indem man die Dauer multipliziert, können längere Pausen erstellt werden. Z. B. dauert diese Pause drei Takte eines 2/4-Taktes:

```
R2*3
```

Wenn die Stimme gedruckt wird, müssen diese Pausen zusammengezogen werden. Das wird durch eine Variable erreicht:

```
\set Score.skipBars = ##t
```

Dieser Befehl setzt die Eigenschaft des `skipBars` („überspringe Takte“) auf wahr (`##t`). Wenn diese Option und die Pause zu der Musik des Beispiels gesetzt wird, erhält man folgendes Ergebnis:

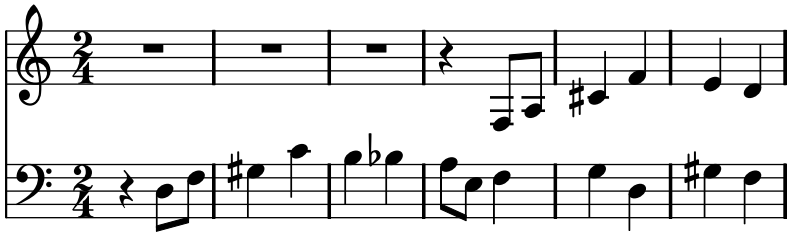


Die Partitur wird erstellt, indem alle Noten zusammengesetzt werden. Angenommen, die andere Stimme trägt den Namen `FagottNoten` und ist in der Datei ‘`Fagott-Noten.ly`’ gespeichert. Die Partitur sieht dann folgendermaßen aus:

```
\include "Fagott-Noten.ly"
\include "Horn-Noten.ly"

<<
  \new Staff \HornNoten
  \new Staff \FagottNoten
>>
```

Und mit LilyPond übersetzt:



Anhang A Vorlagen

Dieser Abschnitt des Handbuches enthält Vorlagen, in denen die LilyPond-Partitur schon eingerichtet ist. Sie müssen nur noch Ihre Noten einfügen, die Datei mit LilyPond übersetzen und sich an dem schönen Notenbild erfreuen!

A.1 Ein einzelnes System
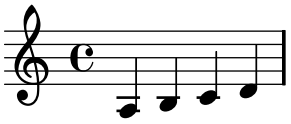
A.1.1 Nur Noten

Das erste Beispiel zeigt ein Notensystem mit Noten, passend für ein Soloinstrument oder ein Melodiefragment. Kopieren Sie es und fügen Sie es in Ihre Datei ein, schreiben Sie die Noten hinzu, und Sie haben eine vollständige Notationsdatei.

```
\version "2.12.1"
Melodie = \relative c' {
  \clef treble
  \key c \major
  \time 4/4

  a4 b c d
}

\score {
  \new Staff \Melodie
  \layout { }
  \midi { }
}
```



A.1.2 Noten und Text

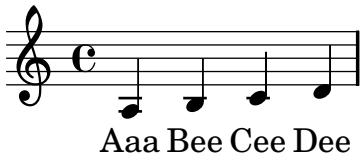
Das nächste Beispiel zeigt eine einfache Melodie mit Text. Kopieren Sie es in Ihre Datei, fügen Sie Noten und Text hinzu und übersetzen Sie es mit LilyPond. In dem Beispiel wird die automatische Balkenverbindung ausgeschaltet (mit dem Befehl `\autoBeamOff`), wie es für Vokalmusik üblich ist. Wenn Sie die Balken wieder einschalten wollen, müssen Sie die entsprechende Zeile entweder ändern oder auskommentieren.

```
\version "2.12.1"
Melodie = \relative c' {
  \clef treble
  \key c \major
  \time 4/4

  a4 b c d
}

Text = \lyricmode {
  Aaa Bee Cee Dee
}
```

```
\score{
  <<
    \new Voice = "eins" {
      \autoBeamOff
      \Melodie
    }
    \new Lyrics \lyricsto "eins" \Text
  >>
  \layout { }
  \midi { }
}
```



A.1.3 Noten und Akkordbezeichnungen
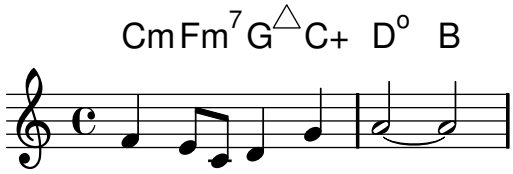
Wollen Sie ein Liedblatt mit Melodie und Akkorden schreiben? Hier ist das richtige Beispiel für Sie!

```
Melodie = \relative c' {
  \clef treble
  \key c \major
  \time 4/4

  f4 e8[ c] d4 g
  a2 ~ a
}

Harmonien = \chordmode {
  c4:m f:min7 g:maj c:aug
  d2:dim b:sus
}

\score {
  <<
    \new ChordNames {
      \set chordChanges = ##t
      \Harmonien
    }
    \new Staff \Melodie
  >>
  \layout{ }
  \midi { }
}
```



A.1.4 Noten, Text und Akkordbezeichnungen

Mit diesem Beispiel können Sie einen Song mit Melodie, Text und Akkorden schreiben.

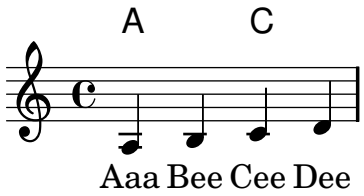
```
Melodie = \relative c' {
  \clef treble
  \key c \major
  \time 4/4

  a4 b c d
}

Text = \lyricmode {
  Aaa Bee Cee Dee
}

Harmonien = \chordmode {
  a2 c
}

\score {
  <<
    \new ChordNames {
      \set chordChanges = ##t
      \Harmonien
    }
    \new Voice = "eins" { \autoBeamOff \Melodie }
    \new Lyrics \lyricsto "eins" \Text
  >>
  \layout { }
  \midi { }
}
```



A.2 Klaviervorlagen

A.2.1 Piano Solo

Hier ein einfaches Klaviersystem.

```
oben = \relative c'' {
  \clef treble
  \key c \major
  \time 4/4
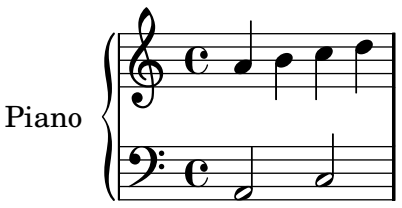
  a4 b c d
}

unten = \relative c {
```

```
  \clef bass
  \key c \major
  \time 4/4

  a2 c
}

\score {
  \new PianoStaff <<
    \set PianoStaff.instrumentName = #"Piano  "
    \new Staff = "oben" \oben
    \new Staff = "unten" \unten
  >>
  \layout { }
  \midi { }
}
```



A.2.2 Klavier und Gesangstimme

Das nächste Beispiel ist typisch für ein Lied: Im oberen System die Melodie mit Text, darunter Klavierbegleitung.

```
Melodie = \relative c'' {
  \clef treble
  \key c \major
  \time 4/4

  a b c d
}

Text = \lyricmode {
  Aaa Bee Cee Dee
}

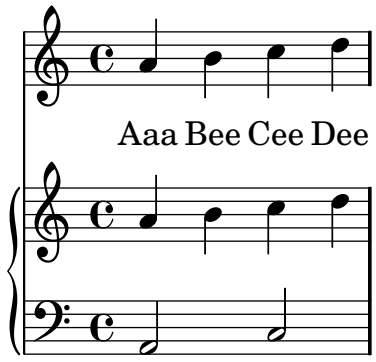
oben = \relative c'' {
  \clef treble
  \key c \major
  \time 4/4

  a4 b c d
}

unten = \relative c {
  \clef bass
  \key c \major
  \time 4/4
```

```
  a2 c
}

\score {
  <<
    \new Voice = "Melodie" { \autoBeamOff \Melodie }
    \new Lyrics \lyricsto Melodie \Text
    \new PianoStaff <<
      \new Staff = "oben" \oben
      \new Staff = "unten" \unten
    >>
  >>
  \layout {
    \context { \RemoveEmptyStaffContext }
  }
  \midi { }
}
```



A.2.3 Klavier mit zentriertem Text

Anstatt ein eigenes System für Melodie und Text zu schreiben, können Sie den Text auch zwischen die beiden Klaviersysteme schreiben (und damit das zusätzliche System für die Gesangstimme auslassen).

```
oben = \relative c'' {
  \clef treble
  \key c \major
  \time 4/4

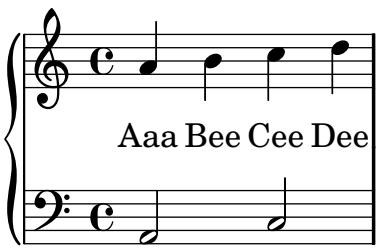
  a4 b c d
}

unten = \relative c {
  \clef bass
  \key c \major
  \time 4/4

  a2 c
}
```

```
Text = \lyricmode {
  Aaa Bee Cee Dee
}

\score {
  \new GrandStaff <<
    \new Staff = oben { \new Voice = "Sänger" \oben }
    \new Lyrics \lyricsto "Sänger" \Text
    \new Staff = unten { \unten }
  >>
  \layout {
    \context {
      \GrandStaff
      \accepts "Lyrics"
    }
    \context {
      \Lyrics
      \consists "Bar_engraver"
    }
  }
  \midi { }
}
```



A.2.4 Klavier mit zentrierten Lautstärkebezeichnungen

In der meisten Klaviernotation werden die Dynamikzeichen zwischen den beiden Systemen zentriert. Für LilyPond muss man die Einstellungen etwas anpassen, aber Sie können ja das angepasste Beispiel von hier kopieren.

```
global = {
  \key c \major
  \time 4/4
}

oben = \relative c'' {
  \clef treble
  a4 b c d
}

unten = \relative c {
  \clef bass
  a2 c
}

Dynamik = {
```

```
  s2\fff\> s4 s\!\pp
}

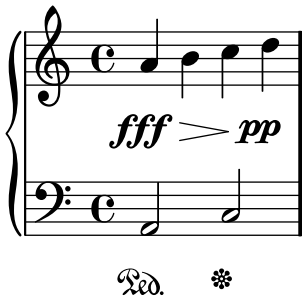
Pedal = {
  s2\sustainOn s\sustainOff
}

\score {
  \new PianoStaff = "PianoStaff_pf" <<
    \new Staff = "Staff_pfUpper" \oben
    \new Dynamics = "Dynamics_pf" \Dynamik
    \new Staff = "Staff_pfLower" << \unten >>
    \new Dynamics = "pedal" \Pedal
  >>

  \layout {
    % Dynamik-Kontext definieren
    \context {
      \type "Engraver_group"
      \name Dynamics
      \alias Voice
      \consists "Output_property_engraver"
      \consists "Piano_pedal_engraver"
      \consists "Script_engraver"
      \consists "New_dynamic_engraver"
      \consists "Dynamic_align_engraver"
      \consists "Text_engraver"
      \consists "Skip_event_swallow_translator"
      \consists "Axis_group_engraver"

      pedalSustainStrings = #'("Ped." "*Ped." "*")
      pedalUnaCordaStrings = #'("una corda" "" "tre corde")
      \override DynamicLineSpanner #'Y-offset = #0
      \override TextScript #'font-size = #2
      \override TextScript #'font-shape = #'italic
      \override VerticalAxisGroup #'minimum-Y-extent = #'(-1 . 1)
    }
    % PianoStaff-Kontext verändern, dass er Dynamics-Kontext akzeptiert
    \context {
      \PianoStaff
      \accepts Dynamics
    }
  }
}

\score {
  \new PianoStaff = "PianoStaff_pf" <<
    \new Staff = "Staff_pfUpper" << \global \oben \Dynamik \Pedal >>
    \new Staff = "Staff_pfLower" << \global \unten \Dynamik \Pedal >>
  >>
  \midi { }
}
```



A.3 Streichquartett

A.3.1 Streichquartett

Dieses Beispiel demonstriert die Partitur für ein Streichquartett. Hier wird auch eine „`\global`“-Variable für Taktart und Vorzeichen benutzt.

```
global= {
  \time 4/4
  \key c \major
}

GeigeEins = \new Voice \relative c'' {
  \set Staff.instrumentName = #"Violin 1 "

  c2 d
  e1

  \bar "|."
}

GeigeZwei = \new Voice \relative c'' {
  \set Staff.instrumentName = #"Violin 2 "

  g2 f
  e1

  \bar "|."
}

Bratsche = \new Voice \relative c' {
  \set Staff.instrumentName = #"Viola "
  \clef alto

  e2 d
  c1

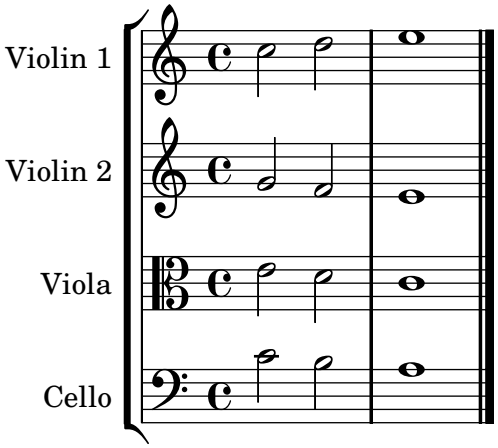
  \bar "|."
}

Cello = \new Voice \relative c' {
  \set Staff.instrumentName = #"Cello "
  \clef bass

  c2 b
  a1
```

```
  \bar "|."
}

\score {
  \new StaffGroup <<
    \new Staff << \global \GeigeEins >>
    \new Staff << \global \GeigeZwei >>
    \new Staff << \global \Bratsche >>
    \new Staff << \global \Cello >>
  >>
  \layout { }
  \midi { }
}
```



A.3.2 Streichquartettstimmen

Mit diesem Beispiel können Sie ein schönes Streichquartett notieren, aber wie gehen Sie vor, wenn Sie Stimmen brauchen? Das Beispiel oben hat gezeigt, wie Sie mit Variablen einzelne Abschnitte getrennt voneinander notieren können. Im nächsten Beispiel wird nun gezeigt, wie Sie mit diesen Variablen einzelne Stimmen erstellen.

Sie müssen das Beispiel in einzelne Dateien aufteilen; die Dateinamen sind in den Kommentaren am Anfang jeder Datei enthalten. `piece.ly` enthält die Noten. Die anderen Dateien – `score.ly`, `vn1.ly`, `vn2.ly`, `vla.ly` und `vlc.ly` – erstellen daraus die entsprechenden Stimmen bzw. die Partitur (`score.ly`). Mit `ag` wird den Stimmen ein Name zugewiesen, auf den zurückgegriffen werden kann.

```
%%%%% piece.ly
%%%%% (Globale Definitionen)

global= {
  \time 4/4
  \key c \major
}

Geigeeins = \new Voice { \relative c''{
  \set Staff.instrumentName = #"Violin 1 "

  c2 d e1
```

```
\bar "|." }}   %*********************************
Geigezwei = \new Voice { \relative c''{
  \set Staff.instrumentName = #"Violin 2 "

  g2 f e1

\bar "|." }}   %*********************************
Bratsche = \new Voice { \relative c' {
  \set Staff.instrumentName = #"Viola "
  \clef alto

  e2 d c1

\bar "|." }}   %*********************************
Cello = \new Voice { \relative c' {
  \set Staff.instrumentName = #"Cello "
  \clef bass

  c2 b a1

\bar "|."}}   %*********************************

Noten = {
  <<
    \tag #'score \tag #'vn1 \new Staff { << \global \Geigeeins >> }
    \tag #'score \tag #'vn2 \new Staff { << \global \Geigezwei>> }
    \tag #'score \tag #'vla \new Staff { << \global \Bratsche>> }
    \tag #'score \tag #'vlc \new Staff { << \global \Cello>> }
  >>
}

%%% Das sind die anderen Dateien, die gespeichert werden müssen

%%%%% score.ly
%%%%% (Das ist die Hauptdatei)


%\include "piece.ly"             %%% uncomment this line when using a separate file
#(set-global-staff-size 14)
\score {
  \new StaffGroup \keepWithTag #'score \Noten
  \layout { }
  \midi { }
}


%{ Diesen Block einkommentieren, wenn extra Dateien benutzt werden

%%%%% vn1.ly
%%%%% (Stimme der ersten Geige)
```

```
\include "piece.ly"
\score {
  \keepWithTag #'vn1 \Noten
  \layout { }
}


%%%%% vn2.ly
%%%%% (Stimme der zweiten Geige)

\include "piece.ly"
\score {
  \keepWithTag #'vn2 \Noten
  \layout { }
}

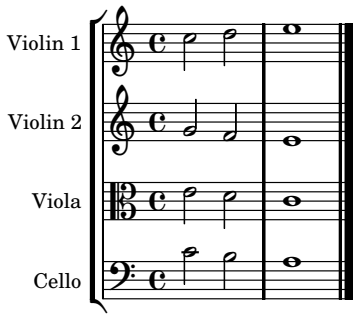

%%%%% vla.ly
%%%%% (Stimme der Bratsche)

\include "piece.ly"
\score {
  \keepWithTag #'vla \Noten
  \layout { }
}


%%%%% vlc.ly
%%%%% (Stimme des Cellos)

\include "piece.ly"
\score {
  \keepWithTag #'vlc \Noten
  \layout { }
}

%}
```



A.4 Vokalensemble

A.4.1 SATB-Partitur

Dieses Beispiel ist für vierstimmigen Gesang (SATB). Bei größeren Stücken ist es oft sinnvoll, eine allgemeine Variable zu bestimmen, die in allen Stimmen eingefügt wird. Taktart und Vorzeichen etwa sind fast immer gleich in allen Stimmen.

```
global = {
  \key c \major
  \time 4/4
}

SoprNoten = \relative c'' {
  c4 c c8[( b)] c4
}
SopranText = \lyricmode {
  hi hi hi hi
}
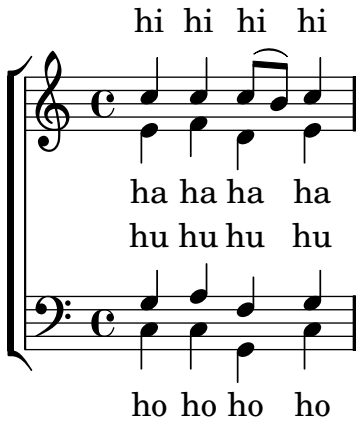
AltNoten = \relative c' {
  e4 f d e
}
AltText = \lyricmode {
  ha ha ha ha
}

TenorNoten = \relative c' {
  g4 a f g
}
TenorText = \lyricmode {
  hu hu hu hu
}

BassNoten = \relative c {
  c4 c g c
}
BassText = \lyricmode {
  ho ho ho ho
}

\score {
  \new ChoirStaff <<
    \new Lyrics = Sopran { s1 }
    \new Staff = frauen <<
      \new Voice = "Sopran" {
        \voiceOne
        << \global \SoprNoten >>
      }
      \new Voice = "Alt" {
        \voiceTwo
        << \global \AltNoten >>
      }
    >>
    \new Lyrics = "Alt" { s1 }
```

```
    \new Lyrics = "Tenor" { s1 }
    \new Staff = Männer <<
      \clef bass
      \new Voice = "Tenor" {
        \voiceOne
        << \global \TenorNoten >>
      }
      \new Voice = "Bass" {
        \voiceTwo << \global \BassNoten >>
      }
    >>
    \new Lyrics = Bass { s1 }
    \context Lyrics = Sopran \lyricsto Sopran \SopranText
    \context Lyrics = Alt \lyricsto Alt \AltText
    \context Lyrics = Tenor \lyricsto Tenor \TenorText
    \context Lyrics = Bass \lyricsto Bass \BassText
  >>
  \layout {
    \context {
      % etwas kleiner, damit der Text
      % näher am System sein kann
      \Staff
      \override VerticalAxisGroup #'minimum-Y-extent = #'(-3 . 3)
    }
  }
}
```



A.4.2 SATB-Partitur und automatischer Klavierauszug

In diesem Beispiel wird ein automatischer Klavierauszug zu der Chorpartitur hinzugefügt. Das zeigt eine der Stärken von LilyPond – man kann eine Variable mehr als einmal benutzen. Wenn Sie irgendeine Änderung an einer Chorstimme vornehmen, (etwa tenorMusic), verändert sich auch der Klavierauszug entsprechend.

```
global = {
  \key c \major
  \time 4/4
}

SoprNoten = \relative c'' {
```

```
  c4 c c8[( b)] c4
}
SopranText = \lyricmode {
  hi hi hi hi
}

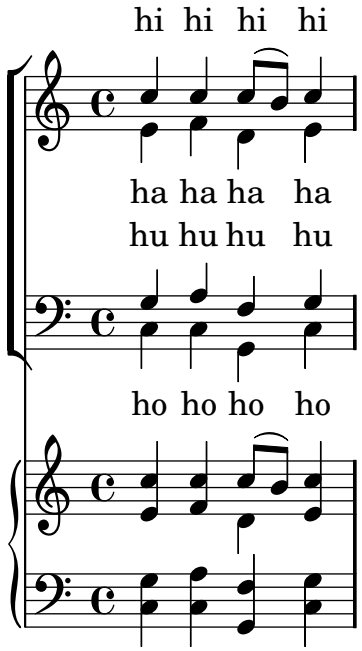
AltNoten = \relative c' {
  e4 f d e
}
AltText =\lyricmode {
  ha ha ha ha
}

TenorNoten = \relative c' {
  g4 a f g
}
TenorText = \lyricmode {
  hu hu hu hu
}

BassNoten = \relative c {
  c4 c g c
}
BassText = \lyricmode {
  ho ho ho ho
}

\score {
  <<
    \new ChoirStaff <<
      \new Lyrics = Sopran { s1 }
      \new Staff = frauen <<
        \new Voice = Sopran { \voiceOne << \global \SoprNoten >> }
        \new Voice = Alt { \voiceTwo << \global \AltNoten >> }
      >>
      \new Lyrics = Alt { s1 }
      \new Lyrics = Tenor { s1 }
      \new Staff = Männer <<
        \clef bass
        \new Voice = Tenor { \voiceOne <<\global \TenorNoten >> }
        \new Voice = Bass { \voiceTwo <<\global \BassNoten >> }
      >>
      \new Lyrics = Bass { s1 }
      \context Lyrics = Sopran \lyricsto Sopran \SopranText
      \context Lyrics = Alt \lyricsto Alt \AltText
      \context Lyrics = Tenor \lyricsto Tenor \TenorText
      \context Lyrics = Bass \lyricsto Bass \BassText
    >>
    \new PianoStaff <<
      \new Staff <<
        \set Staff.printPartCombineTexts = ##f
        \partcombine
```

```
        << \global \SoprNoten >>
        << \global \AltNoten >>
      >>
      \new Staff <<
        \clef bass
        \set Staff.printPartCombineTexts = ##f
        \partcombine
        << \global \TenorNoten >>
        << \global \BassNoten >>
      >>
    >>
  >>
  \layout {
    \context {
      % etwas kleiner, damit der Text
      % näher am System sein kann
      \Staff
      \override VerticalAxisGroup #'minimum-Y-extent = #'(-3 . 3)
    }
  }
}
```



A.4.3 SATB mit zugehörigen Kontexten

In diesem Beispiel werden die Texte mit den Befehlen `alignAboveContext` und `alignBelowContext` über und unter dem System angeordnet.

```
global = {
  \key c \major
  \time 4/4
}

SoprNoten = \relative c'' {
  c4 c c8[( b)] c4
```

```
}
SopranText = \lyricmode {
  hi hi hi hi
}

AltNoten = \relative c' {
  e4 f d e
}
AltText = \lyricmode {
  ha ha ha ha
}
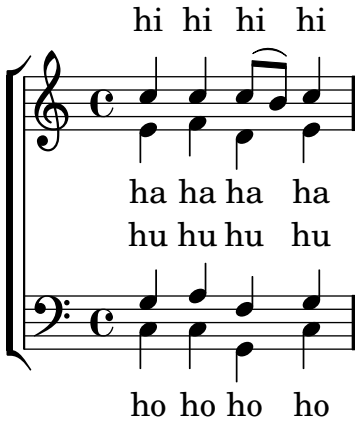
TenorNoten = \relative c' {
  g4 a f g
}
TenorText = \lyricmode {
  hu hu hu hu
}

BassNoten = \relative c {
  c4 c g c
}
BassText = \lyricmode {
  ho ho ho ho
}

\score {
  \new ChoirStaff <<
    \new Staff = frauen <<
      \new Voice = "Sopran" { \voiceOne << \global \SoprNoten >> }
      \new Voice = "Alt" { \voiceTwo << \global \AltNoten >> }
    >>
    \new Lyrics \with { alignAboveContext = frauen } \lyricsto Sopran \SopranText
    \new Lyrics \with { alignBelowContext = frauen } \lyricsto Alt \AltText
    % die Zeile oberhalb könnte mir der Zeile unterhalb entfernt werden, weil
    % der Alt-Text sowieso unter der Altstimme sein soll
    % \newLyrics\lyricstoaltos\AltText

    \new Staff = Männer <<
      \clef bass
      \new Voice = "Tenor" { \voiceOne << \global \TenorNoten >> }
      \new Voice = "Bass" { \voiceTwo << \global \BassNoten >> }
    >>
    \new Lyrics \with { alignAboveContext = Männer } \lyricsto Tenor \TenorText
    \new Lyrics \with { alignBelowContext = Männer } \lyricsto Bass \BassText
    % die Zeile oberhalb könnte mit der Zeile unterhalb ersetzt werden
    % \newLyrics\lyricstobasses\BassText
  >>
  \layout {
    \context {
      % etwas kleiner, damit der Text
      % näher am System sein kann
```

```
      \Staff
      \override VerticalAxisGroup #'minimum-Y-extent = #'(-3 . 3)
    }
  }
}
```



A.5 Vorlagen für alte Notation

A.5.1 Transkription mensuraler Musik

Bei der Transkription von Mensuralmusik ist es oft erwünscht, ein Incipit an den Anfang des Stückes zu stellen, damit klar ist, wie Tempo und Schlüssel in der Originalnotation gesetzt waren. Während heutzutage Musiker an Taktlinien gewöhnt sind, um Rhythmen schneller zu erkennen, wurden diese in der Mensuralmusik nicht verwendet. Tatsächlich ändern sich die Rhythmen auch oft alle paar Noten. Als ein Kompromiss werden die Notenlinien nicht auf dem System, sondern zwischen den Systemen geschrieben.

```
global = {
  \set Score.skipBars = ##t

  % Incipit
  \once \override Score.SystemStartBracket #'transparent = ##t
  \override Score.SpacingSpanner #'spacing-increment = #1.0 % dichter Satz
  \key f \major
  \time 2/2
  \once \override Staff.TimeSignature #'style = #'neomensural
  \override Voice.NoteHead #'style = #'neomensural
  \override Voice.Rest #'style = #'neomensural
  \set Staff.printKeyCancellation = ##f
  \cadenzaOn % Taktstriche ausschalten
  \skip 1*10
  \once \override Staff.BarLine #'transparent = ##f
  \bar "||"
  \skip 1*1 % zusätzliches \skip nötig, damit Schlüsselwechsel gedruckt wird
            % nach der Taktlinie
  \bar ""

  % haupt
  \revert Score.SpacingSpanner #'spacing-increment % CHECK: keine Auswirkung?
  \cadenzaOff % Taktlinien wieder anschalten
```

```
  \once \override Staff.Clef #'full-size-change = ##t
  \set Staff.forceClef = ##t
  \key g \major
  \time 4/4
  \override Voice.NoteHead #'style = #'default
  \override Voice.Rest #'style = #'default

  % FIXME: printKeyCancellation wieder auf #t setzen darf nicht
  % im ersten Takt nach dem Incipit auftauchen. Genauso für forceClef.
  % Darum ein zusätzlicher \skip
  \skip 1*1
  \set Staff.printKeyCancellation = ##t
  \set Staff.forceClef = ##f

  \skip 1*7 % die eigentlichen Noten

  % Finis Taktlinie durch alle Systeme setzen
  \override Staff.BarLine #'transparent = ##f

  % Finis-Taktstrich
  \bar "|."
}

DiskantusNoten = {
  \transpose c' c'' {
    \set Staff.instrumentName = #"Discantus  "

    % Incipit
    \clef "neomensural-c1"
    c'1. s2   % zwei Takte
    \skip 1*8 % acht Takte
    \skip 1*1 % Ein Takt

    % haupt
    \clef "treble"
    d'2. d'4 |
    b e' d'2 |
    c'4 e'4.( d'8 c' b |
    a4) b a2 |
    b4.( c'8 d'4) c'4 |
    \once \override NoteHead #'transparent = ##t c'1 |
    b\breve |
  }
}

DiskantusText = \lyricmode {
  % Incipit
  IV-

  % haupt
  Ju -- bi -- |
  la -- te De -- |
```

```
  o, om --
  nis ter -- |
  ra, __ om- |
  "..." |
  -us. |
}

AltNoten = {
  \transpose c' c'' {
    \set Staff.instrumentName = #"Altus  "

    % Incipit
    \clef "neomensural-c3"
    r1        % Ein Takt
    f1. s2    % zwei Takte
    \skip 1*7 % Sieben Takte
    \skip 1*1 % Ein Takt

    % haupt
    \clef "treble"
    r2 g2. e4 fis g | % zwei Takte
    a2 g4 e |
    fis g4.( fis16 e fis4) |
    g1 |
    \once \override NoteHead #'transparent = ##t g1 |
    g\breve |
  }
}

AltText = \lyricmode {
  % Incipit
  IV-

  % haupt
  Ju -- bi -- la -- te | % zwei Takte
  De -- o, om -- |
  nis ter -- ra, |
  "..." |
  -us. |
}

TenorNoten = {
  \transpose c' c' {
    \set Staff.instrumentName = #"Tenor  "

    % Incipit
    \clef "neomensural-c4"
    r\longa   % vier Takte
    r\breve   % zwei Takte
    r1        % Ein Takt
    c'1. s2   % zwei Takte
    \skip 1*1 % Ein Takt
```

```
    \skip 1*1 % Ein Takt

    % haupt
    \clef "treble_8"
    R1 |
    R1 |
    R1 |
    r2 d'2. d'4 b e' | % zwei Takte
    \once \override NoteHead #'transparent = ##t e'1 |
    d'\breve |
  }
}

TenorText = \lyricmode {
  % Incipit
  IV-

  % haupt
  Ju -- bi -- la -- te | % zwei Takte
  "..." |
  -us. |
}

BassNoten = {
  \transpose c' c' {
    \set Staff.instrumentName = #"Bassus  "

    % Incipit
    \clef "bass"
    r\maxima  % acht Takte
    f1. s2    % zwei Takte
    \skip 1*1 % Ein Takt

    % haupt
    \clef "bass"
    R1 |
    R1 |
    R1 |
    R1 |
    g2. e4 |
    \once \override NoteHead #'transparent = ##t e1 |
    g\breve |
  }
}

BassText = \lyricmode {
  % Incipit
  IV-

  % haupt
  Ju -- bi- |
  "..." |
```

```
  -us. |
}

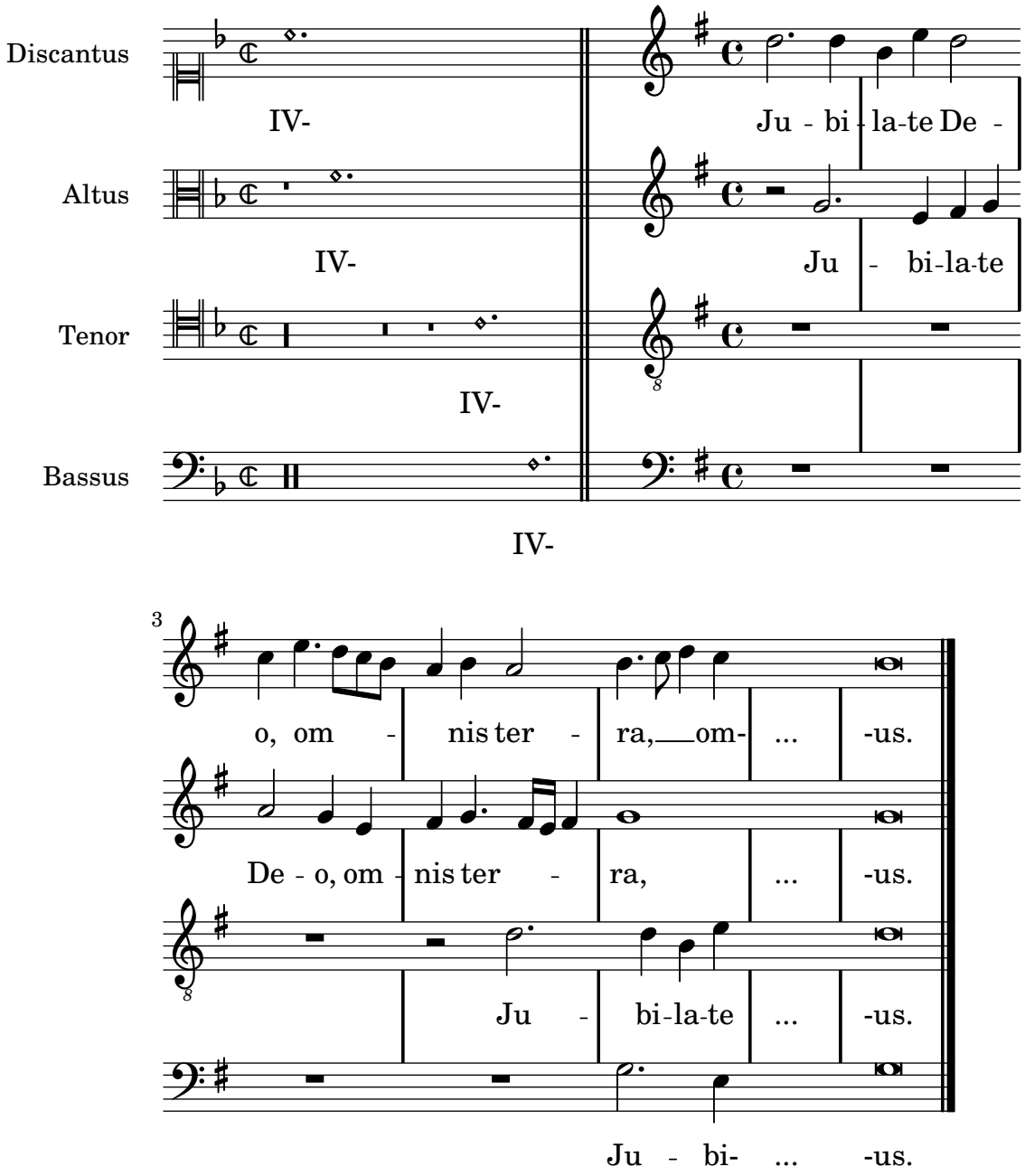
\score {
  \new StaffGroup = choirStaff <<
    \new Voice =
      "DiskantusNoten" << \global \DiskantusNoten >>
    \new Lyrics =
      "discantusLyrics" \lyricsto DiskantusNoten { \DiskantusText }
    \new Voice =
      "AltNoten" << \global \AltNoten >>
    \new Lyrics =
      "altusLyrics" \lyricsto AltNoten { \AltText }
    \new Voice =
      "TenorNoten" << \global \TenorNoten >>
    \new Lyrics =
      "tenorLyrics" \lyricsto TenorNoten { \TenorText }
    \new Voice =
      "BassNoten" << \global \BassNoten >>
    \new Lyrics =
      "bassusLyrics" \lyricsto BassNoten { \BassText }
  >>
  \layout {
    \context {
      \Score

      % keine Taktlinien in den Systemen
      \override BarLine #'transparent = ##t

      % Incipit nicht mit einer Systemklammer beginnen
      \remove "System_start_delimiter_engraver"
    }
    \context {
      \Voice

      % keine Legatobögen
      \override Slur #'transparent = ##t

      % Comment in the below "\remove" command to allow line
      % Zeilenumbrauch auch zu erlauben, wenn Noten über Taktlinie reichen
      % Der Befehl ist auskommentiert in
      % diesem kleinen Beispiel, aber für größere Partituren
      % ergeben sich bessere Zeilenumbrüche und
      % auch die Aufteilung der Noten verbessert sich, wenn der folgende Befehl benutzt
      %\remove "Forbid_line_break_engraver"
    }
  }
}
```



A.5.2 Vorlage zur Transkription von Gregorianik

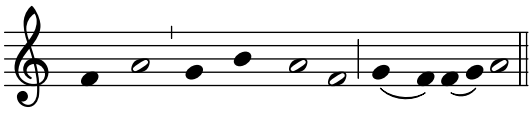
Dieses Beispiel zeigt eine moderne Transkription des Gregorianischen Chorals. Hier gibt es keine Takte, keine Notenhälse und es werden nur halbe und Viertelnoten verwendet. Zusätzliche Zeichen zeigen die Länge von Pausen an.

```
\include "gregorian.ly"

Hymnus = \relative c' {
  \set Score.timing = ##f
  f4 a2 \divisioMinima
  g4 b a2 f2 \divisioMaior
  g4( f) f( g) a2 \finalis
}

verba = \lyricmode {
  Lo -- rem ip -- sum do -- lor sit a -- met
}
```

```
\score {
  \new Staff <<
    \new Voice = "Melodie" \Hymnus
    \new Lyrics = "one" \lyricsto Melodie \verba
  >>
  \layout {
    \context {
      \Staff
      \remove "Time_signature_engraver"
      \remove "Bar_engraver"
      \override Stem #'transparent = ##t
    }
    \context {
      \Voice
      \override Stem #'length = #0
    }
    \context {
      \Score
      barAlways = ##t
    }
  }
}
```



Lorem ipsum dolor sit a-met

A.6 Jazz-Combo

Hier ist ein ziemlich kompliziertes Beispiel für ein Jazz-Ensemble. Achtung: Alle Instrumente sind in `\key c \major` (C-Dur) notiert. Das bezieht sich auf die klingende Musik: LilyPond transponiert die Tonart automatisch, wenn sich die Noten innerhalb eines `ranspose`-Abschnitts befinden.

```
\header {
  title = "Song"
  subtitle = "(tune)"
  composer = "Me"
  meter = "moderato"
  piece = "Swing"
  tagline = \markup {
    \column {
      "LilyPond example file by Amelie Zapf,"
      "Berlin 07/07/2003"
    }
  }
}

%#(set-global-staff-size16)
\include "english.ly"
```

```
%%%%%%%%%%% Einige Makros %%%%%%%%%%%%%%%%%%

sl = {
  \override NoteHead #'style = #'slash
  \override Stem #'transparent = ##t
}
nsl = {
  \revert NoteHead #'style
  \revert Stem #'transparent
}
crOn = \override NoteHead #'style = #'cross
crOff = \revert NoteHead #'style

%% Akkordbezeichnungen hierher

jazzAkkorde = { }

%%%%%%%%%%% Taktart/Tonart %%%%%%%%%%%%%%%%%

global = { \time 4/4 }

Tonart = { \key c \major }

% ############Hörner############

% ------Trompete------
trpt = \transpose c d \relative c'' {
  \Tonart
  c1 | c | c |
}
trpHarmonie = \transpose c' d {
  \jazzAkkorde
}
trompete = {
  \global
  \set Staff.instrumentName = #"Trumpet"
  \clef treble
  <<
    \trpt
  >>
}

% ------Altsaxophon------
alt = \transpose c a \relative c' {
  \Tonart
  c1 | c | c |
}
altHarmonien = \transpose c' a {
  \jazzAkkorde
}
altSax = {
```

```
  \global
  \set Staff.instrumentName = #"Alto Sax"
  \clef treble
  <<
    \alt
  >>
}

% ------Baritonsaxophon------
bari = \transpose c a' \relative c {
  \Tonart
  c1
  c1
  \sl
  d4^"Solo" d d d
  \nsl
}
bariHarmonie = \transpose c' a \chordmode {
  \jazzAkkorde s1 s d2:maj e:m7
}
bariSax = {
  \global
  \set Staff.instrumentName = #"Bari Sax"
  \clef treble
  <<
    \bari
  >>
}

% ------ Posaune ------
pos = \relative c {
  \Tonart
  c1 | c | c
}
PosHarmonie = \chordmode {
  \jazzAkkorde
}
posaune = {
  \global
  \set Staff.instrumentName = #"Trombone"
  \clef Bass
  <<
    \pos
  >>
}

% #############Rhythmus-Abschnitt##############

% ------ Gitarre ------
gtr = \relative c'' {
  \Tonart
  c1
```

```
  \sl
  b4 b b b
  \nsl
  c1
}
gtrHarmonie = \chordmode {
  \jazzAkkorde
  s1 c2:min7+ d2:maj9
}
Gitarre = {
  \global
  \set Staff.instrumentName = #"Guitar"
  \clef treble
  <<
    \gtr
  >>
}

%% ------ Klavier ------
rhOben = \relative c'' {
  \voiceOne
  \Tonart
  c1 | c | c
}
rhUnten = \relative c' {
  \voiceTwo
  \Tonart
  e1 | e | e
}

lhOben = \relative c' {
  \voiceOne
  \Tonart
  g1 | g | g
}
lhUnten = \relative c {
  \voiceTwo
  \Tonart
  c1 | c | c
}

KlavierRH = {
  \clef treble
  \global
  \set Staff.midiInstrument = #"acoustic grand"
  <<
    \new Voice = "eins" \rhOben
    \new Voice = "zwei" \rhUnten
  >>
}
KlavierLH = {
  \clef Bass
```

```
  \global
  \set Staff.midiInstrument = "acoustic grand"
  <<
    \new Voice = "eins" \lhOben
    \new Voice = "zwei" \lhUnten
  >>
}

Klavier = {
  <<
    \set PianoStaff.instrumentName = #"Piano"
    \new Staff = "oben" \KlavierRH
    \new Staff = "unten" \KlavierLH
  >>
}

% ------ Bassgitarre ------
Bass = \relative c {
  \Tonart
  c1 | c | c
}
Bass = {
  \global
  \set Staff.instrumentName = #"Bass"
  \clef Bass
  <<
    \Bass
  >>
}

% ------ Schlagzeugt ------
oben = \drummode {
  \voiceOne
  hh4 <hh sn> hh <hh sn>
  hh4 <hh sn> hh <hh sn>
  hh4 <hh sn> hh <hh sn>
}
unten = \drummode {
  \voiceTwo
  bd4 s bd s
  bd4 s bd s
  bd4 s bd s
}

SchlagInhalt = {
  \global
  <<
    \set DrumStaff.instrumentName = #"Drums"
    \new DrumVoice \oben
    \new DrumVoice \unten
  >>
}
```

```
%%%%%%%% Alles zusammengefügt: %%%%%%%%%%%%%%%%%%%%

\score {
  <<
    \new StaffGroup = "Horn" <<
      \new Staff = "trompete" \trompete
      \new Staff = "altsax" \altSax
      \new ChordNames = "bariakk" \bariHarmonie
      \new Staff = "barsisax" \bariSax
      \new Staff = "posaune" \posaune
    >>

    \new StaffGroup = "Rhythmus" <<
      \new ChordNames = "Akkorde" \gtrHarmonie
      \new Staff = "Gitarre" \Gitarre
      \new PianoStaff = "Klavier" \Klavier
      \new Staff = "Bass" \Bass
      \new DrumStaff \SchlagInhalt
    >>
  >>

  \layout {
    \context { \RemoveEmptyStaffContext }
    \context {
      \Score
      \override BarNumber #'padding = #3
      \override RehearsalMark #'padding = #2
      skipBars = ##t
    }
  }

  \midi { }
}
```

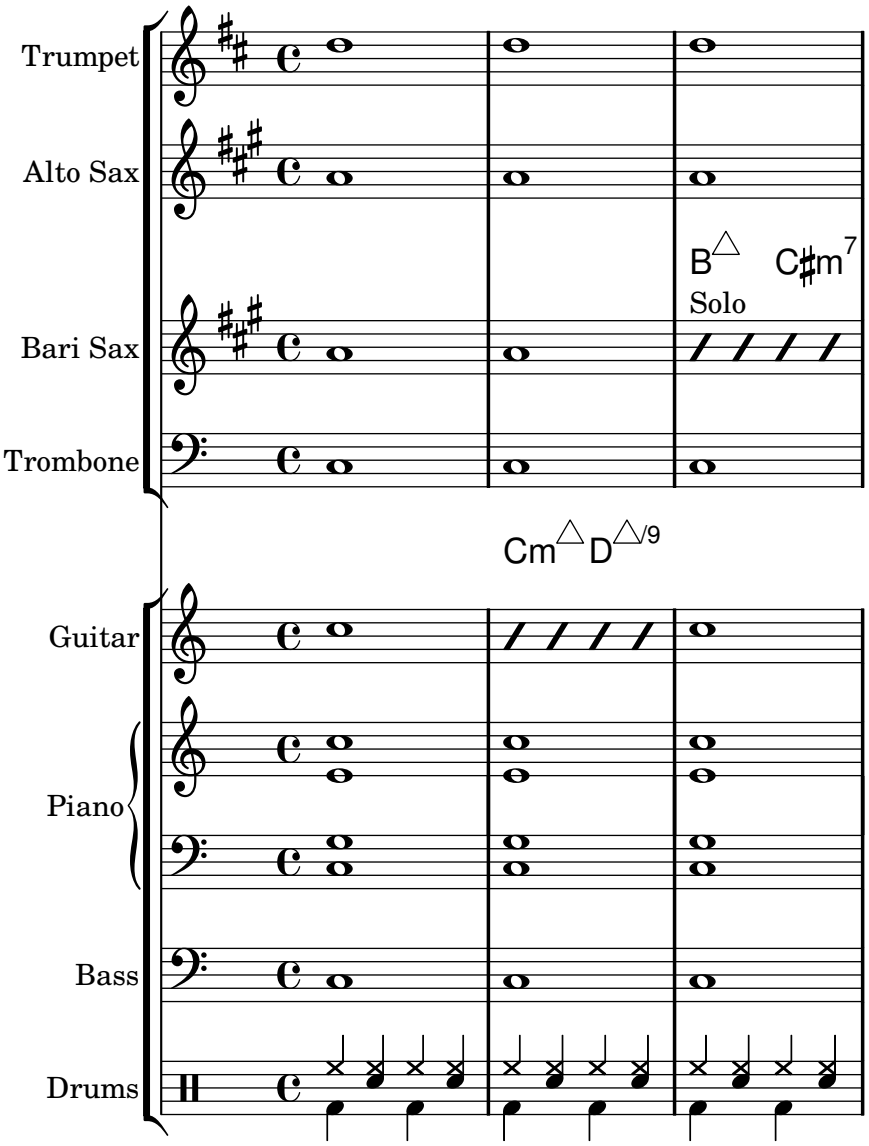
Song

(tune)

Me

moderato

Swing



A.7 Lilypond-book-Vorlagen

Diese Vorlagen können mit `lilypond-book` benutzt werden. Wenn Sie dieses Programm noch nicht kennen, lesen Sie bitte den Abschnitt Abschnitt “LilyPond-book” in *Anwendungsbenutzung*.

A.7.1 LaTeX

LilyPond-Noten können in LaTeX-Dokumente eingefügt werden.

```
\documentclass[]{article}

\begin{document}

Normaler LaTeX-Ttext.

\begin{lilypond}
\relative c'' {
a4 b c d
}
\end{lilypond}

Weiterer LaTeX-Text.
```

```
\begin{lilypond}
\relative c'' {
d4 c b a
}
\end{lilypond}
\end{document}
```

A.7.2 Texinfo

LilyPond-Noten können auch in Texinfo eingefügt werden – dieses gesamte Handbuch ist in Texinfo geschrieben.

```
\input texinfo
@node Top

Texinfo-Text

@lilypond[verbatim,fragment,ragged-right]
a4 b c d
@end lilypond

Weiterer Texinfo-Text

@lilypond[verbatim,fragment,ragged-right]
d4 c b a
@end lilypond

@bye
```

A.7.3 xelatex

```
\documentclass{article}
\usepackage{ifxetex}
\ifxetex
%xetex specific stuff
\usepackage{xunicode,fontspec,xltxtra}
\setmainfont[Numbers=OldStyle]{Times New Roman}
\setsansfont{Arial}
\else
%This can be empty if you are not going to use pdftex
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{mathptmx}%Times
\usepackage{helvet}%Helvetica
\fi
%Here you can insert all packages that pdftex also understands
\usepackage[ngerman,finnish,english]{babel}
\usepackage{graphicx}

\begin{document}
\title{A short document with LilyPond and xelatex}
\maketitle
```

```
Normal \textbf{font} commands inside the \emph{text} work,
because they \textsf{are supported by \LaTeX{} and XeteX.}
If you want to use specific commands like \verb+\XeTeX+, you
should include them again in a \verb+\ifxetex+ environment.
You can use this to print the \ifxetex \XeTeX{} command \else
XeTeX command \fi which is not known to normal \LaTeX .

In normal text you can easily use LilyPond commands, like this:

\begin{lilypond}
{a2 b c'8 c' c' c'}
\end{lilypond}

\noindent
and so on.

The fonts of snippets set with LilyPond will have to be set from
inside
of the snippet. For this you should read the AU on how to use
lilypond-book.

\selectlanguage{ngerman}
Auch Umlaute funktionieren ohne die \LaTeX -Befehle, wie auch alle
anderen
seltsamen Zeichen: ß,␣l,ã,č,,, wenn sie von der Schriftart
unterstützt werden.
\end{document}
```

Anhang B Scheme-Übung

LilyPond verwendet die Scheme-Programmiersprache sowohl als Teil der Eingabesyntax als auch als internen Mechanismus, um Programmmodule zusammenzufügen. Dieser Abschnitt ist ein sehr kurzer Überblick über die Dateneingabe mit Scheme. Wenn Sie mehr über Scheme wissen wollen, gehen Sie zu http://www.schemers.org.

Das Grundlegendste an einer Sprache sind Daten: Zahlen, Zeichen, Zeichenketten, Listen usw. Hier ist eine Liste der Datentypen, die für LilyPond-Eingabedateien relevant sind.

Boolesche Variablen
: Werte einer Booleschen Variable sind Wahr oder Falsch. Die Scheme-Entsprechung für Wahr ist `#t` und für Falsch `#f`.

Zahlen
: Zahlen werden wie üblich eingegeben, `1` ist die (ganze) Zahl Eins, während `-1.5` ist eine Gleitkommazahl (also eine nicht-ganze).

Zeichenketten
: Zeichenketten werden in doppelte Anführungszeichen gesetzt:

```
"Das ist eine Zeichenkette"
```

Zeichenketten können über mehrere Zeilen reichen:

```
"Das
ist
eine Zeichenkette"
```

Anführungszeichen und neue Zeilen können auch mit sogenannten Fluchtsequenzen eingefügt werden. Die Zeichenkette `a sagt "b"` wird wie folgt eingegeben:

```
"a sagt \"b\""
```

Neue Zeilen und Backslashe werden mit `\n` bzw. `\\` eingegeben.

In einer Notationsdatei werden kleine Scheme-Abschnitte mit der Raute (`#`) eingeleitet. Die vorigen Beispiele heißen also in LilyPond:

```
##t ##f
#1 #-1.5
#"Das ist eine Zeichenkette"
#"Das
ist
eine Zeichenkette"
```

Für den Rest dieses Abschnitts nehmen wir an, dass die Daten immer in einer LilyPond-Datei stehen, darum wird immer die Raute verwendet.

Scheme kann verwendet werden, um Berechnungen durchzuführen. Es verwendet eine *Präfix*-Syntax. Um 1 und 2 zu addieren, muss man `(+ 1 2)` schreiben, und nicht $1 + 2$, wie in traditioneller Mathematik.

```
#(+ 1 2)
  ⇒ #3
```

Der Pfeil ⇒ zeigt an, dass das Ergebnis der Auswertung von `(+ 1 2)` `3` ist. Berechnungen können geschachtelt werden und das Ergebnis einer Berechnung kann für eine neue Berechnung eingesetzt werden.

```
#(+ 1 (* 3 4))
  ⇒ #(+ 1 12)
  ⇒ #13
```

Diese Berechnungen sind Beispiele von Auswertungen. Ein Ausdruck wie `(* 3 4)` wird durch seinen Wert `12` ersetzt. Ähnlich verhält es sich mit Variablen. Nachdem eine Variable definiert ist:

```
zwoefl = #12
```

kann man sie in Ausdrücken weiterverwenden:

```
vierundzwanzig = #(* 2 zwoelf)
```

Die 24 wird in der Variablen `vierundzwanzig` gespeichert. Die gleiche Zuweisung kann auch vollständig in Scheme geschrieben werden:

```
#(define vierundzwanzig (* 2 zwoelf))
```

Der *Name* einer Variable ist auch ein Ausdruck, genauso wie eine Zahl oder eine Zeichenkette. Er wird wie folgt eingegeben:

```
#'vierundzwanzig
```

Das Apostroph `'` verhindert, dass bei der Scheme-Auswertung `vierundzwanzig` durch `24` ersetzt wird. Anstatt dessen erhalten wir die Bezeichnung `vierundzwanzig`.

Diese Syntax wird sehr oft verwendet, weil es manche Einstellungsveränderungen erfordern, dass Scheme-Werte einer internen Variable zugewiesen werden, wie etwa

```
\override Stem #'thickness = #2.6
```

Diese Anweisung verändert die Erscheinung der Notenhälse. Der Wert `2.6` wird der Variable `thickness` (Dicke) eines `Stem`-(Hals)-Objektes gleichgesetzt. `thickness` wird relativ zu den Notenlinien errechnet, in diesem Fall sind die Hälse also 2,6 mal so dick wie die Notenlinien. Dadurch werden Hälse fast zweimal so dick dargestellt, wie sie normalerweise sind. Um zwischen Variablen zu unterscheiden, die in den Quelldateien direkt definiert werden (wie `vierundzwanzig` weiter oben), und zwischen denen, die für interne Objekte zuständig sind, werden hier die ersteren „Bezeichner“ genannt, die letzteren dagegen „Eigenschaften“. Das Hals-Objekt hat also eine `thickness`-Eigenschaft, während `vierundzwanzig` ein Bezeichner ist.

Sowohl zweidimensionale Abstände (X- und Y-Koordinaten) als auch Größen von Objekten (Intervalle mit linker und rechter Begrenzung) werden als `pairs` (Paare) eingegeben. Ein Paar[1] wird als `(erster . zweiter)` eingegeben und sie müssen mit dem Apostroph eingeleitet werden, genauso wie Symbole:

```
\override TextScript #'extra-offset = #'(1 . 2)
```

Hierdurch wird das Paar (1, 2) mit der Eigenschaft `extra-offset` des TextScript-Objektes verknüpft. Diese Zahlen werden in Systembreiten gemessen, so dass der Befehl das Objekt eine Systembreite nach rechts verschiebt und zwei Breiten nach oben.

Die zwei Elemente eines Paares können von beliebigem Inhalt sein, etwa

```
#'(1 . 2)
#'(#t . #f)
#'("blah-blah" . 3.14159265)
```

Eine Liste wird eingegeben, indem die Elemente der Liste in Klammern geschrieben werden, mit einem Apostroph davor. Beispielsweise:

```
#'(1 2 3)
#'(1 2 "string" #f)
```

Die ganze Zeit wurde hier schon Listen benutzt. Eine Berechnung, wie `(+ 1 2)`, ist auch eine Liste (welche das Symbol `+` und die Nummern 1 und 2 enthält. Normalerweise werden Listen als Berechnungen interpretiert und der Scheme-Interpreter ersetzt die Liste mit dem Ergebnis der Berechnung. Um eine Liste an sich einzugeben, muss die Auswertung angehalten werden. Das geschieht, indem der Liste ein Apostroph vorangestellt wird. Für Berechnungen kann man also den Apostroph nicht verwenden.

Innerhalb einer zitierten Liste (also mit Apostroph) muss man keine Anführungszeichen mehr setzen. Im Folgenden ein Symbolpaar, eine Symbolliste und eine Liste von Listen:

[1] In der Scheme-Terminologie wird ein Paar `cons` genannt und seine zwei Elemente `car` und `cdr`.

```
#'(stem . head)
#'(staff clef key-signature)
#'((1) (2))
```

B.1 Optimierungen mit Scheme

Wir haben gesehen wie LilyPond-Eingabe massiv beeinflusst werden kann, indem Befehle wie etwa `\override TextScript #'extra-offset = ( 1 . -1)` benutzt werden. Aber es wurde gezeigt, dass Scheme noch mächtiger ist. Eine bessere Erklärung findet sich in derAnhang B [Scheme-Übung], Seite 181 und in Abschnitt “Schnittstellen für Programmierer” in *Notationsreferenz*.
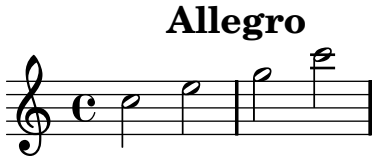
Scheme kann auch in einfachen `\override`-Befehlen benutzt werden:

TODO Find a simple example

Es kann auch benutzt werden, um Befehle zu erstellen:

```
tempoZeichen = #(define-music-function (parser location padding marktext)
                                       (number? string?)
#{
  \once \override Score . RehearsalMark #'padding = $padding
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $marktext }
#})
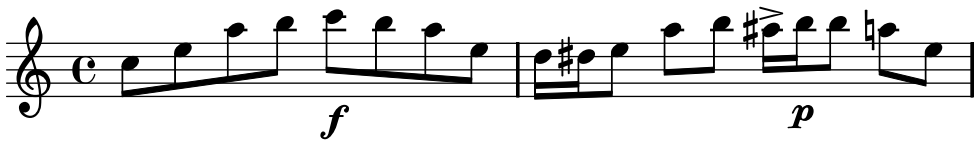
\relative c'' {
  c2 e
  \tempoZeichen #3.0 #"Allegro"
  g c
}
```



Sogar ganze Musikausdrücke können eingefügt werden:

```
Muster = #(define-music-function (parser location x y) (ly:music? ly:music?)
#{
  $x e8 a b $y b a e
#})

\relative c''{
  \Muster c8 c8\f
  \Muster {d16 dis} { ais16-> b\p }
}
```



Anhang C GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of ‚copyleft‘, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The ‚Document‘, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as ‚you‘.

A ‚Modified Version‘ of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A ‚Secondary Section‘ is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The ‚Invariant Sections‘ are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The ‚Cover Texts‘ are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A ‚Transparent‘ copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file

format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not ‚Transparent' is called ‚Opaque'.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The ‚Title Page' means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ‚Title Page' means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled ‚History‘, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled ‚History‘ in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the ‚History‘ section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled ‚Acknowledgments‘ or ‚Dedications‘, preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled ‚Endorsements‘. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as ‚Endorsements‘ or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to

the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled ‚Endorsements', provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled ‚History' in the various original documents, forming one section entitled ‚History'; likewise combine any sections entitled ‚Acknowledgments', and any sections entitled ‚Dedications'. You must delete all sections entitled ‚Endorsements.'

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an ‚aggregate', and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ‚or any later version‘ applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Anhang: Wie kann die Lizenz für eigene Dokumente verwendet werden

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
  Copyright (C)  year  your name.
  Permission is granted to copy, distribute and/or modify this document
  under the terms of the GNU Free Documentation License, Version 1.1
  or any later version published by the Free Software Foundation;
  with the Invariant Sections being list their titles, with the
  Front-Cover Texts being list, and with the Back-Cover Texts being list.
  A copy of the license is included in the section entitled ,GNU
Free Documentation License‘
.
```

If you have no Invariant Sections, write ,with no Invariant Sections‘ instead of saying which ones are invariant. If you have no Front-Cover Texts, write ,no Front-Cover Texts‘ instead of ,Front-Cover Texts being *list*‘; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Anhang D LilyPond-Index

!

! 23

#

..... 181
##f 181
##t 181
#'symbol 182

%

% 18
%{ ... %} 18

'

' 14

(

(...) 21

,

, ... 14

.

... . 18

<

< 23, 30
< ... > 30
<< 27, 30
<< ... >> 27
<< ... \\ ... >> 30
<< \\ >> 49

>

> 23, 30
>> 27, 30

[

[..... 24
[...] 24

]

] 24

^

^ 23

_

_ 23

\

\! 23
\(... \) 21
\< 23
\> 23
\\ 30, 49
\acciaccatura 26
\addlyrics 31
\addlyrics, Beispiel 96
\addlyrics-Beispiel 91
\appoggiatura 26
\autoBeamOff 24, 58
\autoBeamOn 24
\book 41, 43, 62
\clef 17
\consists 72
\context 71
\f 23
\ff 23
\grace 26
\header 38, 43
\key 20
\layout 43, 74
\lyricmode 58
\lyricsto 57
\major 20
\markup 24
\mf 23
\midi 43
\minor 20
\mp 23
\new 28, 65
\new ChoirStaff 58
\new Lyrics 57
\new Staff 28
\new Voice 54
\once 88, 93
\oneVoice 54
\override 87
\overrideProperty 89
\p 23
\partial 25
\pp 23
\relative 14
\remove 72
\revert 88, 94
\score 41, 44
\set 68
\set, Benutzungsbeispiel 110
\shiftOff 57
\shiftOn 57
\shiftOnn 57
\shiftOnnn 57
\startTextSpan 111
\stopTextSpan 111
\textLengthOff 113

\textLengthOn ... 113
\time ... 17
\times ... 25
\tweak ... 89
\tweak-Beispiel ... 89, 90
\unset ... 68
\version ... 38
\voiceFour ... 54
\voiceFourStyle ... 51
\voiceNeutralStyle ... 51
\voiceOne ... 54
\voiceOneStyle ... 51
\voiceThree ... 54
\voiceThreeStyle ... 51
\voiceTwo ... 54
\voiceTwoStyle ... 51
\with ... 71
\with-Beispiel ... 102, 103, 104, 105

~

~ ... 21

A

absolute Werte für Tonhöhen ... 38
absoluter Modus ... 38
Abstände ... 10
Abstände, normal ... 3
acciaccatura ... 26
Accidental, Beispiel zur Veränderung ... 120
AccidentalPlacement, Beispiel zur Veränderung ... 120
addlyrics ... 31
Akkolade ... 29
Akkord ... 30
Akkorde ... 30
Akkorde versus Stimmen ... 49
Aktualisieren von Dateien ... 38
Akzente ... 23
Akzidentien ... 20
alignAboveContext-Eigenschaft, Beispiel ... 102, 103, 104, 105
Allgemeine Eingabe und Ausgabe ... 10
Alt ... 17
Ambitus-Engraver ... 73
Andere rhythmische Aufteilungen ... 26
Anhänge ... 9
Ansicht von Noten ... 12
Anwendungsbenutzung ... 9, 10
Anzeigen der Noten ... 12
Apostroph ... 14
Artikulation ... 23
Artikulationszeichen ... 22, 23
Artikulationszeichen und Verzierungen ... 24
Auflösung von Zusammenstößen ... 128
Auflösungszeichen ... 20
Auftakt ... 25
Auftakte ... 26
Ausdruck, musikalischer ... 26
Ausdrücke ... 18
Ausdrücke, parallel ... 27
Ausdrücke, Verschachteln von ... 55
Ausrichten von Gesangstext ... 32
Ausrichten von Objekten an der Grundlinie ... 120
Ausrichtung von Objekten ... 117
autoBeamOff ... 24, 58
autoBeamOn ... 24
automatische Balken ... 24
Automatische Balken ... 25
Automatische Versetzungszeichen ... 21
automatischer Notensatz ... 3

B

B ... 20
Balance ... 2
Balken ... 16
Balken und Text ... 58
Balken, automatisch ... 24
Balken, Beispiel zur Veränderung ... 123
Balken, manuell ... 24
Balken, manuelle Kontrolle ... 122
Bass ... 17
Befehlsübersicht ... 10
Beispiel, einfach ... 8
Beispiel, erstes ... 12
Beispiele, klickbar ... 19
Benennungskonventionen für Eigenschaften ... 87
Benennungskonventionen für Objekte ... 87
Bennenung von Kontexten ... 66
Benutzung, lilypond-Programm ... 10
Bezeichner ... 36, 43, 141
Bezeichner versus Eigenschaften ... 182
Binde- versus Legatobogen ... 22
Bindebögen ... 21
Bindebogen ... 22
Bindebögen ... 23
Bindebogen, Beispiel zur Veränderung ... 128
Bindestrich ... 32
Block-Kommentare ... 18
Bögen, Legato ... 21
Bögen, manuelle Kontrolle ... 122
book ... 41, 43, 62
book, Benutzung von ... 62
book-Abschnitte, implizit ... 43
bound-details-Eigenschaft, Beispiel ... 111, 112
break-visibility-Eigenschaft ... 99
break-visibility-Eigenschaft, Beispiel ... 99

C

center ... 107
ChoirStaff ... 29, 58
Choral mit mehreren Strophen ... 59
Choralnotation ... 59
ChordNames ... 28
Chorpartitur ... 29
Chorpartitur, Aufbau ... 58
Chorpartitur, Vorlage ... 78
clef ... 17
Clef, Beispiel zur Veränderung ... 101
color-Eigenschaft ... 100
color-Eigenschaft, Beispiel ... 88, 90, 100, 101
consists ... 72
context ... 71
context, Voice ... 49
Contexts ... 6, 69, 72
convert-ly ... 38
Crescendo ... 23

D

Darstellung & Inhalt ... 20
Dateien aktualisieren ... 38
Dateien konstruieren, Hinweise ... 19
`Dateien mit convert-ly aktualisieren` ... 38, 146
Dateistruktur ... 41
Dauern ... 16
`Decrescendo` ... 23
`Der tweak-Befehl` ... 90
Dichte ... 2
Dicke ... 105
Dicke-Eigenschaft, Beispiel ... 92
`Die Dateistruktur` ... 41, 43, 44
`Die Standardeinstellungen von Umgebungen ändern` ... 72, 75
direction-Eigenschaft, Beispiel ... 90, 108
`Doppel-B` ... 20
`Doppelkreuz` ... 20
`down` ... 107
`Dur` ... 20
Durchsichtig machen (transparent) ... 99
DynamicLineSpanner, Beispiel zur Veränderung ... 120
DynamicText, Beispiel zur Veränderung ... 115
`Dynamik` ... 23, 24
DynamikText, Beispiel zur Veränderung ... 120
Dynamikzeichen: Positionierung verändern ... 114

E

Ebenen ... 49
Editoren ... 12
eigene Vorlagen erstellen ... 82
Eigenschaften in Kontexten ... 68
Eigenschaften versus Bezeichner ... 182
Eigenschaften von Grobs ... 91
Eigenschaften von Interfaces ... 95
Eigenschaften von Kontexten ... 68
Eigenschaften von Kontexten, mit \context setzen ... 71
Eigenschaften von Layout-Objekten ... 91
Eigenschaften von Objekten ... 86
Eigenschaften von Schnittstellen ... 95
Eigenschaften, Benennungskonventionen ... 87
Eigenschaftsarten ... 96
`Eignschaften verändern` ... 119
eine Stimme wiederherstellen ... 54
einfache Notation ... 14
einfaches Beispiel ... 8
Einfügen von Text ... 24
Eingabeformat ... 41
`eingestrichenes C` ... 14
Engraver ... 5, 67
Engraver, Entfernen von ... 72
Engraver, Hinzufügen von ... 72
`Engravers and Performers` ... 68
Entfernen von Engravern ... 72
Entfernen von Objekten ... 131
Entfernungen ... 105
Erstellen von eigenen Vorlagen ... 82
Erstellen von Kontexten ... 65
erstes Beispiel ... 12
`es` ... 20
`eses` ... 20
extra-offset-Eigenschaft ... 118, 121
extra-offset-Eigenschaft, Beispiel ... 121
`extra-spacing-width` ... 115
extra-spacing-width-Eigenschaft ... 117
extra-spacing-width-Eigenschaft, Beispiel ... 115, 120

F

Farb-Eigenschaft ... 100
Farb-Eigenschaft, Beispiel ... 88, 89, 90
Farb-Eigenschaft, in Scheme-Prozedur gesetzt ... 137
Farbeigenschaft, Beispiel ... 101
Farben, RGB ... 101
Farben, X11 ... 100
FDL, GNU Free Documentation License ... 184
Fermate, Benutzung in MIDI ... 132
fingerOrientations-Eigenschaft, Beispiel ... 110
`Fingersatz` ... 23
Fingersatz, Akkorde ... 108
Fingersatz, Beispiel zur Veränderung ... 108, 121
Fingersatz, Positionierung ... 108
Fingersatz-Beispiel ... 109, 110
`Fingersatzanweisungen` ... 24
font-series-Eigenschaft, Beispiel ... 134
font-shape-Eigenschaft, Beispiel ... 96, 134
font-size-Eigenschaft, Beispiel ... 89
fontSize (Schriftgröße), Standardeinstellung ... 71
force-hshift-Eigenschaft ... 117, 123
force-hshift-Eigenschaft, Beispiel ... 123, 130
Formatierung einer Partitur ... 4
Formatierungsregeln ... 4
Fremdsprache ... 9
`Fülllinie` ... 32
Füllung (padding) ... 116

G

`ganze Note` ... 16
Ganze Noten ... 16
Ganztaktpausen, Beispiel zur Veränderung ... 121
Gesangstext auf mehreren Systemen ... 36
Gesangstext und Balken ... 58
Gesangstext, Ausrichten ... 32
Gesangstext, Beispiel zur Veränderung ... 134
Gesangstext, schreiben ... 31
Gesangstextmodus, Kontext angeben ... 96
`Gesangtext` ... 31
Gesangtext, Verbindung mit Noten ... 57
geschweifte Klammern ... 18
`Gleichzeitig erscheinende Noten` ... 31
gleichzeitige Noten ... 49
gleichzeitige Noten und relativer Modus ... 27
Glossar ... 9
`grace` ... 26
`GrandStaff` ... 29
graphische Objekte (Grob) ... 86
Gravur, Notensatz ... 2
Grob, Größenveränderung ... 115
Grobs ... 86
Grobs, Eigenschaften von ... 91
Groß- und Kleinschreibung ... 12, 18
Großbuchstaben ... 12, 18
Größe, verändern ... 105
Größen von Objekten verändern ... 102

Größenveränderung von grobs ... 115
GUILE ... 181

H

`halbe Note` ... 16
Halbe Noten ... 16
Hals nach oben ... 53
Hals nach unten ... 53
Hals, Beispiel zur Veränderung ... 130, 132
Hälse, Länge verändern ... 105
Handbuch zum Lernen ... 9
Handbuch, Lesen ... 19
`header` ... 38, 43
Hinweise zur Konstruktion von Dateien ... 19
Hinzufügen von Engravern ... 72
Hinzufügen von Text ... 24
hoch-Eigenschaft ... 107
Hymnus-Notation ... 59

I

Idiom ... 9
implizite book-Umgebung ... 43
Implizite Kontexte ... 42
`Index der LilyPond-Befehle` ... 10
Inhalt & Darstellung ... 20
`Installieren` ... 10
Integration von LilyPond in andere Programme ... 10
Interface-Eigenschaften ... 95
Interfaces ... 86
`Intervalle` ... 14
IR (Referenz der Interna), Benutzung ... 91
`is` ... 20
`isis` ... 20

J

Jargon ... 9

K

`key` ... 20
`Klammer` ... 29
Klammer, Triole ... 90
Klammer-Typen ... 47
Klammern, geschachtelt ... 47
Klammern, geschweift ... 18
Klaviersystem ... 29
Kleinbuchstaben ... 12, 18
klickbare Beispiele ... 19
Kollision von Objekten im System ... 121
Kollisionen von Noten ... 57
Komma ... 14
Kommentare ... 18
Kompilieren ... 12
Komplizierte Rhythmen, Schachtelung von ... 90
Konstruieren von Dateien, Hinweise ... 19
Kontext ... 28
Kontext im Gesangstextmodus angeben ... 96
Kontext, Finden und identifizieren ... 93
Kontext, Stimme ... 49
Kontext-Eigenschaften, Verändern ... 68
Kontexte ... 6
Kontexte erklärt ... 64
Kontexte, Benennung ... 66
Kontexte, Erstellen ... 65
Kontexte, implizit ... 42
Kontexteigenschaft, setzen mit \with ... 71
Kontexteigenschaften, mit \context setzen ... 71
Kontrolle über Triolen, Bögen und Balken manuell ... 122
Kopf ... 38
Kopfzeile ... 43
Korrigieren von überschneidender Notation ... 119
`Korrigierte Musik überspringen` ... 140
`Kreuz` ... 20
kursiv, Beispiel ... 96

L

Länge ... 105
Lautstärke ... 23
`layout` ... 43
Layout ... 43
Layout-Objekt ... 86
Layout-Objekte, Eigenschaften von ... 91
Layout-Umgebung, Platzierung ... 43
left-padding-Eigenschaft ... 116
left-padding-Eigenschaft (Verschiebung nach links) ... 119
Legatobögen ... 21
`Legatobogen` ... 22
`Legatobögen` ... 23
Legatobogen, Beispiel für Veränderung ... 92
Legatobogen, Beispiel zur Veränderung ... 93, 94
Legatobögen, manuelle Kontrolle ... 122
Legatobögen, Phrasierung ... 21
Lesen des Handbuchs ... 19
Lieder ... 31
Liedtext ... 31
LilyPond Snippet Repository ... 10
`LilyPond starten` ... 10
LilyPond unter MacOS X ... 12
`LilyPond-book` ... 10, 178
LilyPond-Schnipsel-Depot ... 10
LISP ... 181
`Liste der Farben` ... 100
`Literatur` ... 10
LSR ... 10
`lyricmode` ... 58
`Lyrics` ... 28, 57
Lyrics context, erstellen ... 57
`lyricsto` ... 57
LyricText, Beispiel zur Veränderung ... 96, 134

M

MacOS X, LilyPond starten ... 12
`magstep` ... 105
magstep-Funktion, Beispiel ... 105
`major` ... 20
Makro ... 36
manuelle Balken ... 24
`Manuelle Balken` ... 25
manuelle Kontrolle über Triolen, Bögen, Balken ... 122
`markup` ... 24
Matrize ... 5

Matrizen-Eigenschaft ... 98
Mehrere Partituren in einem Buch ... 43
mehrere Silben in Gesangstext ... 32
mehrere Stimmen ... 30, 49
Mehrere Stimmen ... 53, 57
mehrere Strophen ... 59
mehrere Systeme ... 27, 28
Mehrere Systeme und Gesangstext ... 36
Mehrstimmigkeit ... 6, 27, 30
Melisma ... 32
Metronom-Bezeichnung, Beispiel zur Veränderung ... 133
Metronom-Bezeichnungsposition verändern ... 112
MetronomMark, Beispiel zur Veränderung ... 119
midi ... 43
MIDI-Dateien erstellen ... 43
MIDI: Fermate erstellen ... 132
minor ... 20
Moll ... 20
MultiMeasureRest, Beispiel zur Veränderung ... 121
Musikalische Notation ... 9, 40
Musikalischer Ausdruck ... 26
Musikausdruck, zusammengesetzter ... 44
Musikstück ... 44
Musiksymbole ... 2

N

N-tolen ... 25
N-tolen, geschachtelt ... 90
N-tolenklammer ... 90
Neue Kontexte ... 65
Neue Version ... 38
neutral ... 107
neutral-Eigenschaft ... 107
new ... 28, 65
new Staff ... 28
normale Abstände ... 3
normale Rhythmen ... 3
Notation von Gesang ... 35, 36, 64
Notation, einfach ... 14
Notationskontext ... 28
Notationsreferenz ... 9
Notationsübersicht ... 10
NoteHead, Beispiel für override ... 88, 89
NoteHead, Beispiel zur Veränderung ... 101
Noten anzeigen ... 12
Noten gleichzeitig ... 49
Noten verstecken ... 56
Noten zwischen Stimmen überbinden ... 131
Noten, durch Text gespreizt ... 113
Noten, unsichtbar ... 30
Notenbezeichnungen ... 38
Notenbezeichnungen in anderen Sprachen ... 14, 20, 21
Notendauer in Akkorden ... 30
Notendauern ... 16
Notengravur ... 2
Notenhals, Beispiel zur Veränderung ... 101, 107, 130, 132
Notenhals, Richtung ... 53
Notenhalslänge, verändern ... 105
Notenhalsrichtung ... 53
Notenhalsrichtung in Stimmen ... 53
Notenkollision ... 57
Notenkolumne ... 57
Notenkolumne, Beispiel zur Veränderung ... 123, 130
Notenkopf, Beispiel für Veränderung ... 88, 89
Notenkopf, Beispiel zur Veränderung ... 101, 137
Notenlinien, Länge verändern ... 105
Notenlinienabstände verändern ... 105
Notensatz ... 2, 5
Notensatz, automatisch ... 3
Notensatz, Mehrstimmigkeit ... 6
Notenschlüssel ... 18
Notensystem für Chor ... 29
Notensystem für Klavier ... 29
Notensystem-Position-Eigenschaft ... 121
Notensysteme, mehrere ... 27
Notenwert ... 25
Notenzeile, Positionierung ... 47
Notenzeilen, temporäre ... 46
notieren von Pausen ... 17

O

Objekt, Layout- ... 86
Objekte ... 86
Objekte an der Grundlinie ausrichten ... 120
Objekte außerhalb des Notensystems ... 106
Objekte entfernen ... 131
Objekte innerhalb des Notensystems ... 106
Objekte unsichtbar machen ... 131
Objekte verstecken ... 131
Objekte weißmalen ... 100
Objekte, Benennungskonventionen ... 87
Objekte, Positionierung ... 121
Objekte, verschieben von Zusammestößen ... 116
Objekteigenschaften ... 86
Objektgrößen, verändern ... 102
Oktave ... 14
Oktavenüberprüfung ... 139
Oktavierungsklammer ... 111
once ... 88, 93
once override ... 93
oneVoice ... 54
Optimierung mit Variablen ... 134
Optischer Ausgleich ... 2
ossia ... 46
Ossia ... 46
Ossia ... 102
Ossia-Systeme ... 47
Ottava-Klammer ... 111
outside-staff-Objekte ... 106
outside-staff-priority-Eigenschaft, Beispiel ... 113
override ... 87
Override nur einmal ... 93
override-Befehl ... 87
Override-Beispiel ... 91
overrideProperty ... 89
overrideProperty-Befehl ... 89

P

padding (Füllung) ... 116
padding (Verschiebungs-Eigenschaft) ... 119
padding (Verschiebuns-Eigenschaft), Beispiel ... 119
Padding-Eigenschaft ... 116

parallele Ausdrücke ... 27
`partial` ... 25
Partitur ... 29, 44
Partitur für Chor ... 29
Partitur, Formatierung ... 4
Partituren, mehrfache ... 43
`Partiturlayout` ... 43
Pausen ... 17
`Pausen eingeben` ... 18
PDF-Datei ... 12
`Phrasierung` ... 22
Phrasierungsbögen ... 21
`Phrasierungsbögen` ... 23
Phrasierungsbogen, Beispiel zur Veränderung ... 122
Phrasierungsbögen, manuelle Kontrolle ... 122
`PianoStaff` ... 29
Plattendruck, Noten ... 2
Platzhalternoten ... 30, 56
Platzierung von layout-Umgebung ... 43
Plugin ... 5
Polyphonie ... 6
`Polyphonie` ... 27
Polyphonie ... 49
Polyphonie und relativer Notationsmodus ... 52
Position der Taktzahl, verändern ... 112
Positionierung einer Notenzeile ... 47
Positionierung von Objekten ... 121
Positionierung, Beispiel ... 122, 123
positions-Eigenschaft ... 118, 122
positions-Eigenschaft, Beispiel ... 122, 123
Property (Layout-Objekte, Grobs) ... 91
`punktierte Note` ... 16
punktierte Noten ... 16

Q

Quelltext, übersetzen ... 12

R

Referenz der Interna ... 9, 11
Referenz der Interna, Benutzung ... 91
Referenz zum Notensatz ... 9
Refrain ... 60
Regeln zur Benennung von Objekten/Eigenschaften ... 87
Regerenz der Interna ... 91
rekursive Strukturen ... 6
`relative` ... 14
relative Noten und simulatane Musik ... 27
relativer Modus ... 14
relativer Modus und Versetzungszeichen ... 14
relativer Notationsmodus und Polyphonie ... 52
`remove` ... 72
`revert` ... 88, 94
Revert ... 94
revert-Befehl ... 88
`rgb-color` ... 101
RGB-Farben ... 101
`Rhythmen eingeben` ... 18
Rhythmen, normal ... 3
Rhythmische Aufteilungen ... 25
Rhythmus ... 16
Richtung des Notenhalses ... 53
Richtungs-Eigenschaft, Beispiel ... 90
Richtungseigenschaft, Beispiel ... 107
right-padding-Eigenschaft ... 116
right-padding-Eigenschaft (Verschiebung nach rechts) ... 119
right-padding-Eigenschaft, Beispiel ... 120
Rückgängig machen ... 94
runter-Eigenschaft ... 107

S

SATB-Partitur ... 59
SATB-Vorlage ... 78
Schachtelung von Klammern ... 47
Scheme ... 181
Scheme, in einer LilyPond-Datei ... 181
Scheme-Programmiersprache ... 4
Schlüssel ... 17
Schlüssel, Beispiel zur Veränderung ... 101, 102, 103, 104, 105
Schnipsel ... 10
Schnipselliste ... 9
Schnittstellen ... 86
`Schnittstellen für Programmierer` ... 10, 183
Schnittstellen, Eigenschaften von ... 95
Schreiben von eigenen Vorlagen ... 82
schreiben von Pausen ... 17
Schriftart ... 2
Schriftart-Eigenschaft, Beispiel ... 105
Schriftgröße, Beispiel ... 89
Schriftgröße, Standardeinstellung ... 71
`Schwierige Korrekturen` ... 89
`score` ... 41, 44
`Score` ... 28
Script, Beispiel zur Veränderung ... 119
Selbstpositionierung von Objekten ... 117
self-alignment-X-Eigenschaft ... 117, 121
`set` ... 68
`Setup` ... 10
shift-Befehle ... 57
`shiftOff` ... 57
`shiftOn` ... 57
`shiftOnn` ... 57
`shiftOnnn` ... 57
Sopran ... 17
Spanners ... 86
`Spezielle Notation` ... 9
Sprachen ... 9
Staccato ... 23
`Staff` ... 28
staff-padding-Eigenschaft ... 116, 120
staff-padding-Eigenschaft, Beispiel ... 120
staff-position-Eigenschaft ... 117, 121
staff-position-Eigenschaft, Beispiel ... 121, 128
staff-space-Eigenschaft verändern ... 105
StaffSymbol, Beispiel zur Veränderung ... 101
`Standardeinstellungen verändern` ... 10
Standardeinstellungen, Wiederherstellen ... 94
Starten von LilyPond ... 12
`startTextSpan` ... 111
Stem, Beispiel zur Veränderung ... 101
Stempel ... 5
Stempel (Engraver) ... 67
Stencil-Eigenschaft ... 98

stencil-Eigenschaft, Beispiel 98, 99, 102, 103, 104, 105, 120
stencil-Eigenschaft, Benutzung 132
Stimmen und Notenhalsrichtung 53
Stimmen versus Akkorde 49
Stimmen, mehrere 49
Stimmen, mehrere in einem System 30
Stimmen, mehrere zu einer zusammenführen 54
Stimmen, temporär 55
Stimmen, Verschachteln von 55
Stimmenkontext 49
Stimmenkontexte, erstellen von 54
Stimmwechsel zwischen Systemen, manuell 29
`stopTextSpan` 111
Strecker 86
StringNumber, Beispiel zur Veränderung 121
Strophe und Refrain 60
Strophen, mehrere 59
`Struktur einer Partitur` 46
Struktur, Datei 41
Syntax 6
System für Klavier 29
`Systeme anzeigen lassen` 30
Systeme, mehrere 28
Systemwechsel, manuell 29

T

`Takt- und Taktzahlüberprüfung` 139
`Taktangabe` 17, 18
Taktart 17
Taktart, Beispiel zur Veränderung 99, 101, 102, 103, 104, 105
Taktlinie, Beispiel zur Veränderung .. 98, 99, 100, 101
Taktlinien, Beispiel zur Veränderung 101
Taktzahlposition verändern 112
`Tasteninstrumente und andere Instrumente mit mehreren Systemen` 30
Template, eigene schreiben 82
Template, Verändern von 75
Templates 19
temporäre Notenzeilen 46
temporäre Stimmen 55
Tenor 17
Terminologie 9
Text 31
`Text eingeben` 24
`Text mit Verbindungslinien` 112
Text und Balken 58
Text, einfügen 24
text-Eigenschaft, Beispiel 90
Text-Eigenschaft, Beispiel 120
Text-Spanner 111
Text-Strecker 111
Textbeschriftung 24
Textbeschriftung, Beispiel zur Veränderung 113
Textbeschriftung, Vermeidung von Zusammenstößen 113
Textbeschriftungsbeispiel 106
Texteditoren 12
`textLengthOff` 113
`textLengthOn` 113
TextScript, Beispiel zur Veränderung 113
TextSpanner, Beispiel zur Veränderung 111, 112
Thesaurus 9
thickness-Eigenschaft, Beispiel 92, 93, 94
`time` 17
`times` 25
TimeSignature, Beispiel zur Veränderung 101
Titel 38
`Titel erstellen` 38
Titelei 38
Tonart, Einstellung von 20
`Tonartbezeichnung` 20, 21
`Tondauer` 16
`Tonhöhe` 14, 20
Tonhöhen 14
`Tonhöhen setzen` 18
Tonhöhen, absolute Werte 38
`Tonhöhenbezeichnungen` 21
`Tonleiter` 14
`Tonumfang` 73
`Top` 9, 11, 91
transparent-Eigenschaft 99
transparent-Eigenschaft, Beispiel ... 90, 99, 130, 132, 133
transparent-Eigenschaft, Benutzung 131
Transparente Objekte 131
`Transposition` 20
`Triole` 25
Triolen 25
Triolen, geschachtelt 90
Triolen-Nummer-Funktion, Beispiel 90
Triolenklammer 90
Triolennummer, Beispiel zur Veränderung 90
Triollen-Klammer, manuelle Kontrolle 122
`Tunable context properties` 69, 72
tuplet-number-Funktion, Beispiel 90
`TupletBracket` 90
TupletNumber, Beispiel zur Veränderung 90
`tweak` 89
tweak-Befehl 89
Typographie 2, 3, 5

U

über dem System anordnen, Beispiel .. 102, 103, 104, 105
Überbinden von Noten zwischen Stimmen 131
Überschneidende Notation korrigieren 119
Übersetzen von Quelltext 12
Übersetzungen 9
Übungszeichenposition verändern 112
`Umgebungen erstellen` 67
`Umgebungs-Plugins verändern` 75
Unix, LilyPond starten 12
`unset` 68
Unsichtbar machen (break-visibility) 99
unsichtbare Noten 30, 56
Unsichtbare Objekte 131
Unterstrich 32
`Unterstützung von Texteditoren` 12
`up` 107
Update 38

V

Variable 36, 141

Variable, erlaubte Zeichen ... 36
Variablen ... 43
Variablen benutzen ... 36
Variablen zuweisen ... 36
Variablen, Benutzung zur Optimierung ... 134
Verändern der Metronom-Bezeichnungsposition ... 112
Verändern der Positionierung von Dynamikzeichen ... 114
Verändern der Taktzahlposition ... 112
Verändern der Übungszeichenposition ... 112
Verändern von Kontext-Eigenschaften ... 68
Veränderung von Eigenschaften in Kontexten ... 68
Veränderung von Objektgrößen ... 102
Veränderung von Vorlagen ... 75
Vermeiden von Zusammenstößen ... 116
Verschachteln von gleichzeitigen Ausdrücken ... 55
Verschachteln von musikalischen Ausdrücken ... 55
Verschachteln von Stimmen ... 55
Verschieben (padding) ... 116
Verschieben von Noten ... 57
Verschieben von überschneidenden Objekten ... 116
Verschieben von Zusammenstößen ... 116
Verschiebung nach rechts (rigth-padding), Beispiel ... 120
Verschiebung nach rechts oder links ... 119
Verschiebungs-Eigenschaft, Beispiel ... 119
`Versetzungszeichen` ... 14, 20, 21
Versetzungszeichen und relativer Modus ... 14
Versetzungszeichen, Beispiel zur Veränderung ... 120
`version` ... 38
Versionsnummern ... 38
Verstecken von Objekten ... 131
Vertikale Position ... 57
Vertikale Verschiebung erzwingen ... 123
vertikale Verschiebung, Beispiel ... 123
`Verzierungen` ... 26
`Viertelnote` ... 16
Viertelnoten ... 16
`Voice` ... 28
Voice context ... 49
Voice context, erstellen von ... 54
`voiceFour` ... 54
`voiceOne` ... 54
`voiceThree` ... 54
`voiceTwo` ... 54
Vokalpartitur, Aufbau ... 58
`Von anderen Formaten konvertieren` ... 10
`Vorhalt` ... 26
Vorlage, Chorpartitur ... 78
Vorlage, SATB ... 78
Vorlage, Verändern von ... 75
Vorlagen ... 19
Vorlagen, eigene schreiben ... 82
`Vorschlag` ... 26
Vorzeichen ... 20
Vorzeichen und Versetzungszeichen ... 20

W

`Was sind Umgebungen?` ... 65
Wechsel zwischen Systemen, manuell ... 29
Wie soll das Handbuch gelesen werden ... 19
Wiederherstellen von Standardeinstellungen ... 94
Windows, LilyPond starten ... 12
`with` ... 71
within-staff-Objekte ... 106
Worte mit mehreren Silben ... 32

X

`x11-color` ... 100
X11-Farben ... 100
x11-Farben, Beispiel ... 101
x11-Farben, Beispiel zur Benutzung ... 137

Z

Zeichen, in Variablen erlaubt ... 36
Zeilenkommentare ... 18
zentriert-Eigenschaft ... 107
Zitieren in Scheme ... 182
zusammengesetzter musikalischer Ausdruck ... 26
zusammengesetzter Musikausdruck ... 44
Zusammenstöße vermeiden ... 116
Zusammenstöße vermeiden mit Textbeschriftung ... 113
Zusammenstöße von Noten ... 57
Zusammenstöße von Objekten im System ... 121
Zusätzlicher Abstand, Positionierung ... 121
Zuweisen von Variablen ... 36