

GNU LilyPond

The music typesetter

Copyright © 1999–2002 by the authors

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”.

Table of Contents

Preface	1
1 Introduction	2
1.1 Batch processing	2
1.2 Music engraving	2
1.3 Music representation	4
1.4 About this manual	5
2 Tutorial	6
2.1 First steps	6
2.2 Running LilyPond	8
Windows users	9
2.3 More basics	10
2.4 Printing lyrics	17
2.5 A melody with chords	18
2.6 More stanzas	22
2.7 More movements	23
2.8 A piano excerpt	26
2.9 Fine tuning a piece	30
2.10 An orchestral score	35
2.10.1 The full score	36
2.10.2 Extracting an individual part	40
2.11 Integrating text and music	42
2.12 End of tutorial	44
3 Reference Manual	45
3.1 Note entry	45
3.1.1 Notes	45
3.1.2 Pitches	45
3.1.3 Chromatic alterations	46
3.1.4 Rests	46
3.1.5 Skips	47
3.1.6 Durations	47
3.1.7 Ties	48
3.1.8 Automatic note splitting	49
3.1.9 Tuplets	50
3.1.10 Easy Notation note heads	50
3.2 Easier music entry	51
3.2.1 Graphical interfaces	51
3.2.2 Relative octaves	51
3.2.3 Bar check	52
3.2.4 Point and click	53

3.2.5	Skipping corrected music	54
3.3	Staff notation	54
3.3.1	Staff symbol	55
3.3.2	Key signature	55
3.3.3	Clef	55
3.3.4	Time signature	56
3.3.5	Partial	56
3.3.6	Unmetered music	57
3.3.7	Bar lines	57
3.4	Polyphony	58
3.5	Beaming	60
3.5.1	Manual beams	60
3.5.2	Setting automatic beam behavior	61
3.6	Accidentals	63
3.6.1	Using the predefined accidental macros	63
3.6.2	Defining your own accidental typesettings	65
3.7	Expressive marks	66
3.7.1	Slurs	66
3.7.2	Phrasing slurs	67
3.7.3	Breath marks	67
3.7.4	Tempo	67
3.7.5	Text spanners	68
3.8	Ornaments	68
3.8.1	Articulations	68
3.8.2	Text scripts	70
3.8.3	Grace notes	70
3.8.4	Glissando	73
3.8.5	Dynamics	73
3.9	Repeats	74
3.9.1	Repeat syntax	74
3.9.2	Repeats and MIDI	75
3.9.3	Manual repeat commands	76
3.9.4	Tremolo repeats	76
3.9.5	Tremolo subdivisions	77
3.9.6	Measure repeats	77
3.10	Rhythmic music	77
3.10.1	Percussion staves	78
3.10.1.1	Percussion staves with normal staves ..	80
3.10.1.2	Percussion midi output	80
3.11	Piano music	81
3.11.1	Automatic staff changes	81
3.11.2	Manual staff switches	82
3.11.3	Pedals	82
3.11.4	Arpeggio	83
3.11.5	Voice follower lines	84
3.12	Tablatures	85
3.12.1	Tablatures basic	85
3.12.2	Non-guitar tablatures	86

3.12.3	Tablature in addition to normal staff.....	87
3.13	Chords.....	87
3.13.1	Chords mode.....	88
3.13.2	Printing named chords.....	89
3.14	Writing parts.....	92
3.14.1	Rehearsal marks.....	92
3.14.2	Bar numbers.....	92
3.14.3	Instrument names.....	93
3.14.4	Transpose.....	94
3.14.5	Multi measure rests.....	94
3.14.6	Automatic part combining.....	94
3.14.7	Hara kiri staves.....	96
3.14.8	Sound output for transposing instruments.....	97
3.15	Ancient notation.....	97
3.15.1	Ancient note heads.....	97
3.15.2	Ancient clefs.....	97
3.15.3	Custodes.....	101
3.15.4	Ligatures.....	102
3.15.4.1	White mensural ligatures.....	103
3.15.5	Figured bass.....	104
3.16	Tuning output.....	105
3.16.1	Tuning groups of objects.....	105
3.16.2	Tuning per object.....	107
3.16.3	Font selection.....	108
3.16.4	Text markup.....	109
3.17	Global layout.....	111
3.17.1	Vertical spacing.....	112
3.17.2	Horizontal Spacing.....	112
3.17.3	Font size.....	114
3.17.4	Line breaking.....	114
3.17.5	Page layout.....	114
3.18	Sound.....	115
3.18.1	MIDI block.....	115
3.18.2	MIDI instrument names.....	116
4	Advanced Topics.....	117
4.1	Interpretation context.....	118
4.1.1	Creating contexts.....	118
4.1.2	Default contexts.....	119
4.1.3	Context properties.....	120
4.1.4	Engravers and performers.....	121
4.1.5	Changing context definitions.....	121
4.1.6	Defining new contexts.....	122
4.2	Syntactic details.....	123
4.2.1	Identifiers.....	123
4.2.2	Music expressions.....	124
4.2.3	Manipulating music expressions.....	124
4.2.4	Span requests.....	126

4.2.5	Assignments	126
4.2.6	Lexical modes	127
4.2.7	Ambiguities	127
4.3	Lexical details	127
4.3.1	Direct Scheme	127
4.3.2	Reals	128
4.3.3	Strings	128
4.4	Output details	128
5	Invoking LilyPond	130
5.1	Command line options	130
5.2	Environment variables	131
5.3	Reporting bugs	132
5.4	Website	132
5.5	Invoking ly2dvi	132
5.5.1	Titling layout	133
5.5.2	Additional parameters	134
5.5.3	Environment variables	135
6	Integrating text and music with lilypond-book	136
6.1	Integrating Texinfo and music	136
6.2	Integrating LaTeX and music	137
6.3	Integrating HTML and music	137
6.4	Music fragment options	138
6.5	Invoking lilypond-book	140
6.6	Bugs	141
7	Converting from other formats	142
7.1	Invoking convert-ly	142
7.2	Invoking midi2ly	142
7.3	Invoking etf2ly	144
7.4	Invoking abc2ly	144
7.5	Invoking pmx2ly	145
7.6	Invoking musedata2ly	145
7.7	Invoking mup2ly	146
8	Literature	147
	Index	149

Appendix A Refman appendix 156

A.1	Lyrics mode definition	156
A.2	American Chords	156
A.3	Jazz chords	157
A.4	MIDI instruments	159
A.5	The Feta font	160

Appendix B GNU Free Documentation License 163

B.0.1	ADDENDUM: How to use this License for your documents	169
-------	---	-----

Preface

It must have been during a rehearsal of the EJE (Eindhoven Youth Orchestra), somewhere in 1995 that Jan, one of the cranked violists told Han-Wen, one of the distorted French horn players, about the grand new project he was working on. It was an automated system for printing music (to be precise, it was MPP, a preprocessor for MusiXTeX). As it happened, Han-Wen accidentally wanted to print out some parts from a score, so he started looking at the software, and he quickly got hooked. It was decided that MPP was a dead end. After lots of philosophizing and heated e-mail exchanges Han-Wen started LilyPond in 1996. This time, Jan got sucked into Han-Wen's new project.

In some ways, developing a computer program is like learning to play an instrument. In the beginning, discovering how it works is fun, and the things you can't do are challenging. After the initial excitement, you have to practice and practice. Scales and studies can be dull, and if you aren't motivated by others—teachers, conductors or audience—it is very tempting to give up. You continue, and gradually playing becomes a part of your life. Some days it comes naturally, and it's wonderful, and on some days it just doesn't work, but you keep playing, day after day.

Like making music, working on LilyPond is can be dull work, and on some days it feels like plodding through a morass of bugs. Nevertheless, it has become a part of our life, and we keep doing it. Probably the most important motivation is that our program actually does something useful for people. When we browse around the net we find many people that use LilyPond, and use it to produce impressive pieces of sheet music. Seeing that still feels unreal, but in a very pleasant way.

Our users not only give us good vibes by using our program, many of them also help us by giving suggestions and sending bugreports. So first and foremost, we would like to thank all users that sent us bugreports, gave suggestions or contributed in any other way to LilyPond.

We would also like to thank the following people: Mats Bengtsson for the incountable number of questions he answered on the mailing list, and Rune Zedeler for his energy in finding and fixing bugs. Nicola Bernardini for inviting us to his workshop on music publishing, which was truly a masterclass, and Heinz Stolba and James Ingram for teaching us there.

Playing and printing music is more than nice analogy: programming together is a lot of fun, and helping people is deeply satisfying, but ultimately, working on LilyPond is a way to express our deep love for music. May it help you create lots of beautiful music!

Han-Wen and Jan

Utrecht/Eindhoven, The Netherlands, July 2002.

1 Introduction

LilyPond is a program to print sheet music. If you have used notation programs before, then the way to use this program might be surprising at first sight. To print music with lilypond, you have to enter musical codes in a file. Then you run LilyPond on the file, and the music is produced without any further intervention. For example, something like this:

```
\key c \minor r8 c16 b c8 g as c16 b c8 d | g,4
```

produces this



Encoding music using letters and digits may appear strange, intimidating or even clumsy at first. Nevertheless, when you take the effort to learn the codes and the program you will find that it is easier than it seems. Entering music can be done quickly, and you never have to remember how you made the program do something complicated: it's all in the input code, and you only have to read the file to see how it works. Moreover, when you use LilyPond, you are rewarded with very nicely looking output.

In this chapter, we will explain the reasoning behind this unusual design, and how this approach affects you as a user.

1.1 Batch processing

When we started developing LilyPond, we were still studying at the university. We were interested in music notation, not as publishers or musicians, but as students and scientists. We wanted to figure to what extent formatting sheet music could be automated. Back then GUIs were not as ubiquitous as they are today, and we were immersed in the UNIX operating system, where it is very common to use compilers to achieve computing tasks, so our computerized music engraving experiment took on the form of a compiler.

You can freely use, modify and redistribute LilyPond. This choice was also motivated by our academic background. In the scientific community it has always been a tradition to share knowledge, also if that knowledge was packaged as software. One of the most visible groups that stimulated this philosophy, was the Free Software Foundation, whose popular GNU project aimed to replace closed and proprietary computing solutions with free (as in “Libre”) variants. We jumped on that bandwagon, and released LilyPond as free software. That is the reason that you can get LilyPond at no cost and without any strings attached.

1.2 Music engraving

Making sheet music may seem trivial at first (“you print 5 lines, and then put in the notes at different heights”), *music engraving*, i.e. professional music typography, is in another ballpark. The term ‘music engraving’ derives from the traditional process of music printing. Only a few decades ago, sheet music was made by cutting and stamping the music into zinc or pewter plates, mirrored. The plate would be inked, and the depressions caused by the cutting and stamping would hold ink. A positive image was formed by pressing paper

to the plate. Stamping and cutting was completely done by hand. Making corrections was cumbersome, so engraving had to be done correctly in one go. As you can imagine this was a highly specialized skill, much more so than the traditional process of printing books.

The following fact illustrates that. In the traditional German craftsmanship six years of full-time training, more than any other craft, were required before a student could call himself a master of the art. After that many more years of practical experience were needed to become an established music engraver. Even today, with the use of high-speed computers and advanced software, music requires lots of manual fine tuning before it acceptable to be published.

When we wanted to write a computer program to do create music typography, we encountered the first problem: there were no sets of musical symbols available: either they were not available freely, or they didn't look well to our taste. Not let down, we decided to try font design ourselves. We created a font of musical symbols, relying on nice printouts of hand-engraved music. It was a good decision to design our own font. The experience helped develop a typographical taste, and it made us appreciate subtle design details. Without that experience, we would not have realized how ugly the fonts were that we admired at first.



The figure above shows a few notable glyphs. For example, the half-notehead is not elliptic but slightly diamond shaped. The vertical stem of a flat symbol should be slightly brushed, i.e. becoming wider at the top. Fine endings, such as the one on the bottom of the quarter rest, should not end in sharp points, but rather in rounded shapes. Taken together, the blackness of the font must be carefully tuned together with the thickness of lines, beams and slurs to give a strong yet balanced overall impression.

Producing a strong and balanced look is the real challenge of music engraving. It is a recurring theme with many variations. In spacing, the balance is in a distribution that reflects the character of the music. The spacing should not lead to unnatural clusters of black and big gaps with white space. The distances between notes should reflect the durations between notes, but adhering with mathematical precision to the duration will lead to a poor result. Shown here is an example of a motive, printed twice. It is printed using both exact, mathematical spacing, and with some corrections. Can you spot which is which?



The fragment that was printed uses only quarter notes: notes that are played in a constant rhythm. The spacing should reflect that. Unfortunately, the eye deceives us a little: the eye not only notices the distance between note heads, but also between consecutive stems. The notes of a up-stem/down-stem combination should be put farther apart, and the notes of a down-up combination should be put closer together, all depending on the combined vertical positions of the notes. The first two measures are printed with this correction, the last two measures without. The notes in the last two measures form downstem/upstems clumps of notes.

We hope that these examples show that music typography is a subtle business, and that it requires skill and knowledge to produce good engraving. It was our challenge to see if we could put such knowledge into a computer program.

1.3 Music representation

One of the big questions when making programs, is what kind of input the program should expect. Many music notation programs offer a graphical interface that shows notation, and allow you to enter the music by placing notes on a staff. Although this is a obvious way to design a program, from our point of view, it is cheating. After all, the core message of a piece of music notation simply is the music itself. If you start by offering notation to the user, you have already skipped one conversion, even if it is implicit. If we want to generate music notation from something else, then the obvious candidate for the source is the music itself.

On paper this theory sounds very good. In practice, it opens a can of worms. What really *is* music? Many philosophical treatises must have been written on the subject. Even if you are more practically inclined, you will notice that there exist an enormous number of ways to represent music in a computer, and they are much more incompatible than the formats for word processors and spreadsheets. Anyone who has tried to exchange data files from between different notation programs can attest to this.

This problem is caused by the two-dimensional nature of music: in polyphonic music, notes have time and pitch as their two coordinates, and they often are related in both directions. Computer files on the other hand are essentially one-dimensional: they are a long stream of characters. When you represent music in a file, then you have to flatten this two-dimensional information breaking either timing or pitch relations, and there is no universal agreement on how to do this.

Fortunately, we have a concrete application, so we don't run the risk of losing ourselves in philosophical arguments over the essence of music. We want to produce a printed score from a music representation, so this gives us a nice guide for designing a format: we need a format containing mainly musical elements, such as pitch and duration, but also enough information to print a score. Our users have to key in the music into the file directly, so the input format should have a friendly syntax. Finally, we as programmers and scientists want a clean formal definition. After all, producing music notation is a difficult problem, and in the scientific world, problems can only be solved if they are well-specified. Moreover, formally defined formats are easier to write programs for.

These ideas shaped our music representation: it is a compact format that can easily be typed by hand. It complex musical constructs from simple entities like notes and rests, in much the same way that one builds complex formulas from simple expressions such as numbers and mathematical operators. The strict separation between musical information and typesetting also gives a blueprint of the program: first it reads the music representation, then it interprets the music—reading it 'left-to-right', and translating the musical information to a layout specification. When the layout is computed, the resulting symbols are written to an output file.

1.4 About this manual

As you will notice in the coming pages the program makes good decisions in a lot of cases: what comes out of LilyPond generally looks good. The default layout of lilypond even is suitable for publication for some specific files. However, some aspects of the formatting are not yet very good. For us programmers, this gives inspiration for improving the program. However, most users are more interested in improving their printouts, and then they have to make manual adjustments to the output. Another aspect of our system of encoding through ASCII then shows: it can be complicated to fine tune the layout of a piece. There is no graphical user interface, where you can simply click and drag a symbol. On the other hand, if you have written the code for tuning one specific aspect of the layout, then you can simply store the file on disk, retrieve it when you need it: there is no need to remember how you did it, since it is all in the input file.

Lilypond also comes with a huge collection of snippets that show all kinds of tricks. This collection is much needed, because of the way LilyPond is structured. It is a large program, but almost all of the internal functionality is exported: that is, the variables that are internally used for formatting the sheet music are available directly to the user. These are variables to control thicknesses, distances, and other formatting options. There are a huge number of them, and it would be impossible to describe them all in a hand-written manual. There is no need to despair, there is an ‘automatic’ manual, that lists all of the variables that are available. It is directly generated from the definitions that LilyPond itself uses, so it is always up to date. If you are reading this from a screen: it is available from the web, and is included with most binary distributions. If you’re reading this from paper, then we advise you to use the digital version anyway: the hyperlinks make finding topics in the lilypond-internals manual much easier.

For those who really want to get their hands dirty: it is even possible to add your own functionality, by extending LilyPond in the built-in scripting language, a dialect of the powerful programming language Scheme. There is no real distinction between what a user can do and what a programmer is allowed to do.

In summary, this manual does not pretend to be exhaustive, but it is merely a guide that tries to explain the most important principles, and shows popular input idioms. The rest of the manual is structured as follows: it starts with a tutorial that explains how to use lilypond. In the tutorial, a number of fragments of increasing complexity are shown and explained. Then comes the reference manual, which gives more detailed information on all features. If you’re new to lilypond, then you should start reading the tutorial, and experiment for yourself. If you already have some experience, then you can simply use the manual as reference: there is an extensive index.¹

¹ If you are looking for something, and you can’t find it by using the index, that is considered a bug. In that case, please file a bug report

2 Tutorial

Operating lilypond is done through text files: To print a piece of music, you enter the music in a file. When you run lilypond (normally using the program `ly2dvi`) on that file, the program produces another file which contains sheet music that you can print or view.

This tutorial starts with a small introduction to the LilyPond music language. After this first contact, we will show you how to run LilyPond to produce printed output; you should then be able to create your first sheets of music. The tutorial continues with more and more complex examples.

2.1 First steps

We start off by showing how very simple music is entered in LilyPond: you get a note simply by typing its note name, from ‘a’ through ‘g’. So if you enter

```
c d e f g a b
```

then the result looks like this:



We will continue with this format: First we show a snippet of input, then the resulting output.

The length of a note is specified by adding a number, ‘1’ for a whole note, ‘2’ for a half note, and so on:

```
a1 a2 a4 a16 a32
```



If you don’t specify a duration, the previous one is used:

```
a4 a a2 a
```



A sharp (#) is made by adding ‘is’, a flat (b) by adding ‘es’. As you would expect, a double sharp or double flat is made by adding ‘isis’ or ‘eses’:

```
cis1 ees fisis aeses
```



Add a dot ‘.’ after the duration to get a dotted note:

```
a2. a4 a8. a16
```



The meter (or time signature) can be set with the `\time` command:

```
\time 3/4
\time 6/8
\time 4/4
```



The clef can be set using the `\clef` command:

```
\clef treble
\clef bass
\clef alto
\clef tenor
```



When you enter these commands in a file, you must to enclose them in `\notes {...}`. This lets LilyPond know that music (and not lyrics, for example) follows:

```
\notes {
  \time 3/4
  \clef bass
  c2 e4 g2.
  f4 e d c2.
}
```

Now the piece of music is almost ready to be printed. The final step is to combine the music with a printing command.

The printing command is the so-called `\paper` block. Later on you will see that the `\paper` block is used to customize printing specifics. The music and the `\paper` block are combined by enclosing them in `\score { ... }`. This is what a full LilyPond source file looks like:

```
\score {
  \notes {
    \time 3/4
    \clef bass
    c2 e4 g2.
    f4 e d c2.
  }
  \paper { }
}
```



2.2 Running LilyPond

In the last section we explained what kind of things you could enter in a lilypond file. In this section we explain how to run LilyPond and how to view or print the output. If you have not used LilyPond before, want to test your setup of LilyPond, or want to run an example file yourself, read this section. The instructions that follow are for running LilyPond on Unix-like systems. Some additional instructions for running LilyPond on Windows are given at the end of this section.

Begin by opening a terminal window and starting a text editor. For example, you could open an xterm and execute `joe`. In your text editor, enter the following input and save the file as `test.ly`:

```
\score {
  \notes { c'4 e' g' }
}
```

LilyPond is the program that computes the sheet music. All other things, such as adding titles, page breaking and other page layout, are done by a small wrapper program called `ly2dvi`. `ly2dvi` calls lilypond to render the music, and then adds the titling and page layout instructions. To process `test.ly` with `ly2dvi`, proceed as follows:

```
ly2dvi -P test.ly
```

You will see the following on your screen:

```
GNU LilyPond 1.6.0
Now processing: '/home/fred/ly/test.ly'
Parsing...
Interpreting music...[1]
... more interesting stuff ...
PS output to 'test.ps'...
DVI output to 'test.dvi'...
```

The results of the `ly2dvi` run are two files, `test.dvi` and `test.ps`. The PS file (`test.ps`) is the one you can print. You can view the PS file using the program `ghostview`. If a version of `ghostview` is installed on your system, one of these commands will produce a window with some music notation on your screen:

```
gv test.ps
ghostview test.ps
ggv test.ps
kghostview test.ps
```

If the music on your screen looks good, you can print it by clicking File/Print inside `ghostview`.

The DVI file (`test.dvi`) contains the same sheet music in a different format. DVI files are more easily processed by the computer, so viewing them usually is quicker. You can run `xdvi test.dvi` or `kdvi test.dvi` to view the DVI file. In `Xdvi`, the mouse buttons

activate magnifying glasses. Unfortunately, variable symbols (such as beams and slurs) are not displayed in the magnifying glasses.

If your DVI viewer does not have a "Print" button, you can print the file by executing `lpr test.ps`.

If your system does not support printing PostScript files, then you can install Ghostscript, a PostScript emulator. Refer to Ghostscript's website at <http://www.ghostscript.com>.

A final option is to use the popular PDF format. You can get a PDF file by running `ly2dvi --pdf test.ly`. With `--pdf` you will get DVI, PS and PDF files. Viewers for PS files also accept PDF files, but there are also many other applications for viewing PDF files.

If you are familiar with T_EX, be warned: do not use other DVI drivers like `dvilj`. The T_EX coming out of LilyPond uses embedded PostScript code and will not render correctly if you use anything other than `dvips`.

Windows users

Windows users can start the terminal by clicking on the LilyPond or Cygwin icon. You can use any text editor (such as NotePad, Emacs or Vim) to edit the LilyPond file. If you install the Cygwin's XFree86 X11 window system, `tetex-x11` and `ghostscript-x11` packages too, you can view the dvi output doing `xdvi test.dvi` as described above. If you have installed a PostScript/PDF viewer, such as GSView from <http://www.cs.wisc.edu/~ghost>, viewing the PS file can be done with:



```
gsview32 test.ps
```

You can also print from the command line by executing:

```
gsview32 /s test.ps
```

SUMMARY

To run LilyPond, input a text file, then run the command `ly2dvi` on that file. The resulting files are either DVI or PostScript, and can be viewed with `xdvi` (unix) and `ghostview` (unix and windows) respectively. The following table summarizes the constructs that were discussed in the previous two sections.

Syntax	Description	Example
1 2 8 16	durations	
. . .	augmentation dots	

`c d e f g a b`

scale

`\clef treble \clef bass`

clefs

`\time 3/4 \time 4/4`

time signature



2.3 More basics

We continue with the introduction of the remaining musical constructs. Normal rests are entered just like notes with the name “`r`”:

`r2 r4 r8 r16`

To raise a note by an octave, add a high quote `'` (apostrophe) to the note name, to lower a note one octave, add a “low quote” `,` (a comma). Middle C is `c'`:

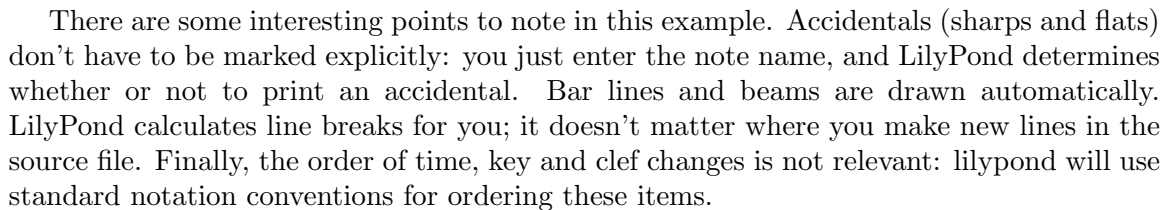
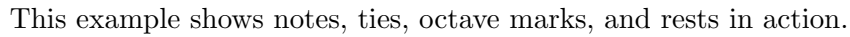
`c'4 c'' c''' \clef bass c c,`

A tie is created by entering a tilde “`~`” between the notes to be tied. A tie between two notes means that the second note must not be played separately; it just makes the first note sound longer:

`g'4 ~ g' a'2 ~ a'4`

The key signature is set with the command “`\key`”. One caution: you need to specify whether the key is `\major` or `\minor`.

```
\key d \major
g'1
\key c \minor
g'
```

The solution is to use “relative octave” mode. In practice, this is the most convenient way to copy existing music. To use relative mode, add `\relative` before the piece of music. You must also give a note from which relative starts, in this case `c''`. If you don’t use octavation quotes (ie don’t add `'` or `,` after a note), relative mode chooses the note that is closest to the previous one. Since most music has small intervals, you can write quite a lot in relative mode without using octavation quotes. For example: `c f` goes up; `c g` goes down:

You can make larger intervals by adding octavation quotes. Note that quotes or commas do not determine the absolute height of a note; the height of a note is relative to the previous one. For example: `c f,` goes down; `f, f` are both the same; `c' c` are the same; and `c g'` goes up:

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



Here's an example of the difference between relative mode and "normal" (non-relative) mode:

```
\relative a {
\clef bass
  a d a e d c' d'
}
```






```
\clef bass
  a d a e d c' d'
```



SUMMARY

The following table summarizes the syntax learned so far in this section.

Syntax	Description	Example
<code>r4 r8</code>	rest	
<code>~</code>	tie	
<code>\key es \major</code>	key signature	

note'

raise octave

*note,*

lower octave



A slur is drawn across many notes, and indicates bound articulation (legato). The starting note and ending note are marked with a “(” and a “)” respectively:

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



If you need two slurs at the same time (one for articulation, one for phrasing), you can also make a phrasing slur with \ (and \).

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



Beams are drawn automatically, but if you don't like the choices, you can enter beams by hand. Surround the notes to be grouped with [and]:

```
[a8 ais] [d es r d]
```



To print more than one staff, each piece of music that makes up a staff is marked by adding `\context Staff` before it. These `Staff`'s are then grouped inside `<` and `>`, as is demonstrated here:

```
<
  \context Staff = staffA { \clef violin c'' }
  \context Staff = staffB { \clef bass c }
>
```



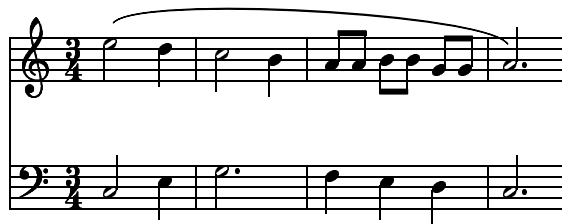
In this example, `staffA` and `staffB` are names that are given to the staves. It doesn't matter what names you give, as long as each staff has a different name.

We can now typeset a melody with two staves:

```

\score {
  \notes
  < \context Staff = staffA {
    \time 3/4
    \clef violin
    \relative c'' { e2 ( d4 c2 b4 [a8 a] [b b] [g g] )a2. }
  }
  \context Staff = staffB {
    \clef bass
    c2 e4 g2.
    f4 e d c2.
  }
  >
  \paper {}
}

```



Notice that the time signature is specified in one melody staff only (the top staff), but is printed on both. LilyPond knows that the time signature should be the same for all staves.

Common accents can be added to a note using `-.`, `--`, `->`:

```
c-. c-- c->
```



Dynamic signs are made by adding the markings to the note:

```
c-\ff c-\mf
```



Crescendi and decrescendi are started with the commands `\<` and `\>`. The command `\!` finishes a crescendo on the following note.

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



Chords can be made by surrounding notes with `<` and `>`:

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



You can combine beams and ties with chords. Beam and tie markings must be placed outside the chord markers:

```
r4 [<c8 e g> <c8 f a>] ~ <c8 f a>
```



When you want to combine chords with slurs and dynamics, technical detail crops up: you have to type these commands next to the notes, which means that they have to be inside the < >. Don't get confused by the chord < > and the dynamic \< \>!

```
r4 <c8 e g \> ( > <c e g> <c e g> < ) \! c8 f a>
```



There is one caution when using chords: if you use a chord at the very beginning of the piece, LilyPond might not understand what you want:

```
\score { \notes <c'2 e'2> }
```



If you have a piece that begins with a chord, you must explicitly state that the notes of the chord are to be put on the same staff, in the same voice. This is done by specifying \context Staff or \context Voice for the notes:

```
\score { \notes \context Voice <c'2 e'2> }
```



SUMMARY

Syntax

()

Description

slur

Example



`\(\)`

phrasing slur

`[]`

beam

`< \context Staff ... >`

more staves

`-> -.`

articulations

`-\mf -\sfz`

dynamics

`\< \!`

crescendo

`\> \!`

decrescendo

`< >`

chord



Now you know the basic ingredients of a music file, so this is the right moment to try your at hand at doing it yourself: try typing some simple examples, and experiment a little.

When you're comfortable with the basics, you might want to read the rest of this chapter. It continues in tutorial-style, but it is much more in-depth, dealing with more advanced topics such as lyrics, chords, orchestral scores and parts, fine tuning of output, polyphonic music, and integrating text and music.

2.4 Printing lyrics

In this section we shall explain how to typeset the following fragment of The Free Software Song:

Join us now_____and share the soft — ware;



To print lyrics, you must enter them and then instruct lilypond to print the lyrics. You can enter lyrics in a special input mode of LilyPond. This mode is called Lyrics mode, and it is introduced by the keyword `\lyrics`. The purpose of this mode is that you can enter lyrics as plain text, punctuation, and accents without any hassle.

Syllables are entered like notes, but with pitches replaced by text. For example, **Twin-kle twin-kle** enters four syllables. Note that the hyphen has no special meaning for lyrics, and does not introduce special symbols.

Spaces can be introduced into a lyric either by using quotes: "He could" not or by using an underscore without quotes: He_ could not. All unquoted underscores are converted to spaces.

These are the lyrics for the free software song:

```
\lyrics {
  Join us now __ and
  share the soft -- ware; }
```

As you can see, extender lines are entered as `__`. This will create an extender, which is a line that extends over the entire duration of the lyric. This line will run all the way to the start of the next lyric, so you may want to shorten it by using a blank lyric (using `_`).

You can use ordinary hyphens at the end of a syllable, i.e.

soft- ware

but then the hyphen will be attached to the end of the first syllable.

If you want them centered between syllables you can use the special `--` lyric as a separate word between syllables. The hyphen will have variable length depending on the space between the syllables and it will be centered between the syllables.

Normally the notes that you enter are transformed into note heads. Note heads alone make no sense, so they need surrounding information: a key signature, a clef, staff lines, etc. They need *context*. In LilyPond, these symbols are created by objects called ‘interpretation contexts’. Interpretation contexts exist for generating notation (‘notation context’) and for generating sound (‘performance context’). These objects only exist while LilyPond is executing.

When LilyPond interprets music, it will create a Staff context. We don’t want that default here, because we want lyric. The command

```
\context Lyrics
```

explicitly creates an interpretation context of **Lyrics** type to interpret the song text that we entered.

The melody of the song doesn’t offer anything new:

```

\notes \relative c' {
  \time 7/4
  d'2 c4 b16 ( a g a b a b ) c a2
  b2 c4 b8 ( a16 g ) a4 g2 }

```

Both can be combined with the `\addlyrics`:

```

\addlyrics
\notes \relative c' ...
\context Lyrics \lyrics ...

```

The lyrics are also music expressions, similar to notes. Each syllable of the lyrics is put under a note of the melody. The complete file is listed here:

```

\score { \notes { \addlyrics
  \notes \relative c' {
    \time 7/4
    d'2 c4 b16 ( a g a b a b ) c a2
    b2 c4 b8 ( a16 g ) a4 g2 }
  \context Lyrics \lyrics {
    Join us now __ and
    share the soft -- ware; }
}
\paper { linewidth = -1. }
}

```

2.5 A melody with chords

In this section we show how to typeset a melody with chord accompaniment. This file is included as ‘input/tutorial/flowing.ly’.

```

\include "paper16.ly"
melody = \notes \relative c' {
  \partial 8
  \key c \minor
  g8 |
  c4 c8 d [es ( ) d] c4 | f4 f8 g [es( ) d] c g |
  c4 c8 d [es ( ) d] c4 | d4 es8 d c4.
  \bar "|."
}

accompaniment = \chords {
  r8
  c2:3- f:3-.7 d:min es4 c8:min r8
  c2:min f:min7 g:7^3.5 c:min }

\score {
  \simultaneous {
    %\accompaniment
    \context ChordNames \accompaniment

```



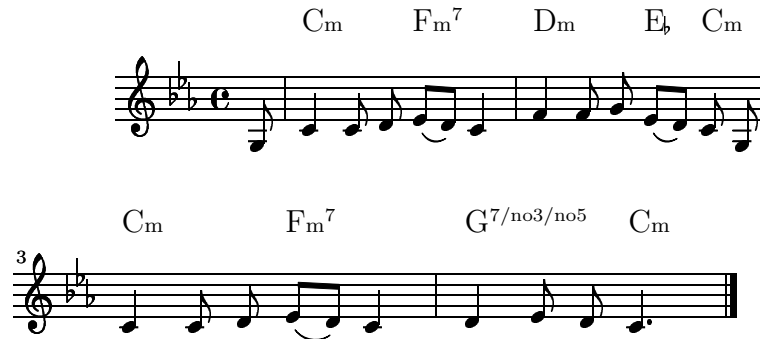
```

\context Staff = mel {
  \melody
}

}
\midi { \tempo 4=72 }
\paper { linewidth = 10.0\cm }
}

```

The result would look this.



We will dissect this file line by line.

```
\include "paper16.ly"
```

Smaller size (suitable for inclusion in a book).

```
melody = \notes \relative c' {
```

The structure of the file will be the same as the previous one: a `\score` block with music in it. To keep things readable, we will give different names to the different parts of music, and use the names to construct the music within the score block.

```
\partial 8
```

The piece starts with an anacrusis (or “pickup”) of one eighth.

```

c4 c8 d [es () d] c4 | f4 f8 g [es() d] c g |
c4 c8 d [es () d] c4 | d4 es8 d c4.
\bar "|."

```

We use explicit beaming. Since this is a song, we turn automatic beams off and use explicit beaming where needed.

```
}
```

This ends the definition of `melody`.

```
text = \lyrics {
```

This defines the lyrics, similar to what we have seen before.

```
accompaniment = \chords {
```

We'll put chords over the music. To enter them, we use a special mode analogous to `\lyrics` and `\notes` mode, where you can give the names of the chords you want instead of listing the notes comprising the chord.

```
r8
```

There is no accompaniment during the anacrusis.

```
c2:3-
```

This is a c minor chord, lasting for a half note. Chords are entered using the tonic. Notes can be changed to create different chords. In this case, a lowered third is used (making a C major chord into a C minor chord). The code for this is 3-.

```
f:3-.7
```

Similarly, 7 modifies (adds) a seventh, which is small by default to create the **f a c e s** chord. Multiple modifiers must be separated by dots.

```
d:min es4 c8:min r8
```

Some modifiers have predefined names, e.g. `min` is the same as 3-, so **d-min** is a minor **d** chord.

```
c2:min f:min7 g:7^3.5 c:min }
```

A named modifier `min` and a normal modifier 7 do not have to be separated by a dot. Tones from a chord are removed with chord subtractions. Subtractions are started with a caret, and they are also separated by dots. In this example, `g:7^3.5` produces a minor seventh (a G7 chord without the third or the fifth). The brace ends the sequential music.

```
\score {
  \simultaneous {
```

We assemble the music in the `\score` block. Melody, lyrics and accompaniment have to sound at the same time, so they should be `\simultaneous`.

```
%\accompaniment
```

Chord mode generates notes grouped in `\simultaneous` music. If you remove the comment sign, you can see the chords in normal notation: they will be printed as note heads on a separate staff. To print them as chords names, they have to be interpreted as being chords, not notes. This is done with the following command:

```
\context ChordNames \accompaniment
```

Normally the notes that you enter are transformed into note heads. Note heads alone make no sense, so they need surrounding information: a key signature, a clef, staff lines, etc. They need *context*. In LilyPond, these symbols are created by objects called ‘interpretation contexts’. Interpretation contexts exist for generating notation (‘notation context’) and for generating sound (‘performance context’). These objects only exist while LilyPond is executing.

When LilyPond interprets music, it will create a Staff context. If the % sign in the previous line were removed, you could see that mechanism in action.

We don’t want that default here, because we want chord names. The command above explicitly creates an interpretation context of `ChordNames` type to interpret the music `\accompaniment`.

```
\context Staff = mel {
```

We place the melody on a staff called `mel`. We give it a name to differentiate it from the one that would contain note heads for the chords, if you would remove the comment before the “note heads” version of the accompaniment. By giving this staff a name, it is forced to be different.

```
\property Staff.autoBeaming = ##f
```

An interpretation context has variables, called properties, that tune its behavior. One of the variables is `autoBeaming`. Setting this Staff’s property to `##f`, which is the boolean value *false*, turns the automatic beaming mechanism off for the current staff.

```
\melody
}
```

Finally, we put the melody on the current staff. Note that the `\property` directives and `\melody` are grouped in sequential music, so the property settings are done before the melody is processed.

```
\midi { \tempo 4=72}
```

MIDI (Musical Instrument Digital Interface) is a standard for connecting and recording digital instruments. So a MIDI file is like a tape recording of an instrument. The `\midi`

block makes the music go to a MIDI file, so you can listen to the music you entered. It is great for checking the music. Whenever you hear something weird, you probably hear a typing error.

Syntactically, `\midi` is similar to `\paper { }`, since it also specifies an output method. You can specify the tempo using the `\tempo` command, in this case the tempo of quarter notes is set to 72 beats per minute.

```
\paper { linewidth = 10.0\cm }
```

We also want notation output. The linewidth is short so that the piece will be set in two lines.

2.6 More stanzas

If you have multiple stanzas printed underneath each other, the vertical groups of syllables should be aligned around punctuation. LilyPond can do this if you tell it which lyric lines belong to which melody. We show how you can do this by showing how you could print a frivolous fragment of a fictional Sesame Street duet.

```
\score {
\addlyrics
  \notes \relative c'' \context Voice = duet { \time 3/4
    g2 e4 a2 f4 g2. }
  \lyrics \context Lyrics <
  \context LyricsVoice = "duet-1" {
    \property LyricsVoice . stanza = "Bert"
    Hi, my name is bert. }
  \context LyricsVoice = "duet-2" {
    \property LyricsVoice . stanza = "Ernie"
    Ooooo, ch\’e -- ri, je t’aime. }
  >
}
```



Bert Hi, my name is bert.
Ernie Ooooo, ché – ri, je t’aime.

To this end, give the Voice context an identity, and set the LyricsVoice to a name starting with that identity followed by a dash. In the following example, the Voice identity is `duet`,

```
\context Voice = duet {
  \time 3/4
  g2 e4 a2 f4 g2. }
```

and the identities of the LyricsVoices are `duet-1` and `duet-2`.

```
\context LyricsVoice = "duet-1" {
  Hi, my name is bert. }
```

```
\context LyricsVoice = "duet-2" {
  Ooooo, ch\'e -- ri, je t'aime. }
```

We add the names of the singers. This can be done by setting `LyricsVoice.Stanza` (for the first system) and `LyricsVoice.stz` for the following systems. Note that you must surround dots with spaces in `\lyrics` mode.

```
\property LyricsVoice . stanza = "Bert"
...
\property LyricsVoice . stanza = "Ernie"
```

The convention for naming `LyricsVoice` and `Voice` must also be used to get melismata on rests correct.

2.7 More movements

The program `lilypond` only produces sheet music and does not create titles, subtitles, or print the composer's name. To do that, you need to use `ly2dvi`, which comes with LilyPond. `ly2dvi` creates the title, then calls `lilypond` to format the sheet music. In this section, we show you how to create titles like this:

Two miniatures

Opus 1.

Up



Opus 2.

Down



For example, consider the following file (`miniatures.ly`)

```
\version "1.5.72"
\header {
  title = "Two miniatures"
  composer = "F. Bar Baz"
  tagline = "small is beautiful" }

\paper { linewidth = -1.0 }

%{

Mental note: discuss Schenkerian analysis of these key pieces.

%}
```

```

\score {
  \notes { c'4 d'4 }
  \header {
    opus = "Opus 1."
    piece = "Up" }
}
\score {
  \notes { d'4 c'4 }
  \header {
    opus = "Opus 2."
    piece = "Down" }
}

```

The information for the global titling is in a so-called header block. The information in this block is not used by LilyPond, but it is passed into `ly2dvi`, which uses this information to print titles above the music. the `\header` block contains assignments. In each assignment, a variable is set to a value. The header block for this file looks like this

```

\header {
  title = "Two miniatures"
  composer = "F. Bar Baz"
  tagline = "small is beautiful"
}

```

When you process a file with `ly2dvi`, a signature line is printed at the bottom of the last page. This signature is produced from the `tagline` field of `\header`. The default "Lily was here, *version number*" is convenient for programmers: archiving the layouts of different versions allows programmers to compare the versions using old print-outs.

Many people find the default tagline ("Lily was here") too dull. If that is the case, you can change `tagline` to something else, as shown above.

```

\paper {
  linewidth = -1.0 }

```

A paper block at top level (i.e. not in a `\score` block) sets the default page layout. The following `\score` blocks don't have `\paper` sections, so the settings of this block are used.

The variable `linewidth` normally sets the length of the systems on the page. However, a negative value has a special meaning. If `linewidth` is less than 0, no line breaks are inserted into the score, and the spacing is set to natural length: a short phrase takes up little space, a longer phrase takes more space, all on the same line.

```
%{
```

```
Mental note: discuss Schenkerian analysis of these key pieces.
```

```
%}
```

Mental notes to yourself can be put into comments. There are two types of comments. Line comments are introduced by `%`, and block comments are delimited by `%{` and `%}`.

```

\score {
  \notes { c'4 d'4 }

```

In previous examples, most notes were specified in relative octaves (i.e. each note was put in the octave that is closest to its predecessor). Besides relative, there is also absolute octave specification, which you get when you don't include `\relative` in your input file. In this input mode, the middle C is denoted by `c'`. Going down, you get `c c, c,,` etc. Going up, you get `c'' c'''` etc.

When you're copying music from existing sheet music, relative octaves are probably the easiest to use: you have to do less typing, and errors are easily spotted. However, if you write LilyPond input directly, either by hand (i.e. composing) or by computer, absolute octaves may be easier to use.

```
\header {
```

The `\header` is normally at the top of the file, where it sets values for the rest of the file. If you want to typeset different pieces from one file (for example, if there are multiple movements, or if you're making an exercise book), you can put different `\score` blocks into the input file. `ly2dvi` will assemble all LilyPond output files into a big document. The contents of `\header` blocks specified within each score is used for the title of that movement.

```
    opus = "Opus 1."
    piece = "Up" }
```

For example, the Opus number is put at the right, and the "piece" string will be at the left.

```
\version "1.5.72"
\header {
  title = "Two miniatures"
  composer = "F. Bar Baz"
  tagline = "small is beautiful" }

\paper { linewidth = -1.0 }

\score {
  \notes { c'4 d'4 }
  \header {
    opus = "Opus 1."
    piece = "Up" }
}
\score {
  \notes { d'4 c'4 }
  \header {
    opus = "Opus 2."
    piece = "Down" }
}

\version "1.5.72"
```

Lilypond and its language are still under development, and occasionally details of the syntax are changed. The `version` fragment indicates which version of lilypond the input file was written for. When you compile this file, the version number will be checked and you will get a warning when the file is too old. This version number is also used by the

`convert-ly` program (See Section 7.1 [Invoking `convert-ly`], page 142), which can be used to update the file to the latest lily version.

2.8 A piano excerpt

Our eighth subject is a piece of piano music. The fragment in the input file is a piano reduction of the G major Sinfonia by Giovanni Battista Sammartini, composed around 1740. It's in the source package under the name `'input/tutorial/sammartini.ly'`.

```
\version "1.5.68"

\include "paper16.ly"

viola = \notes \relative c' \context Voice = viola {
  <c4-\arpeggio g' c>
  \voiceTwo
  g'8. b,16
  s1 s2. r4
  g
}

oboes = \notes \relative c'' \context Voice = oboes {
  \voiceOne
  s4 g8. b,16 c8 r <e'8. g> <f16 a>
  \grace <e8( g> <d4 )f> <c2 e>
  \times 2/3 { <d8 f> <e g> <f a> }
  <
    { \times 2/3 { a8 g c } c2 }
  \\\
    { f,8 e e2 }
  >

  \grace <c,8( e> <)b8. d8.-\trill> <c16 e> |
  [<d ( f> <)f8. a>] <)b,8 d> r [<d16( f> <f8. )a>] <b,8 d> r |
  [<c16( e> <)e8. g>] <c8 e,>
}

hoomPah = \repeat unfold 8 \notes
  \transpose c' {
\translator Staff = down
\stemUp
c8
\translator Staff = up
\stemDown
c'8 }

bassvoices = \notes \relative c' {
```



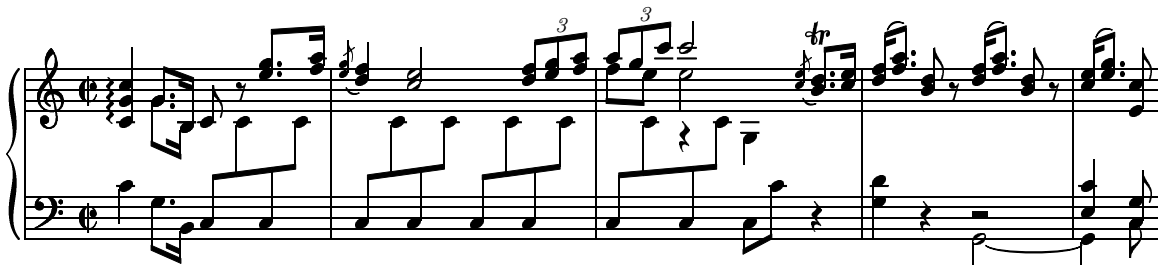
```

c4 g8. b,16
\context Voice \hoomPah
\translator Staff = down
\stemBoth

[c8 c'8] r4
<g d'> r4
< { r2 <e4 c'> <c8 g'> } \\\
  { g2 ~ | g4 c8 } >
}

\score {
  \context PianoStaff \notes <
    \context Staff = up <
      \oboes
      \viola
    >
    \context Staff = down < \time 2/2 \clef bass
      \bassvoices
    >
  >
  \midi { }
  \paper {
    indent = 0.0
    linewidth = 15.0 \cm }
}

```



As you can see, this example features multiple voices on one staff. To make room for those voices, their notes should be stemmed in opposite directions.

LilyPond includes the identifiers `\stemUp`, `\stemDown` along with some other commonly used formatting instructions, but to explain how it works, we wrote our own here. Of course, you should use predefined identifiers like these if possible: you will be less affected by changes between different versions of LilyPond.

```
viola = \notes \relative c' \context Voice = viola {
```

In this example you can see multiple parts on a staff. Each part is associated with one notation context. This notation context handles stems and dynamics (among other things). The type name of this context is `Voice`. For each part we have to make sure that there is precisely one `Voice` context, so we give it a unique name (`'viola'`).

```
<c4-\arpeggio g' c>
```

The delimiters < and > are shorthands for `\simultaneous {` and `}`. The expression enclosed in < and > is a chord.

`\arpeggio` typesets an arpeggio sign (a wavy vertical line) before the chord.

```
\voiceTwo
```

We want the viola to have stems down, and have all the other characteristics of a second voice. This is enforced using the `\voiceTwo` command: it inserts instructions that makes stem, ties, slurs, etc. go down.

```
g'8. b,16
```

Relative octaves work a little differently with chords. The starting point for the note following a chord is the first note of the chord. So the `g` gets an octave up quote: it is a fifth above the starting note of the previous chord (the central C).

```
s1 s2. r4
```

`s` is a spacer rest. It does not print anything, but it does have the duration of a rest. It is useful for filling up voices that temporarily don't play. In this case, the viola doesn't come until one and a half measure later.

```
oboes = \notes \relative c'' \context Voice = oboe {
```

Now comes a part for two oboes. They play homophonically, so we print the notes as one voice that makes chords. Again, we insure that these notes are indeed processed by precisely one context with `\context`.

```
\voiceOne s4 g8. b,16 c8 r <e'8. g> <f16 a>
```

The oboes should have stems up to keep them from interfering with the staff-jumping bass figure. To do that, we use `\voiceOne`.

```
\grace <e8( g> <d4 )f> <c2 e>
```

`\grace` introduces grace notes. It takes one argument, in this case a chord. A slur is introduced starting from the `\grace` ending on the following chord.

```
\times 2/3
```

Tuplets are made with the `\times` keyword. It takes two arguments: a fraction and a piece of music. The duration of the piece of music is multiplied by the fraction. Triplets make notes occupy 2/3 of their notated duration, so in this case the fraction is 2/3.

```
{ <d8 f> <e g> <f a> }
```

The piece of music to be 'tripletted' is sequential music containing three chords.

```
<
```

At this point, the homophonic music splits into two rhythmically different parts. We can't use a sequence of chords to enter this, so we make a "chord of sequences" to do it. We start with the upper voice, which continues with upward stems:

```
{ \times 2/3 { a8 g c } c2 }
\\
```

The easiest way to enter multiple voices is demonstrated here. Separate the components of the voice (single notes or entire sequences) with `\\` in a simultaneous music expression. The `\\` separators split first voice, second voice, third voice, and so on.

As far as relative mode is concerned, the previous note is the `c''2` of the upper voice, so we have to go an octave down for the `f`.

```

    f,8 e e2
} >

```

This ends the two-part section.

```

\stemBoth
\grace <c,8( e> <)b8. d8.-\trill> <c16 e> |

```

`\stemBoth` ends the forced stem directions. From here, stems are positioned as if it were single part music.

The bass has a little hoom-pah melody to demonstrate parts switching between staves. Since it is repetitive, we use repeats:

```

hoomPah = \repeat unfold 8

```

The unfolded repeat prints the notes in its argument as if they were written out in full eight times.

```

\notes \transpose c' {

```

Transposing can be done with `\transpose`, which takes two arguments. The first specifies what central C should be transposed to. The second is the to-be-transposed music. As you can see, in this case, the transposition has no effect, as central C stays at central C.

The purpose of this no-op is to protect it from being interpreted as relative notes. Relative mode can not be used together with transposition, so `\relative` will leave the contents of `\hoomPah` alone. We can use it without having to worry about getting the motive in a wrong octave.

```

\translator Staff = down
\stemUp
c8
\translator Staff = up
\stemDown
c'8 }

```

Voices can switch between staves. Here you see two staff switching commands. The first one moves to the lower staff, the second one to the lower one. If you set the stem directions explicitly (using the identifiers `\stemUp` and `\stemDown`, the notes can be beamed together (despite jumping between staves).

```

bassvoices = \notes \relative c' {
c4 g8. b,16
\autochange Staff \hoomPah \context Voice
\translator Staff = down

```

We want the remaining part of this melody on the lower staff, so we do a manual staff switch here.

```

\context Voice = reallyLow {\stemDown g2 ~ | g4 c8 } >

```

After skipping some lines, we see `~`. This mark makes ties. Note that ties and slurs are different things. A tie can only connect two note heads of the same pitch, whereas a slur can connect many notes with one curve.

```

\context PianoStaff

```

A special context is needed to get cross staff beaming right. This context is called `PianoStaff`.

```
\context Staff = bottom < \time 2/2 \clef bass
```

The bottom staff must have a different clef.

```
indent = 0.0
```

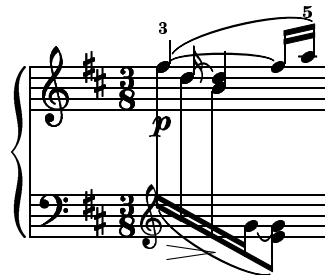
To make some more room on the line, the first (in this case the only) line is not indented. The line still looks very cramped, but that is due to the page layout of this document.

2.9 Fine tuning a piece

In this section we show some ways to fine tune the final output of a piece. We do so using a single measure of a moderately complex piano piece: a Brahms intermezzo (opus 119, no. 1). Both fragments (the tuned and the untuned versions) are in ‘input/tutorial/’.

The code for the untuned example shows us some new things.

```
\version "1.5.68"
\score {
  \notes\context PianoStaff <
  \context Staff = up
  \relative c'' <
    { \key d\major
      fis4-3_\p ( ~
        fis16 )a-5 } \\\
    {
      fis16( \> d \! b \translator Staff = down
        \clef treble g ~ < g8 )e> } \\\
    { s16
      d'
      ~ < d4 b4 > }
  >
  \context Staff = down {
    \key d \major
    \time 3/8 \clef bass s4. }
  >
  \paper { linewidth = -1. }
}
```

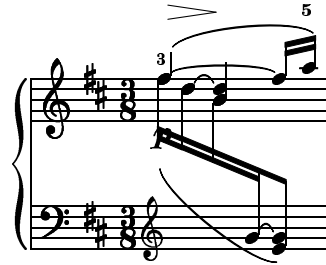


Hairpin dynamics can be easily added to Lilypond scores. Beginning a crescendo is indicated with \< and ending a crescendo is indicated with \!. A decrescendo can be

indicated with `\>` and `\!`. Absolute dynamics (sometimes called “Letter dynamics”) can be entered using `\p`, `\mf`, etc. All these dynamics will apply to the whole chord where they are entered, but for syntactical reasons they must be attached to one of the notes of the chord.

Fingering indications are entered with `-N`, where N is a digit.

Now that we have the basic piece of music entered, we want to fine tune it so that we get something that resembles the original printed edition by Schott/Universal Edition:



Fine tuning involves overriding the defaults of the printing system. We do this by setting variables which control how Lilypond prints symbols. Printed symbols are called graphical objects (often abbreviated to *grob*). Each object is described by a bunch of settings. Every setting is a variable: it has a name and a value which you can change. These values determine the fonts, offsets, sub-routines to be called on the object, etc. The initial values of these settings are set in the Scheme file ‘`scm/grob-description.scm`’.

We start with the slur in the upper part, running from F sharp to A. In the printed edition, this slur runs from stem to stem; in our version, the slur begins at the note head of the F sharp. The following property setting forces all slurs to run from stem to stem (not from or to note heads!).

```
\property Voice.Slur \set #'attachment = #'(stem . stem)
```

More precisely, this command modifies the definition of the `Slur` object in the current `Voice`. The variable `attachment` is set to the pair of symbols ‘`(stem . stem)`’.

Although this fixes the slur, it isn’t very helpful for fine tuning in general: the lilypond back-end supports approximately 240 variables like `attachment`, each with their own meaning and own type (eg. number, symbol, list, etc). Besides slur, LilyPond has 80 different types of graphical objects, that may be created in 14 different context types besides `Voice`.

The interesting information is how you can figure out which properties to tune for your own scores. To discover this, you must have a copy of the internals document. This is a set of HTML pages which should be included if you installed a binary distribution¹. These HTML pages are also available on the web: go to the lilypond website, click “Documentation: Index” on the side bar, look in the “Information for uses” section, and click on “Documentation of internals”.

You might want to bookmark either the HTML files on your disk, or the one on the web (the HTML on your hard drive will load much faster than the ones on the web!). One word of caution: the internals documentation is generated from the definitions that lily uses. For that reason, it is strongly tied to the version of LilyPond that you use. Before you proceed,

¹ You can also compile them by executing `make -C Documentation/user/ out/lilypond-internals.html` in the source package.

please make sure that you are using the documentation that corresponds to the version of LilyPond that you are using.

Suppose that you wanted to tune the behavior of the slur. The first step is to get some general information on slurs in lilypond. Turn to the index, and look up “slur”. The section on slurs says

The grob for this object is **Slur**, generally in **Voice** context.

So the graphical object for this object is called **Slur**, and slurs are created in the **Voice** context. If you are reading this tutorial in the HTML version, then you can simply click **Slur**, otherwise, you should look it up the internal documentation: click “grob overview” and select “slur” (the list is alphabetical).

Now you get a list of all the properties that the slur object supports, along with their default values. Among the properties we find the **attachment** property with its default setting. The property documentation explains that the following setting will produce the desired effect:

```
\property Voice.Slur \set #'attachment = #'(stem . stem)
```

If you ran the previous example, you have unknowingly already used this kind of command. The ‘ly/property-init.ly’ contains the definition of **\stemUp**:

```
stemUp = \property Voice.Stem \set #'direction = #1
```

Next we want to move the fingering ‘3’. In the printed edition it is not above the stem, but a little lower and slightly left of the stem. From the user manual we find that the associated graphical object is called **Fingering**, but how do we know if we should use **Voice** or **Staff**? In many cases, **Voice** is a safe bet, but you can also deduce this information from the internals documentation: if you visit the documentation of **Fingering**, you will notice

Fingering grobs are created by: **Fingering_engraver**

Clicking **Fingering_engraver** will show you the documentation of the module responsible for interpreting the fingering instructions and translating them to a **Fingering** object. Such a module is called an *engraver*. The documentation of the **Fingering_engraver** says

Fingering_engraver is part of contexts: **Voice** and **TabVoice**

so tuning the settings for **Fingering** should be done using either

```
\property Voice.Fingering \set ...
```

or

```
\property TabVoice.Fingering \set ...
```

Since the **TabVoice** is only used for tab notation, we see that the first guess **Voice** was indeed correct.

For shifting the fingering, we use the property **extra-offset**. The following command manually adds an offset to the object. We move it a little to the left, and 1.8 staff space downwards.

```
\property Voice.Fingering \set #'extra-offset = #'(-0.3 . -1.8)
```

The **extra-offset** is a low-level feature: it moves around objects in the printout; the formatting engine is completely oblivious to these offsets. The unit of these offsets are staff-spaces. The first number controls left-right movement; a positive number will move

the object to the right. The second number controls up-down movement; a positive number will move it higher.

We only want to offset a single object, so after the F-sharp we must undo the setting. The technical term is to revert the property.

```
\property Voice.Fingering \revert #'extra-offset
```

There are three different types of variables in LilyPond, something which can be confusing at first (and for some people it stays confusing :). Variables such as `extra-offset` and `attachment` are called grob properties. They are not the same as translator properties, like `autoBeaming`. Finally, music expressions are internally stored using properties (so-called music properties). You will encounter music properties if you run Scheme functions on music using `\apply`.

The second fingering instruction should be moved up a little to avoid a collision with the slur. This could be achieved with `extra-offset`, but in this case, a simpler mechanism also works. We insert an empty text between the 5 and the note. The empty text pushes the fingering instruction away:

```
a^" "^#'(finger "5")
```

Lilypond tries to put fingering instructions as close to the notes as possible. To make sure that Lilypond doesn't do that, we disguise the fingering as text: `(finger "5")`.

Normally one would specify all dynamics in the same voice, so that dynamics (such as **f** and **p**) will be aligned with hairpins. But in this case, we don't want the decrescendo to be aligned with the piano sign. We achieve this by putting the dynamic markings in different voices. The crescendo should be above the upper staff. This can be forced by using the precooked command

```
\dynamicsUp
```

However, if you do that the decrescendo will be too close to the upper voice and collide with the stems. Looking at the manual for dynamics, we notice that “Vertical positioning of these symbols is handled by the `DynamicLineSpanner` grob.”. If we turn to the documentation of `DynamicLineSpanner`, we find that `DynamicLineSpanner` supports several so-called ‘interfaces’. This object not only puts dynamic objects next to the staff (`side-position-interface`), but it also groups dynamic objects (`axis-group-interface`), is considered a dynamic sign itself (`dynamic-interface`), and is an object. It has the standard `grob-interface` with all the variables that come with it.

For the moment we are interested in side positioning:

```
side-position-interface
```

Position a victim object (this one) next to other objects (the support). In this case, the direction signifies where to put the victim object relative to the support (left or right, up or down?)

Between the object and its support (in this case the notes in the voice going down), there should be more space. This space is controlled by `padding`, so we increase it.

```
\property Voice.DynamicLineSpanner \override #'padding = #5.0
```

This command is almost like the command for setting slur attachments, but subtly different in its details. Grob properties can be manipulated with two commands: `\override` extends the variables with a setting, and `\revert` releases this setting. This has a certain

theoretical appeal: the operations are simple and symmetric. But for practical use, it can be cumbersome. The commands act like parentheses: you should carefully balance the use of `\override` and `\revert`. The `\set` command is more friendly: it first does a `\revert` followed by `\override`.

Brahms uses music notation is a slightly unorthodox way. Ties usually happen only within one voice. In this piece, the composer gladly produces ties that jump voices. We deal with this by faking these ties: whenever we need such a tie, we insert a notehead in a different voice, and blank the stem. This is done in the following snippet of code.

```
\property Voice.Stem \set #'transparent = ##t
d'
```

Blanking the stem should be done for only one object. One of the ways to achieve that, is by setting the property before a note. Reverting it afterwards is tedious, so for setting a property only once, we have the syntax `\once`: it reverts the property directly before proceeding to the next step in time.

The `\once` keyword is added to `\property`.

Finally, the last tie is forced up using `\tieUp`.

Here's the complete "fine tuned" version, which includes all the modifications we discussed in this section:

```
\version "1.5.68"
\score {
  \notes\context PianoStaff <
  \context Staff = up
  \relative c'' <
    { \key d\major
      \property Voice.Slur \set #'attachment = #'(stem . stem)

      \property Voice.Fingering \set #'extra-offset = #'(-0.3 . -1.8)■
      fis4-3_\p ( ~

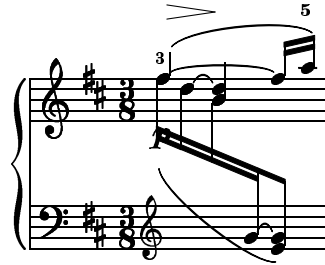
      \property Voice.Fingering \revert #'extra-offset
      fis16 )a^" ^#'(finger "5") } \\\
    {
      \dynamicUp
      \property Voice.DynamicLineSpanner \override #'padding = #5.0■
      \tieUp
      fis16( \> d \! b \translator Staff = down
    \stemUp
    \clef treble g ~ < g8 )e> } \\\
    { s16
  \property Voice.Stem \set #'transparent = ##t
  d'
  \property Voice.Stem \revert #'transparent
  ~ < d4 b4 > }
}
\context Staff = down {
```



```

\key d \major
\time 3/8 \clef bass s4. }
>
\paper { linewidth = -1. }
}

```



2.10 An orchestral score

Our next two examples demonstrate one way to create an orchestral score in LilyPond. When typesetting a piece for several instruments, you'll want to create a full score (for the conductor) along with several individual parts (for the players).

LilyPond is well suited for this task. We will declare the music for each instrument individually, giving the music of each instrument its own name. These pieces of music are then combined in different `\score` blocks to produce different combinations of instruments (for example, one `\score` block may only include the cello part; another `\score` block may be for all the strings, and yet another `\score` block may be for all parts together).

This orchestral score example consists of three input files. In the first file, `'os-music.ly'`, we define the music for all instruments. This file will be used for producing the score and the separate parts, but the file doesn't produce any sheet music itself. Other files reference this file by doing `\include "os-music.ly"`.

```

% os-music.ly
\header {
  title = "Zo, goed lieverd?"
  subtitle = "How's, this babe?"
  composer = "JCN"
  opus = "1"
  piece = "Laid back"
}
global = {
  \time 2/4
  \skip 2*4 \bar "|."
}
Key = \notes \key as \major
flautoI = \notes\relative c'' {
  f8 g f g f g f g

```

```

    bes as bes as bes as bes as
  }
  flautoII = \notes\relative c'' {
    as8 bes as bes R1 d4 ~ d
  }
  tromboI = \notes\relative c'' {
    c4. c8 c8 c4. es4 r as, r
  }
  tromboII = \notes\relative c'' {
    as4. as8 as8 as4. R1*1/2 as4 es'
  }
  timpani = \notes\relative c, {
    \times 2/3 { f4 f f }
    \times 4/5 { as8 as as as as }
    R1
  }
  corno = \notes\relative c' {
    bes4 d f, bes d f, bes d
  }

```

We will not examine this example line by line, since you already know most of it. We'll examine a few lines which contain new elements.

```

global = {
  \time 2/4
  \skip 2*4 \bar "|."
}

```

Declare setting to be used globally. The `\skip` command produces no output, but moves forward in time: in this case, the duration of a half note (2), and that four times (`*4`). This brings us to the end of the piece, and we can set the end bar. You can use `s` as a shortcut for `\skip` (the last line of this section would be `s2*4 \bar"|."`).

```
Key = \notes \key as \major
```

Declare the key signature of the piece and assign it to the identifier *Key*. Later on we'll use `\Key` for all staves except those for transposing instruments.

2.10.1 The full score

The second file, 'input/tutorial/os-score.ly', reads the definitions of the first ('input/tutorial/os-music.ly'), and defines the `\score` block for the full conductor's score.

```

% os-score.ly
\include "os-music.ly"
\include "paper13.ly"

#(set-point-and-click! 'line-column)
#(define text-flat '((font-relative-size . -2)
  (music "accidentals--1")))

```

```

\score {
  <
    \global
    \property Score.BarNumber \override #'padding = #3
    \context StaffGroup = woodwind <
      \context Staff = flauti <
        \property Staff.midiInstrument = #"flute"
        \property Staff.instrument = "2 Flauti"
        \property Staff.instr = "Fl."
        \Key
        \context Voice=one { \voiceOne \flautoI }
        \context Voice=two { \voiceTwo \flautoII }
      >
    >
    \context StaffGroup = timpani <
      \context Staff = timpani <
        \property Staff.midiInstrument = #"timpani"
        \property Staff.instrument = #'(lines "Timpani" "(C-G)")
        \property Staff.instr = #"Timp."
        \clef bass
        \Key
        \timpani
      >
    >
    \context StaffGroup = brass <
      \context Staff = trombe <
        \property Staff.midiInstrument = #"trumpet"
        \property Staff.instrument = #'(lines "2 Trombe" "(C)")
        \property Staff.instr = #'(lines "Tbe." "(C)")
        \Key
        \context Voice=one \partcombine Voice
        \context Thread=one \tromboI
        \context Thread=two \tromboII
      >
      \context Staff = corni <
        \property Staff.midiInstrument = #"french horn"
        \property Staff.instrument = #'(lines "Corno"
          (columns "(E" ,text-flat ")"))
        \property Staff.instr = #'(lines "Cor."
          (columns "(E" ,text-flat ")"))
        \property Staff.transposing = #3
        \notes \key bes \major
        \context Voice=one \corno
      >
    >
  >
}

```

```

\paper {
  indent = 15 * \staffspace
  linewidth = 55 * \staffspace
  textheight = 90 * \staffspace
  \translator{
    \HaraKiriStaffContext
  }
}
\midi {
  \tempo 4 = 75
}
}

```

Zo, goed lieverd?

How's, this babe?

Opus 1.

LAI D BACK

The image shows a musical score for Opus 1. It consists of four staves: 2 Flauti, Timpani (C-G), 2 Trombe (C), and Corno (Bb). The music is in 2/4 time. The Flauti part features a melodic line with triplets and quintuplets. The Timpani part has a rhythmic pattern with triplets and quintuplets. The Trombe part includes a solo section marked 'Solo a2' and 'Solo II'. The Corno part has a melodic line with various intervals.

```
\include "os-music.ly"
```

First we need to include the music definitions we made in ‘os-music.ly’.

```

\set-point-and-click! 'line-column

```

This piece of Scheme code executes the function `set-point-and-click!` with the argument `line-column`. Editing input files can be complicated if you’re working with large files: if you’re digitizing existing music, you have to synchronize the .ly file, the sheet music on your lap and the sheet music on the screen. The point-and-click mechanism makes it easy to find the origin of an error in the LY file: when you view the file with Xdvi and click on a note, your editor will jump to the spot where that note was entered. For more information, see Section 3.2.4 [Point and click], page 53.

```

\define text-flat '((font-relative-size . -2)
  (music "accidentals--1"))

```

To name the transposition of the french horn, we will need a piece of text with a flat sign. LilyPond has a mechanism for font selection and kerning called Scheme markup text (See Section 3.16.4 [Text markup], page 109). The flat sign is taken from the music font, and its name is `accidentals--1` (The natural sign is called `accidentals-0`). The default font is too big for text, so we select a relative size of `-2`.

```
<
\global
```

All staves are simultaneous and use the same global settings.

```
\property Score.BarNumber \override #'padding = #3
```

LilyPond prints bar numbers at the start of each line, but unfortunately they end up a bit too close to the staff in this example. In LilyPond, a bar number is called *BarNumber*. *BarNumber* objects can be manipulated through their *side-position-interface*. One of the properties of a *side-position-interface* that can be tweaked is *padding*: the amount of extra space that is put between this and other objects. We set the padding to three staff spaces.

You can find information on all these kind of properties in LilyPond's automatically generated documentation in the online documentation or in the previous section of the tutorial.

```
\context StaffGroup = woodwind <
\context Staff = flauti <
```

A new notation context: the *StaffGroup*. *StaffGroup* can hold one or more *Staff*'s, and will print a big bracket at the left of the score. This starts a new staff group for the woodwind section (just the flutes in this case). Immediately after that, we start the staff for the two flutes, who also play simultaneously.

```
\property Staff.midiInstrument = #"flute"
```

Specify the instrument for MIDI output (see Section 3.18.2 [MIDI instrument names], page 116).

```
\property Staff.instrument = "2 Flauti"
\property Staff.instr = "Fl."
```

This defines the instrument names to be printed in the margin. *instrument* specifies the name for the first line of the score, *instr* is used for the rest of the score.

```
\Key
```

The flutes play in the default key.

```
\context Voice=one { \voiceOne \flautoI }
\context Voice=two { \voiceTwo \flautoII }
```

Last come the actual flute parts. Remember that we're still in simultaneous mode. We name both voices differently, so that LilyPond will create two *Voice* contexts. The flute parts are simple, so we specify manually which voice is which: *\voiceOne* forces the direction of stems, beams, slurs and ties up, *\voiceTwo* sets directions down.

```
>
>
```

Close the flutes staff and woodwind staff group.

```
\property Staff.instrument = #'(lines "Timpani" "(C-G)")
```

The timpani staff demonstrates a new piece of scheme markup, it sets two lines of text.

```
\context Voice=one \partcombine Voice
\context Thread=one \tromboI
\context Thread=two \tromboII
```

You have seen the notation contexts `Staff` and `Voice`, but here's a new one: `Thread`. One or more `Threads` can be part of a `Voice`. `Thread` takes care of note heads and rests; `Voice` combine note heads onto a stem.

For the trumpets we use the automatic part combiner (see Section 3.14.6 [Automatic part combining], page 94) to combine the two simultaneous trumpet parts onto the trumpet staff. Each trumpet gets its own `Thread` context, which must be named `one` and `two`). The part combiner makes these two threads share a `Voice` when they're similar, and splits the threads up when they're different.

```
\property Staff.instrument = #'(lines "Corno"
  (columns "(E" ,text-flat ")"))
```

The french horn (“Corno”) has the most complex scheme markup name, made up of two lines of text. The second line has three elements (columns) – the (E, the flat sign `text-flat` that we defined previously, and a final `)`). Note that we use a backquote instead of an ordinary quote at the beginning of the Scheme expression to be able to access the `text-flat` identifier, ‘unquoting’ it with a `“,”`.

```
\property Staff.transposing = #3
```

The french horn is to be tuned in E-flat, so we tell the MIDI back-end to transpose this staff by three steps.

Note how we can choose different tunings for the text input, sheet music output and, and MIDI output, using `\transpose` and the MIDI Staff property *transposing*.

```
\notes \key bes \major
```

Since the horn is transposing, it's in a different key.

```
indent = 15 * \staffspace
linewidth = 55 * \staffspace
```

We specify a big indent for the first line and a small linewidth for this tutorial.

Usually LilyPond's default setup of notation contexts (`Thread`, `Voice`, `Staff`, `Staffgroup`, `Score`) is just fine. But in this case we want a different type of `Staff` context.

```
\translator{
  \HaraKiriStaffContext
}
```

In orchestral scores it often happens that one instrument only has rests during one line of the score. `HaraKiriStaffContext` can be used as a regular `StaffContext` drop-in and will take care of the automatic removing of empty staves – so if the strings are the only instruments playing for a line, LilyPond will only print the string parts for that line of the score. This reduces the number of page turns (and the number of dead trees!) required in a score.

2.10.2 Extracting an individual part

The third file, ‘`os-flute-2.ly`’ also reads the definitions of the first (‘`os-music.ly`’), and defines the `\score` block for the second flute part.

```
\include "os-music.ly"
\include "paper16.ly"
```

```

\score {
  \context Staff <
    \property Score.skipBars = ##t
    \property Staff.midiInstrument = #"flute"
    \global
    \Key
    \flautoII
  >
  \header {
    instrument = "Flauto II"
  }
  \paper {
    linewidth = 80 * \staffspace
    textheight = 200 * \staffspace
  }
  \midi {
    \tempo 4 = 75
  }
}

```

Zo, goed lieverd?

How's, this babe?

Flauto II

Opus 1.

LAI D BACK



Because we separated the music definitions from the `\score` instantiations, we can easily define a second score with the music of the second flute. This is the part for the second flute player. Of course, we would make separate parts for all individual instruments if we were preparing the score for an orchestra.

```
\flautoII
```

In this individual part the second flute has a whole staff for itself, so we don't want to force stem or tie directions.

```

\header {
  instrument = "Flauto II"
}

```

The `\header` definitions were also read from 'os-music.ly', but we need to set the instrument for this particular score.

```
\property Score.skipBars = ##t
```

In the conductor's full score, all bars with rests are printed, but for the individual parts, we want to print one multimeasure rest instead of many consecutive empty bars. LilyPond will do this if `Score.skipBars` is set to true (`##t`).

2.11 Integrating text and music

Sometimes you might want to use music examples in a text that you are writing (for example a musicological treatise, a songbook, or (like us) the LilyPond manual). You can make such texts by hand, simply by importing a PostScript figure into your word processor. However, there is an automated procedure to reduce the amount of work.

If you use HTML, LaTeX, or texinfo, you can mix text and LilyPond code. A script called `lilypond-book` will extract the music fragments, run LilyPond on them, and put back the resulting notation. This program is fully described in the chapter Chapter 6 [Integrating text and music with `lilypond-book`], page 136. Here we show a small example. Since the example also contains explanatory text, we won't comment on the contents.

```
\documentclass[a4paper]{article}
\begin{document}
```

In a `lilypond-book` document, you can freely mix music and text. For example:

```
\begin{lilypond}
  \score { \notes \relative c' {
    c2 g'2 \times 2/3 { f8 e d } c'2 g4
  } }
\end{lilypond}
```

Notice that the music line length matches the margin settings of the document.

If you have no `\verb+\score+` block in the fragment, `\texttt{lilypond-book}` will supply one:

```
\begin{lilypond}
  c'4
\end{lilypond}
```

In the example you see here, two things happened: a `\verb+\score+` block was added, and the line width was set to natural length. You can specify many more options using LaTeX style options in brackets:

```
\begin[verbatim,11pt,singleline,
  fragment,relative,intertext="hi there!"]{lilypond}
  c'4 f bes es
\end{lilypond}
```

`\texttt{verbatim}` prints the LilyPond code in addition to the graphical score,
`\texttt{11pt}` selects the default music size,
`\texttt{fragment}` adds a score block,
`\texttt{relative}` uses relative mode for the fragment, and

`\texttt{intertext}` specifies what to print between the `\texttt{verbatim}` code and the music.

If you want to include large examples into the text, it may be more convenient to put the example in a separate file:

```
\lilypondfile[printfilename]{sammartini.ly}
```

The `\texttt{printfilename}` option adds the file name to the output.

```
\end{document}
```

Under Unix, you can view the results as follows.

```
$ cd input/tutorial
$ mkdir -p out/
$ lilypond-book --outdir=out/ lilbook.tex
lilypond-book (GNU LilyPond) 1.6.1
Reading 'input/tutorial/lilbook.tex'
Reading 'input/tutorial/sammartini.ly'
lots of stuff deleted
Writing 'out/lilbook.latex'
$ cd out
$ latex lilbook.latex
lots of stuff deleted
$ xdvi lilbook
```

Notice the `outdir` option to `lilypond-book`. Running `lilypond-book` and running `latex` creates a lot of temporary files, and you wouldn't want those to clutter up your working directory. Hence, we have them created in a separate subdirectory.

The result looks more or less like this:

In a `lilypond-book` document, you can freely mix music and text. For example:



Notice that the music line length matches the margin settings of the document.

If you have no `\score` block in the fragment, `lilypond-book` will supply one:



In the example you see here, a number of things happened: a `\score` block was added, and the line width was set to natural length. You can specify many more options using LaTeX style options in brackets:

```
c'4 f bes es
```

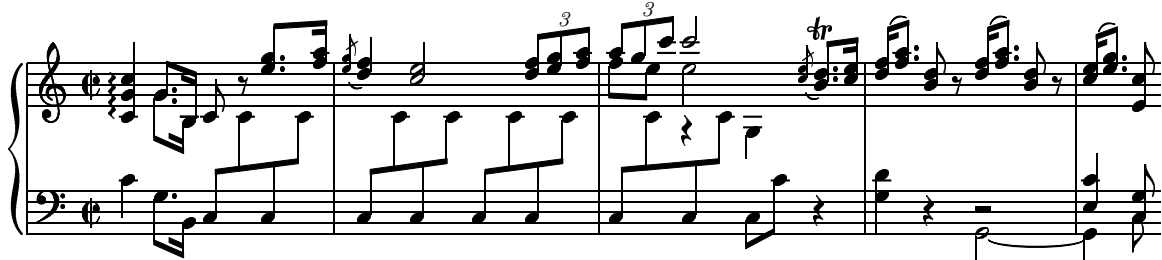
hi there!



`verbatim` also shows the LilyPond code, `11pt` selects the default music size, `fragment` adds a score block, `relative` uses relative mode for the fragment, and `intertext` specifies what to print between the `verbatim` code and the music.

If you include large examples into the text, it may be more convenient to put the example in a separate file:

```
'../../input/tutorial/sammartini.ly':
```



The `printfilename` option adds the file name to the output.

2.12 End of tutorial

This is the end of the tutorial. If you read everything until this point (and understood everything!) then you're now an accomplished lilypond hacker. From here you should try fiddling with input files or making your own input files. Come back to the reference manual for more information if you get stuck!

Don't forget to check out the templates, examples and feature tests. If you want to see real action LilyPond, head over to <http://www.mutopiaproject.org>, which has many examples of classical music typeset with LilyPond.

3 Reference Manual

This document describes GNU LilyPond and its input format. The last revision of this document was made for LilyPond 1.6.2. It assumes that you already know a little bit about LilyPond input (how to make an input file, how to create sheet music from that input file, etc). New users are encouraged to study the tutorial before reading this manual.

3.1 Note entry

Notes constitute the most basic elements of LilyPond input, but they do not form valid input on their own without a `\score` block. However, for the sake of brevity and simplicity we will generally omit `\score` blocks and `\paper` declarations in this manual.

3.1.1 Notes

A note is printed by specifying its pitch and then its duration.

```
cis'4 d'8 e'16 c'16
```



3.1.2 Pitches

The verbose syntax for pitch specification is

```
\pitch scmpitch
```

where *scmpitch* is a pitch scheme object.

In Note and Chord mode, pitches may be designated by names. The default names are the Dutch note names. The notes are specified by the letters **a** through **g**, while the octave is formed with notes ranging from **c** to **b**. The pitch **c** is an octave below middle C and the letters span the octave above that C. Here's an example which should make things more clear:

```
\clef bass
```

```
a,4 b, c d e f g a b c' d' e' \clef treble f' g' a' b' c''
```



In Dutch, a sharp is formed by adding **-is** to the end of a pitch name and a flat is formed by adding **-es**. Double sharps and double flats are obtained by adding **-isis** or **-eses**. **aes** and **ees** are contracted to **as** and **es** in Dutch, but both forms are accepted.

LilyPond has predefined sets of note names for various other languages. To use them, simply include the language specific init file. For example: `\include "english.ly"`. The available language files and the note names they define are:

	Note Names							sharp	flat
nederlands.ly	c	d	e	f	g	a	bes b	-is	-es
english.ly	c	d	e	f	g	a	bf b	-s/-sharp	-f/-flat
deutsch.ly	c	d	e	f	g	a	b h	-is	-es
norsk.ly	c	d	e	f	g	a	b h	-iss/-is	-ess/-es
svenska.ly	c	d	e	f	g	a	b h	-iss	-ess
italiano.ly	do	re	mi	fa	sol	la	sib si	-d	-b
catalan.ly	do	re	mi	fa	sol	la	sib si	-d/-s	-b
espanol.ly	do	re	mi	fa	sol	la	sib si	-s	-b

The optional octave specification takes the form of a series of single quote (‘’) characters or a series of comma (‘,’) characters. Each ’ raises the pitch by one octave; each , lowers the pitch by an octave.

c’ c’’ es’ g’ as’ gisis’ ais’



3.1.3 Chromatic alterations

Normally accidentals are printed automatically, but you may also print them manually. A reminder accidental can be forced by adding an exclamation mark ! after the pitch. A cautionary accidental (an accidental within parentheses) can be obtained by adding the question mark ? after the pitch.

cis’ cis’ cis’! cis’?

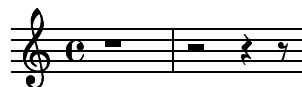


The automatic production of accidentals can be tuned in many ways. For more information, refer to Section 3.6 [Accidentals], page 63.

3.1.4 Rests

Rests are entered like notes, with a “note name” of ‘r’:

r1 r2 r4 r8



Whole bar rests, centered in middle of the bar, are specified using R (capital R); see Section 3.14.5 [Multi measure rests], page 94. See also internals document, **Rest**.

For some music, you may wish to explicitly specify the rest’s vertical position. This can be achieved by entering a note with the `\rest` keyword appended. Rest collision testing will leave these rests alone.

```
a'4\rest d'4\rest
```



3.1.5 Skips

An invisible rest (also called a ‘skip’) can be entered like a note with note name ‘s’ or with `\skip duration`:

```
a2 s4 a4 \skip 1 a4
```



In Lyrics mode, invisible rests are entered using either ‘" ’ or ‘_’:

```
<
  \context Lyrics \lyrics { lah2 di4 " " dah2 _4 di }
  \notes\relative c'' { a2 a4 a a2 a4 a }
>
      lah  di          dah      di
```



Note that the `s` syntax is only available in Note mode and Chord mode. In other situations, you should use the `\skip` command, which will work outside of those two modes:

```
\score {
  \context Staff <
    { \time 4/8 \skip 2 \time 4/4 }
    \notes\relative c'' { a2 a1 }
  >
}
```



The skip command is merely an empty musical placeholder. It doesn’t produce any output, not even transparent output.

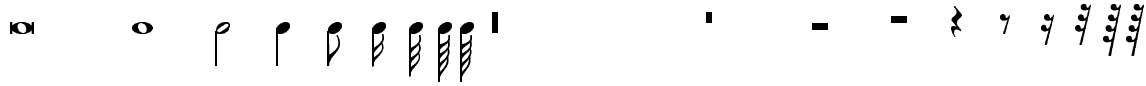
3.1.6 Durations

In Note, Chord, and Lyrics mode, durations are designated by numbers and dots: durations are entered as their reciprocal values. For example, a quarter note is entered using a 4 (since it’s a 1/4 note), while a half note is entered using a 2 (since it’s a 1/2 note). For notes longer than a whole you must use identifiers.

```

c'\breve
c'1 c'2 c'4 c'8 c'16 c'32 c'64 c'64
r\longa r\breve
r1 r2 r4 r8 r16 r32 r64 r64

```



If the duration is omitted then it is set to the previously entered duration. At the start of parsing, a quarter note is assumed. The duration can be followed by dots (‘.’) in order to obtain dotted note lengths:

```

a' b' c''8 b' a'4 a'4. b'4.. c'8.

```



You can alter the length of duration by a fraction N/M appending ‘ $*N/M$ ’ (or ‘ $*N$ ’ if $M=1$). This won’t affect the appearance of the notes or rests produced.

```

a'2*2 b'4*2 a'8*4 a'4*3/2 gis'4*3/2 a'4*3/2 a'4

```



Durations can also be produced through GUILF extension mechanism.

```

c'\duration #(make-duration 2 1)

```



BUGS

Dot placement for chords is not perfect. In some cases, dots overlap:



3.1.7 Ties

A tie connects two adjacent note heads of the same pitch. The tie in effect extends the length of a note. Ties should not be confused with slurs, which indicate articulation, or phrasing slurs, which indicate musical phrasing. A tie is entered using the tilde symbol ‘~’.

```

e' ~ e' <c' e' g'> ~ <c' e' g'>

```



When a tie is applied to a chord, all note heads (whose pitches match) are connected. If you try to tie together chords that have no common pitches, no ties will be created.

If you want less ties created for a chord, you can set `Voice.sparseTies` to true. In this case, a single tie is used for every tied chord.

```
\property Voice.sparseTies = ##t
<c' e' g'> ~ <c' e' g'>
```



In its meaning a tie is just a way of extending a note duration, similar to the augmentation dot: the following example are two ways of notating exactly the same concept.



If you need to tie notes over bars, it may be easier to use Section 3.1.8 [Automatic note splitting], page 49.

See also internals document, `Tie`.

BUGS

At present, the tie is represented as a separate event, temporally located in between the notes. Tying only a subset of the note heads of a chord is not supported in a simple way. It can be achieved by moving the tie-engraver into the Thread context and turning on and off ties per Thread.

Switching staves when a tie is active will not work.

3.1.8 Automatic note splitting

LilyPond can automatically converting long notes to tied notes. This is done by replacing the `Note_heads_engraver` by the `Completion_heads_engraver`.

```
\score{
  \notes\relative c'{ \time 2/4
    c2. c8 d4 e f g a b c8 c2 b4 a g16 f4 e d c8. c2
  }
  \paper{ \translator{
    \ThreadContext
    \remove "Note_heads_engraver"
    \consists "Completion_heads_engraver"
  } } }
```



This engraver splits all running notes at the bar line, and inserts ties. One of its uses is to debug complex scores: if the measures are not entirely filled, then the ties exactly show how much each measure is off.

BUGS

Not all durations (especially those containing tuplets) can be represented exactly; the engraver will not insert tuplets.

3.1.9 Tuplets

Tuplets are made out of a music expression by multiplying all durations with a fraction.

```
\times fraction musicexpr
```

The duration of *musicexpr* will be multiplied by the fraction. In the sheet music, the fraction's denominator will be printed over the notes, optionally with a bracket. The most common tuplet is the triplet in which 3 notes have the length of 2, so the notes are 2/3 of their written length:

```
g'4 \times 2/3 {c'4 c' c'} d'4 d'4
```



The property `tupletSpannerDuration` specifies how long each bracket should last. With this, you can make lots of tuplets while typing `\times` only once, saving you lots of typing.

```
\property Voice.tupletSpannerDuration = #(make-moment 1 4)
\times 2/3 { c'8 c c c c c }
```



The format of the number is determined by the property `tupletNumberFormatFunction`. The default prints only the denominator, but if you set it to the Scheme function `fraction-tuplet-formatter`, Lilypond will print *num:den* instead.

See also internals document, `TupletBracket`.

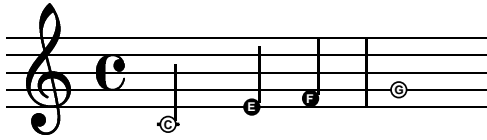
BUGS

Nested tuplets are not formatted automatically. In this case, outer tuplet brackets should be moved automatically.

3.1.10 Easy Notation note heads

A entirely different type of note head is the "easyplay" note head: a note head that includes a note name. It is used in some publications by Hal-Leonard Inc. music publishers.

```
\score {
  \notes { c'2 e'4 f' | g'1 }
  \paper { \translator { \EasyNotation } }
}
```

Note that `EasyNotation` overrides a `Score` context. You probably will want to print it with magnification or a large font size to make it more readable. To print with magnification, you must create a dvi (with ‘`ly2dvi`’) and then enlarge it with something like ‘`dvips -x 2000 file.dvi`’. See ‘`man dvips`’ for details. To print with a larger font, see Section 3.17.3 [Font Size], page 114.

If you view the result with Xdvi, then staff lines will show through the letters. Printing the PostScript file obtained with `ly2dvi` does produce the correct result.

3.2 Easier music entry

When entering music with LilyPond, it is easy to introduce errors. This section deals with tricks and features that help you enter music, and find and correct mistakes.

3.2.1 Graphical interfaces

One way to avoid entering notes using the keyboard is to use a graphical user interface. The following programs are known to have a lilypond export option:

- Denemo was once intended as a LilyPond graphical user interface. It runs on Gnome/GTK.

<http://denemo.sourceforge.net/>

- Noteedit, a graphical score editor that runs under KDE/Qt.

<http://rnvs.informatik.tu-chemnitz.de/~jan/noteedit/noteedit.html>

- RoseGarden was once the inspiration for naming LilyPond. Nowadays it has been rewritten from scratch and supports LilyPond export as of version 0.1.6.

<http://rosegarden.sf.net/>

Another option is to enter the music using your favorite MIDI sequencer, and then import it using `midi2ly`. `midi2ly` is described in Section 7.2 [Invoking `midi2ly`], page 142.

3.2.2 Relative octaves

Octaves are specified by adding ‘`’` and ‘`,`’ to pitch names. When you copy existing music, it is easy to accidentally put a pitch in the wrong octave and hard to find such an error. To prevent these errors, LilyPond features octave entry.

```
\relative startpitch musicexpr
```

The octave of notes that appear in *musicexpr* are calculated as follows: If no octave changing marks are used, the basic interval between this and the last note is always taken to be a fourth or less (This distance is determined without regarding alterations; a `fisis` following a `ceses` will be put above the `ceses`)

The octave changing marks ' and , can be added to raise or lower the pitch by an extra octave. Upon entering relative mode, an absolute starting pitch must be specified that will act as the predecessor of the first note of *musicexpr*.

Entering music that changes octave frequently is easy in relative mode.

```
\relative c'' {
  b c d c b c bes a
}
```



And octave changing marks are used for intervals greater than a fourth.

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



If the preceding item is a chord, the first note of the chord is used to determine the first note of the next chord. However, other notes within the second chord are determined by looking at the immediately preceding note.

```
\relative c' {
  c <c e g>
  <c' e g>
  <c, e' g>
}
```



The pitch after the `\relative` contains a note name. To parse the pitch as a note name, you have to be in note mode, so there must be a surrounding `\notes` keyword (which is not shown here).

The relative conversion will not affect `\transpose`, `\chords` or `\relative` sections in its argument. If you want to use relative within transposed music, you must place an additional `\relative` inside the `\transpose`.

3.2.3 Bar check

Whenever a bar check is encountered during interpretation, a warning message is issued if it doesn't fall at a measure boundary. This can help you find errors in the input. Depending on the value of `barCheckSynchronize`, the beginning of the measure will be relocated, so this can also be used to shorten measures.

A bar check is entered using the bar symbol, |:

```
\time 3/4 c2 e4 | g2.
```

Failed bar checks are most often caused by entering incorrect durations. Incorrect durations often completely garble up the score, especially if it is polyphonic, so you should start correcting the score by scanning for failed bar checks and incorrect durations. To speed up this process, you can use `skipTypesetting` (See Section 3.2.5 [Skipping corrected music], page 54)).

3.2.4 Point and click

Point and click lets you find notes in the input by clicking on them in the Xdvi window. This makes it very easy to find input that causes some error in the sheet music.

To use it, you need the following software

- A dvi viewer that supports src specials.
 - Xdvi, version 22.36 or newer. Available from <ftp://ftp.math.berkeley.edu/pub/Software/TeX/xdvi.tar.gz>.
Note that most T_EX distributions ship with xdvik, which is always a few versions behind the official Xdvi. To find out which xdvi you are running, try `xdvi -version` or `xdvi.bin -version`.
 - KDVI. A dvi viewer for KDE. You need KDVI from KDE 3.0 or newer. Enable option *Inverse search* in the menu *Settings*.
- An editor with a client/server interface (or a lightweight GUI editor).
 - Emacs. Emacs is an extensible text-editor. It is available from <http://www.gnu.org/software/emacs/>. You need version 21 to use column location.
LilyPond also comes with support files for emacs: `lilypond-mode` for emacs provides indentation, syntax coloring and handy compile short-cuts. If `lilypond-mode` is not installed on your platform, then refer to the installation instructions for more information
 - XEmacs. Xemacs is very similar to emacs.
 - NEdit. NEdit runs under Windows, and Unix. It is available from <http://www.nedit.org>.
 - GVim. GVim is a GUI variant of VIM, the popular VI clone. It is available from <http://www.vim.org>.

Xdvi must be configured to find the T_EX fonts and music fonts. Refer to the Xdvi documentation for more information.

To use point-and-click, add one of these lines to the top of your `.ly` file.

```
 #(set-point-and-click! 'line)
```

When viewing, Control-Mousebutton 1 will take you to the originating spot in the `'ly` file. Control-Mousebutton 2 will show all clickable boxes.

If you correct large files with point-and-click, be sure to start correcting at the end of the file. When you start at the top, and insert one line, all following locations will be off by a line.

For using point-and-click with emacs, add the following In your emacs startup file (usually ‘~/*.emacs*’),

```
(server-start)
```

Make sure that the environment variable *XEDITOR* is set to

```
emacsclient --no-wait +%l %f
```

If you use xemacs instead of emacs, you use (*gnuserve-start*) in your ‘*.emacs*’, and set *XEDITOR* to *gnuclient -q +%l %f*

For using Vim, set *XEDITOR* to *gvim --remote +%l %f*, or use this argument with *xdvi*’s *-editor* option. For using NEdit, set *XEDITOR* to *nc -noask +%l %f*, or use this argument with *xdvi*’s *-editor* option.

If can also make your editor jump to the exact location of the note you clicked. This is only supported on Emacs. Users of version 20 must apply the patch ‘*emacsclient.patch*’. Users of version 21 must apply ‘*server.el.patch*’ (version 21.2 and earlier). At the top of the *ly* file, replace the *set!* line with the following line,

```
#{set-point-and-click! 'line-column)
```

and set *XEDITOR* to *emacsclient --no-wait +%l:%c %f*.

BUGS

When you convert the *T_EX* file to PostScript using *dvips*, it will complain about not finding *src:X:Y* files. These complaints are harmless, and can be ignored.

3.2.5 Skipping corrected music

The property *Score.skipTypesetting* can be used to switch on and off typesetting completely during the interpretation phase. When typesetting is switched off, the music is processed much more quickly. You can use this to skip over the parts of a score that you have already checked for errors.

```
\relative c'' { c8 d
\property Score.skipTypesetting = ##t
  e f g a g c, f e d
\property Score.skipTypesetting = ##f
c d b bes a g c2 }
```



3.3 Staff notation

This section deals with music notation that occurs on staff level, such as keys, clefs and time signatures.

3.3.1 Staff symbol

The lines of the staff symbol are formed by the `StaffSymbol` object. This object is created at the moment that their context is created. You can not change the appearance of the staff symbol by using `\override` or `\set`. At the moment that `\property Staff` is interpreted, a `Staff` context is made, and the `StaffSymbol` is created before any `\override` is effective. You can deal with this either overriding properties in a `\translator` definition, or by using `\outputproperty`.

BUGS

If you end a staff half way a piece, the staff symbol may not end exactly on the barline.

3.3.2 Key signature

Setting or changing the key signature is done with the `\key` command.

`\key pitch type`

Here, *type* should be `\major` or `\minor` to get *pitch*-major or *pitch*-minor, respectively. The standard mode names `\ionian`, `\locrian`, `\aeolian`, `\mixolydian`, `\lydian`, `\phrygian`, and `\dorian` are also defined.

This command sets the context property `Staff.keySignature`. Non-standard key signatures can be specified by setting this property directly.

The printed signature is a `KeySignature` object, typically created in `Staff` context.

3.3.3 Clef

The clef can be set or changed with the `\clef` command:

`\key f\major c''2 \clef alto g'2`



Supported clef-names include

`treble`, `violin`, `G`, `G2`

G clef on 2nd line

`alto`, `C` C clef on 3rd line

`tenor` C clef on 4th line

`bass`, `F` F clef on 4th line

`french` G clef on 1st line, so-called French violin clef

`soprano` C clef on 1st line

`mezzosoprano`

C clef on 2nd line

`baritone` C clef on 5th line

```

varbaritone      F clef on 3rd line
subbass          F clef on 5th line
percussion
percussion clef

```

By adding `_8` or `^8` to the clef name, the clef is transposed one octave down or up, respectively. Note that you have to enclose *clefname* in quotes if you use underscores or digits in the name. For example,

```
\clef "G_8"
```

The object for this symbol is `Clef`.

This command is equivalent to setting `clefGlyph`, `clefPosition` (which controls the Y position of the clef), `centralCPosition` and `clefOctavation`. A clef is created when any of these properties are changed.

3.3.4 Time signature

The time signature is set or changed by the `\time` command.

```
\time 2/4 c'2 \time 3/4 c'2.
```



The actual symbol that's printed can be customized with the `style` property. Setting it to `#'()` uses fraction style for 4/4 and 2/2 time.

The object for this symbol is `TimeSignature`. There are many more options for its layout. They are selected through the `style` object property. See `'input/test/time.ly'` for more examples.

This command sets the property `timeSignatureFraction`, `beatLength` and `measureLength` in the `Timing` context, which is normally aliased to `Score`. The property `timeSignatureFraction` determine where bar lines should be inserted, and how automatic beams should be generated. Changing the value of `timeSignatureFraction` also causes a time signature symbol to be printed.

3.3.5 Partial

Partial measures, for example in upbeats, are entered using the `\partial` command:

```
\partial 4* 5/16 c'16 c4 f16 a'2. ~ a'8. a'16 | g'1
```



The syntax for this command is

`\partial duration`

This is internally translated into

`\property Timing.measurePosition = -length of duration`

The property `measurePosition` contains a rational number indicating how much of the measure has passed at this point.

3.3.6 Unmetered music

Bar lines and bar numbers are calculated automatically. For unmetered music (e.g. cadenzas), this is not desirable. The commands `\cadenzaOn` and `\cadenzaOff` can be used to switch off the timing information:

```
c'2.  
\cadenzaOn  
c2  
\cadenzaOff  
c4 c4 c4
```

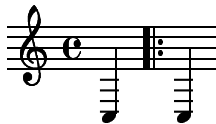


The property `Score.timing` can be used to switch off this automatic timing

3.3.7 Bar lines

Bar lines are inserted automatically, but if you need a special type of barline, you can force one using the `\bar` command:

```
c4 \bar "|:" c4
```



The following bar types are available

```
c4  
\bar "|" c  
\bar "" c  
\bar "|:" c  
\bar "||" c  
\bar ":||" c  
\bar ".|" c  
\bar ".|." c  
\bar "|."
```



You are encouraged to use `\repeat` for repetitions. See Section 3.9 [Repeats], page 74.

In scores with many staves, the barlines are automatically placed at top level, and they are connected between different staves of a `StaffGroup`:

```
< \context StaffGroup <
  \context Staff = up { e'4 d'
    \bar "||"
    f' e' }
  \context Staff = down { \clef bass c4 g e g } >
\context Staff = pedal { \clef bass c2 c2 } >
```



The objects that are created at `Staff` level. The name is `BarLine`.

The command `\bar bartype` is a short cut for doing `\property Score.whichBar = bartype`. Whenever `whichBar` is set to a string, a bar line of that type is created. `whichBar` is usually set automatically: at the start of a measure it is set to `defaultBarType`. The contents of `repeatCommands` is used to override default measure bars.

`whichBar` can also be set directly, using `\property` or `\bar`. These settings take precedence over the automatic `whichBar` settings.

3.4 Polyphony

The easiest way to enter such fragments with more than one voice on a staff is to split chords using the separator `\\`. You can use it for small, short-lived voices (make a chord of voices) or for single chords:

```
\context Voice = VA \relative c'' {
  c4 < { f d e } \\ { b c2 } > c4 < g' \\ b, \\ f \\ d >
}
```



The separator causes `Voice` contexts to be instantiated, bearing the names "1", "2", etc.

Sometimes, it is necessary to instantiate these contexts by hand: For Instantiate a separate `Voice` context for each part, and use `\voiceOne`, up to `\voiceFour` to assign a stem directions and horizontal shift for each part.


```

\relative c''
\context Staff < \context Voice = VA { \voiceOne cis2 b  }
  \context Voice = VB { \voiceThree b4 ais ~ ais4 gis4 }
  \context Voice = VC { \voiceTwo fis4~ fis4 f ~ f  } >

```



The identifiers `\voiceOne` to `\voiceFour` set directions ties, slurs and stems, and set shift directions.

If you want more than four voices, you can also manually set horizontal shifts and stem directions, as is shown in the following example:

```

\context Staff \notes\relative c''<
  \context Voice=one {
    \shiftOff \stemUp e4
  }
  \context Voice=two {
    \shiftOn \stemUp cis
  }
  \context Voice=three {
    \shiftOnn \stemUp ais
  }
  \context Voice=four {
    \shiftOnnn \stemUp fis
  }
>

```



Normally, note heads with a different number of dots are not merged, but if you set the object property `merge-differently-dotted`, they are:

```

\context Voice < {
  g'8 g'8
  \property Staff.NoteCollision \override
    #'merge-differently-dotted = ##t
  g'8 g'8
} \ { [g'8. f16] [g'8. f'16] }
>

```



Similarly, you can merge half note heads with eighth notes, by setting `merge-differently-headed`:

```

\context Voice < {

```

```

c8 c4.
\property Staff.NoteCollision
  \override #'merge-differently-headed = ##t
c8 c4. } \\ { c2 c2 } >

```



LilyPond also vertically shifts rests that are opposite of a stem.

```
\context Voice < c''4 \\ r4 >
```



See also `NoteCollision` and `RestCollision`

BUGS

Resolving collisions is a very intricate subject, and LilyPond only handles a few situations. When it can not cope, you are advised to use `force-hshift` of the `NoteColumn` object and pitched rests to override typesetting decisions.

3.5 Beaming

Beams are used to group short notes into chunks that are aligned with the metrum. They are inserted automatically in most cases.

```
\time 2/4 c8 c c c \time 6/8 c c c c8. c16 c8
```



If you're not satisfied with the automatic beaming, you can enter the beams explicitly. If you have beaming patterns that differ from the defaults, you can also set the patterns for automatic beamer.

See also `Beam`.

3.5.1 Manual beams

In some cases it may be necessary to override LilyPond's automatic beaming algorithm. For example, the auto beamer will not beam over rests or bar lines, If you want that, specify the begin and end point manually using a `[` before the first beamed note and a `]` after the last note:

```

\context Staff {
  r4 [r8 g' a r8] r8 [g | a] r8
}

```



Normally, beaming patterns within a beam are determined automatically. When this mechanism fouls up, the properties `Voice.stemLeftBeamCount` and `Voice.stemRightBeamCount` can be used to control the beam subdivision on a stem. If you set either property, its value will be used only once, and then it is erased.

```
\context Staff {
  [f8 r16 f g a]
  [f8 r16 \property Voice.stemLeftBeamCount = #1 f g a]
}
```



The property `subdivideBeams` can be set in order to subdivide all 16th or shorter beams at beat positions. This accomplishes the same effect as twiddling with `stemLeftBeamCount` and `stemRightBeamCount`, but it take less typing.

```
[c16 c c c c c c c]
\property Voice.subdivideBeams = ##t
[c16 c c c c c c c]
[c32 c c c c c c c c c c c c c c c]
\property Score.beatLength = #(make-moment 1 8)
[c32 c c c c c c c c c c c c c c c]
```



Knee beams are inserted automatically, when a large gap between two adjacent beamed notes is detected. This behavior can be tuned through the object property `auto-knee-gap`.

BUGS

Auto knee beams can not be used together with *hara kiri* staves.

[TODO from bugs]

The Automatic beamer does not put **unfinished** beams on the last notes of a score.

Formatting of ties is a difficult subject. LilyPond often does not give optimal results.

3.5.2 Setting automatic beam behavior

In normal time signatures, automatic beams can start on any note but can only end in a few positions within the measure: beams can end on a beat, or at durations specified by the properties in `Voice.autoBeamSettings`. The defaults for `autoBeamSettings` are defined in `'scm/auto-beam.scm'`.

The value of `autoBeamSettings` is changed using `\override` and unset using `\revert`:

```
\property Voice.autoBeamSettings \override #'(BE P Q N M) = dur
\property Voice.autoBeamSettings \revert #'(BE P Q N M)
```

Here, *BE* is the symbol `begin` or `end`. It determines whether the rule applies to begin or end-points. The quantity P/Q refers to the length of the beamed notes (and ‘* *’ designates notes of any length), N/M refers to a time signature (wildcards, ‘* *’ may be entered to designate all time signatures).

For example, if you want automatic beams to end on every quarter note, you can use the following:

```
\property Voice.autoBeamSettings \override
  #'(end * * * *) = #(make-moment 1 4)
```

Since the duration of a quarter note is 1/4 of a whole note, it is entered as `(make-moment 1 4)`.

The same syntax can be used to specify beam starting points. In this example, automatic beams can only end on a dotted quarter note.

```
\property Voice.autoBeamSettings \override
  #'(end * * * *) = #(make-moment 3 8)
```

In 4/4 time signature, this means that automatic beams could end only on 3/8 and on the fourth beat of the measure (after 3/4, that is 2 times 3/8 has passed within the measure).

You can also restrict rules to specific time signatures. A rule that should only be applied in N/M time signature is formed by replacing the second asterisks by N and M . For example, a rule for 6/8 time exclusively looks like

```
\property Voice.autoBeamSettings \override
  #'(begin * * 6 8) = ...
```

If you want a rule to apply to certain types of beams, you can use the first pair of asterisks. Beams are classified according to the shortest note they contain. For a beam ending rule that only applies to beams with 32nd notes (and no shorter notes), you would use `(end 1 32 * *)`.

If a score ends while an automatic beam has not been ended and is still accepting notes, this last beam will not be typeset at all.

For melodies that have lyrics, you may want to switch off automatic beaming. This is done by setting `Voice.autoBeaming` to `#f`.

BUGS

It is not possible to specify beaming parameters for beams with mixed durations, that differ from the beaming parameters of all separate durations, i.e., you’ll have to specify manual beams to get:



It is not possible to specify beaming parameters that act differently in different parts of a measure. This means that it is not possible to use automatic beaming in irregular meters such as 5/8.

3.6 Accidentals

This section describes how to change the way that LilyPond automatically inserts accidentals before the running notes.

3.6.1 Using the predefined accidental macros

The constructs for describing the accidental typesetting rules are quite hairy, so non-experts should stick to the macros defined in ‘ly/property-init.ly’.

The macros operate on the “Current” context (see Section 4.1.3 [Context properties], page 120). This means that the macros should normally be invoked right after the creation of the context in which the accidental typesetting described by the macro is to take effect. I.e. if you want to use piano-accidentals in a pianostaff then you issue `\pianoAccidentals` first thing after the creation of the piano staff:

```
\score {
  \notes \relative c'' <
    \context Staff = sa { cis4 d e2 }
    \context GrandStaff <
      \pianoAccidentals
      \context Staff = sb { cis4 d e2 }
      \context Staff = sc { es2 c }
    >
    \context Staff = sd { es2 c }
  >
}
```



The macros are:

`\defaultAccidentals`

This is the default typesetting behaviour. It should correspond to 18th century common practice: Accidentals are remembered to the end of the measure in which they occur and only on their own octave.

`\voiceAccidentals`

The normal behaviour is to remember the accidentals on Staff-level. This macro, however, typesets accidentals individually for each voice. Apart from that the rule is similar to `\defaultAccidentals`.

Warning: This leads to some weird and often unwanted results because accidentals from one voice DO NOT get cancelled in other voices:

```
\context Staff <
  \voiceAccidentals
  \context Voice=va { \voiceOne es g }
  \context Voice=vb { \voiceTwo c, e }
>
```



Hence you should only use `\voiceAccidentals` if the voices are to be read solely by individual musicians. If the staff should be readable also by one musician/conductor then you should use `\modernVoiceAccidentals` or `\modernVoiceCautionaries` instead.

`\modernAccidentals`

This rule should correspond to the common practice in the 20th century. The rule is a bit more complex than `\defaultAccidentals`: You get all the same accidentals, but temporary accidentals also get cancelled in other octaves. Furthermore, in the same octave, they also get cancelled in the following measure:

```
\modernAccidentals
cis' c'' cis'2 | c'' c'
```



`\modernCautionaries`

This rule is similar to `\modernAccidentals`, but the “extra” accidentals (the ones not typeset by `\defaultAccidentals`) are typeset as cautionary accidentals (i.e. in reduced size):

```
\modernCautionaries
cis' c'' cis'2 | c'' c'
```



`\modernVoiceAccidentals`

Multivoice accidentals to be read both by musicians playing one voice and musicians playing all voices.

Accidentals are typeset for each voice, but they ARE cancelled across voices in the same `Staff`.

`\modernVoiceCautionaries`

The same as `\modernVoiceAccidentals`, but with the extra accidentals (the ones not typeset by `\voiceAccidentals`) typeset as cautionaries. Notice that even though all accidentals typeset by `\defaultAccidentals` ARE typeset by this macro then some of them are typeset as cautionaries.

\pianoAccidentals

20th century practice for piano notation. Very similar to **\modernAccidentals** but accidentals also get cancelled across the staves in the same **GrandStaff** or **PianoStaff**.

\pianoCautionaries

As **\pianoAccidentals** but with the extra accidentals typeset as cautionaries.

\noResetKey

Same as **\defaultAccidentals** but with accidentals lasting “forever” and not only until the next measure:

```
\noResetKey
c1 cis cis c
```

**\forgetAccidentals**

This is sort of the opposite of **\noResetKey**: Accidentals are not remembered at all - and hence all accidentals are typeset relative to the key signature, regardless of what was before in the music:

```
\forgetAccidentals
\key d\major c4 c cis cis d d dis dis
```



3.6.2 Defining your own accidental typesettings

This section must be considered gurus-only, and hence it must be sufficient with a short description of the system and a reference to the internal documentation.

The idea of the algorithm is to try several different rules and then use the rule that gives the highest number of accidentals. Each rule consists of

Context: In which context is the rule applied. I.e. if context is **Score** then all staves share accidentals, and if context is **Staff** then all voices in the same staff share accidentals, but staves don't - like normally.

Octavation:

Whether the accidental changes all octaves or only the current octave.

Lazyness: Over how many barlines the accidental lasts. If lazyness is **-1** then the accidental is forget immediately, and if lazyness is **#t** then the accidental lasts forever.

As described in the internal documentation of **Accidental_engraver**, the properties **autoAccidentals** and **autoCautionaries** contain lists of rule descriptions. Notice that the contexts must be listed from in to out - that is **Thread** before **Voice**, **Voice** before

Staff, etc. see the macros in ‘ly/property-init.ly’ for examples of how the properties are set.

BUGS

Currently the simultaneous notes are considered to be entered in sequential mode. This means that in a chord the accidentals are typeset as if the notes in the chord happened one at a time - in the order in which they appear in the input file.

Of course this is only a problem when you have simultaneous notes which accidentals should depend on each other. Notice that the problem only occurs when using non-default accidentals - as the default accidentals only depend on other accidentals on the same staff and same pitch and hence cannot depend on other simultaneous notes.

This example shows two examples of the same music giving different accidentals depending on the order in which the notes occur in the input file:

```
\property Staff.autoAccidentals = #'( Staff (any-octave . 0) )
cis'4 <c'' c'> r2 | cis'4 <c' c''> r2 | <cis' c''> r | <c'' cis'> r |
```



The only solution is to manually insert the problematic accidentals using ! and ?.

3.7 Expressive marks

3.7.1 Slurs

A slur indicates that notes are to be played bound or *legato*. They are entered using parentheses:

```
f'()g'()a' [a'8 b'[] a'4 g'2 )f'4
```



See also internals document, **Slur**.

Slurs avoid crossing stems, and are generally attached to note heads. However, in some situations with beams, slurs may be attached to stem ends. If you want to override this layout you can do this through the object property **attachment** of **Slur** in **Voice** context. Its value is a pair of symbols, specifying the attachment type of the left and right end points.

```
\slurUp
\property Voice.Stem \set #'length = #5.5
g'8(g)g4
\property Voice.Slur \set #'attachment = #'(stem . stem)
g8(g)g4
```




If a slur would strike through a stem or beam, the slur will be moved away upward or downward. If this happens, attaching the slur to the stems might look better:

```
\stemUp \slurUp
d32( d'4 )d8..
\property Voice.Slur \set #'attachment = #'(stem . stem)
d,32( d'4 )d8..
```



BUGS

Producing nice slurs is a difficult problem, and LilyPond currently uses a simple, empiric method to produce slurs. In some cases, the results of this method are ugly.

3.7.2 Phrasing slurs

A phrasing slur (or phrasing mark) connects chords and is used to indicate a musical sentence. It is started using `\(` and `\)` respectively.

```
\time 6/4 c' \( d () e f () e \) d
```



Typographically, the phrasing slur behaves almost exactly like a normal slur. See also internals document, `PhrasingSlur`. But although they behave similarly to normal slurs, phrasing slurs count as different objects. A `\slurUp` will have no effect on a phrasing slur; instead, you should use `\phrasingSlurUp`, `\phrasingSlurDown`, and `\phrasingSlurBoth`.

Note that the commands `\slurUp`, `\slurDown`, and `\slurBoth` will only affect normal slurs and not phrasing slurs.

3.7.3 Breath marks

Breath marks are entered using `\breathe`. See also internals document, `BreathingSign`.



3.7.4 Tempo

Metronome settings can be entered as follows:

```
\tempo duration = perminute
```

For example, `\tempo 4 = 76` requests output with 76 quarter notes per minute.

BUGS

The tempo setting is not printed, but is only used in the MIDI output. You can trick lily into producing a metronome mark, though. Details are in Section 3.16.4 [Text markup], page 109.

3.7.5 Text spanners

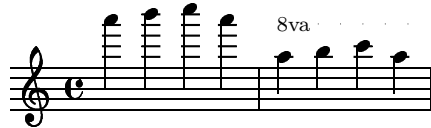
Some textual indications, e.g. *rallentando* or *accelerando*, often extend over many measures. This is indicated by following the text with a dotted line. You can create such texts using text spanners. The syntax is as follows:

```
\spanrequest \start "text"
\spanrequest \stop "text"
```

LilyPond will respond by creating a `TextSpanner` object (typically in `Voice` context). The string to be printed, as well as the style is set through object properties.

An application—or rather, a hack—is to fake octavation indications.

```
\relative c' { a''' b c a
  \property Voice.TextSpanner \set #'type = #'dotted-line
  \property Voice.TextSpanner \set #'edge-height = #'(0 . 1.5)
  \property Voice.TextSpanner \set #'edge-text = #'("8va " . "")
  \property Staff.centralCPosition = #-13
  a\spanrequest \start "text" b c a \spanrequest \stop "text" }
```



3.8 Ornaments

3.8.1 Articulations

A variety of symbols can appear above and below notes to indicate different characteristics of the performance. They are added to a note by adding a dash and the character signifying the articulation. They are demonstrated here.

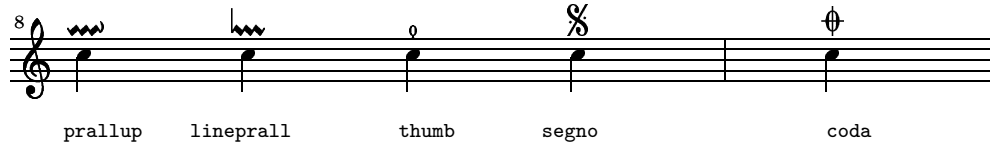
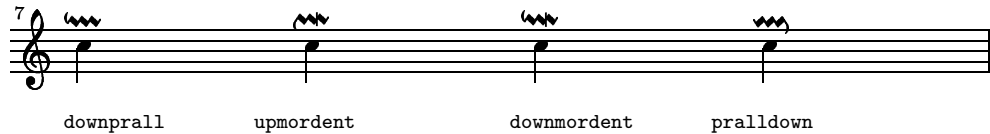
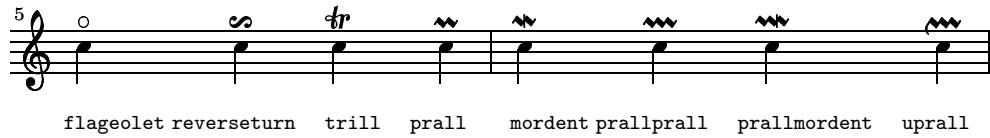
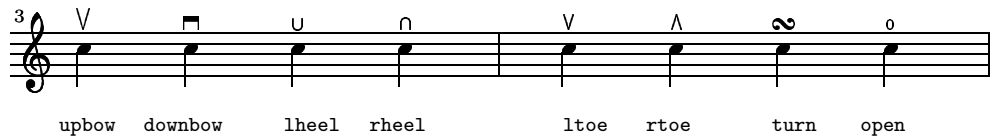
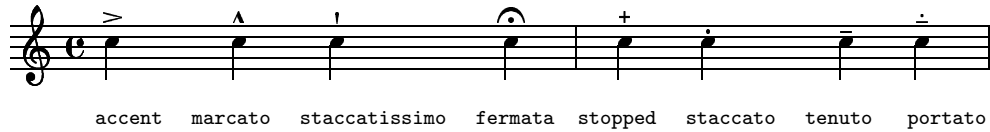


The script is automatically placed, but if you need to force directions, you can use `_` to force them down, or `^` to put them up:

c''4^^ c''4_~

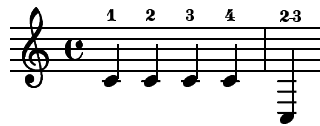


Other symbols can be added using the syntax *note-\name*. Again, they can be forced up or down using *^* and *_*.



Fingering instructions can also be entered in this shorthand. For finger changes, use markup texts:

c'4-1 c'4-2 c'4-3 c'4-4
c^#'(finger "2-3")



See also [internals document, Script](#) and [internals document, Fingering](#).

BUGS

All of these note ornaments appear in the printed output but have no effect on the MIDI rendering of the music.

Unfortunately, there is no support for adding fingering instructions or ornaments to individual note heads. Some hacks exist, though. See ‘input/test/script-horizontal.ly’.

3.8.2 Text scripts

In addition, it is possible to place arbitrary strings of text or markup text (see Section 3.16.4 [Text markup], page 109) above or below notes by using a string: `c^"text"`.

By default, these indications do not influence the note spacing, but by using the command `\fatText`, the widths will be taken into account.

```
\relative c' {
  c4^"longtext" \fatText c4_"longlongtext" c4 }
```



It is possible to use \TeX commands in the strings, but this should be avoided because it makes it impossible for LilyPond to compute the exact length of the string, which may lead to collisions. Also, \TeX commands won't work with direct PostScript output.

Text scripts are created in form of `TextScript` objects, in `Voice` context.

Section 3.16.4 [Text markup], page 109 describes how to change the font or access special symbols in text scripts.

3.8.3 Grace notes

Grace notes are ornaments that are written out

```
c4 \grace c16 c4 \grace {
  [c16 d16] } c4
```



In normal notation, grace notes are supposed to take up no logical time in a measure. Such an idea is practical for normal notation, but is not strict enough to put it into a program. The model that LilyPond uses for grace notes internally is that all timing is done in two steps:

Every point in musical time consists of two rational numbers: one denotes the logical time, one denotes the grace timing. The above example is shown here with timing tuples.



The advantage of this approach is that you can use almost any lilypond construction together with grace notes, for example slurs and clef changes may appear halfway in between grace notes:

```
c4 \grace { [ c16 c, \clef bass c, b[] ] } )c4
```



The placement of these grace notes is synchronized between different staves, using this grace timing.

```
< \context Staff = SA { e4 \grace { c16 d e f } e4 }
  \context Staff = SB { c4 \grace { g8 b } c4 } >
```



Unbeamed eighth notes and shorter by default have a slash through the stem. This can be controlled with object property `stroke-style` of `Stem`. The change in formatting is accomplished by inserting `\startGraceMusic` before handling the grace notes, and `\stopGraceMusic` after finishing the grace notes. You can add to these definitions to globally change grace note formatting. The standard definitions are in `'ly/grace-init.ly'`.

Notice how the `\override` is carefully matched with a `\revert`.

```
\relative c'' \context Voice {
  \grace c8 c4 \grace { [c16 c16] } c4
  \grace {
    \property Voice.Stem \override #'stroke-style = #'()
    c16
    \property Voice.Stem \revert #'stroke-style
  } c4
}
```



If you want to end a note with a grace note, then the standard trick is to put the grace notes before a phantom “space note”, e.g.

```
\context Voice {
  < { d1^\trill ( }
    { s2 \grace { [c16 d] } } >
  )c4
}
```



A `\grace` section has some default values, and LilyPond will use those default values unless you specify otherwise inside the `\grace` section. For example, if you specify `\slurUp` *before* your `\grace` section, a slur which starts inside the `\grace` won't be forced up, even if the slur ends outside of the `\grace`. Note the difference between the first and second bars in this example:

```
\relative c'' \context Voice {
  \slurUp
  \grace {
    a4 ( )
  } a4 a4 ( ) a2
  \slurBoth

  \grace {
    \slurUp
    a4 ( )
  } a4 a4 ( ) a2
  \slurBoth
}
```



BUGS

Grace notes can not be used in the smallest size (`'paper11.ly'`).

Grace note synchronization can also lead to surprises. Staff notation, such as key signatures, barlines, etc. are also synchronized. Take care when you mix staves with grace notes and staves without.

```
< \context Staff = SA { e4 \bar "|:" \grace c16 d4 }
  \context Staff = SB { c4 \bar "|:" d4 } >
```



Grace sections should only be used within sequential music expressions. Nesting, juxtaposing, or ending sequential music with a grace section is not supported, and might produce crashes or other errors.

3.8.4 Glissando

A glissando line can be requested by attaching a `\glissando` to a note:

```
c'-\glissando c'
```



BUGS

Printing of an additional text (such as *gliss.*) must be done manually. See also internals document, *Glissando*.

3.8.5 Dynamics

Absolute dynamic marks are specified using an identifier after a note: `c4-\ff`. The available dynamic marks are: `\ppp`, `\pp`, `\p`, `\mp`, `\mf`, `\f`, `\ff`, `\fff`, `\fff`, `\fp`, `\sf`, `\sff`, `\sp`, `\spp`, `\sfz`, and `\rfz`.

```
c'\ppp c\pp c \p c\mp c\mf c\f c\ff c\fff
c2\sf c\rfz
```



A crescendo mark is started with `\cr` and terminated with `\rc` (the textual reverse of `cr`). A decrescendo mark is started with `\decr` and terminated with `\rced`. There are also shorthands for these marks. A crescendo can be started with `\<` and a decrescendo can be started with `\>`. Either one can be terminated with `\!`. Note that `\!` must go before the last note of the dynamic mark whereas `\rc` and `\rced` go after the last note. Because these marks are bound to notes, if you want several marks during one note, you have to use spacer notes.

```
c'' \< \! c'' d'' \decr e'' \rced
< f''1 { s4 s4 \< \! s4 \> \! s4 } >
```



You can also use a text saying *cresc.* instead of hairpins. Here is an example how to do it:

```
c4 \cresc c4 \endcresc c4
```



You can also supply your own texts:

```

\context Voice {
  \property Voice.crescendoText = "cresc. poco"
  \property Voice.crescendoSpanner = #'dashed-line
  a'2\mf\< a a \!a
}

```



Dynamics are objects of `DynamicText` and `Hairpin`. Vertical positioning of these symbols is handled by the `DynamicLineSpanner` object. If you want to adjust padding or vertical direction of the dynamics, you must set properties for the `DynamicLineSpanner` object. Predefined identifiers to set the vertical direction are `\dynamicUp` and `\dynamicDown`.

3.9 Repeats

To specify repeats, use the `\repeat` keyword. Since repeats should work differently when played or printed, there are a few different variants of repeats.

- unfold** Repeated music is fully written (played) out. Useful for MIDI output, and entering repetitive music.
- volta** This is the normal notation: Repeats are not written out, but alternative endings (voltas) are printed, left to right.
- fold** Alternative endings are written stacked. This has limited use but may be used to typeset two lines of lyrics in songs with repeats, see ‘input/star-spangled-banner.ly’.
- tremolo** Make tremolo beams.
- percent** Make beat or measure repeats. These look like percent signs.

3.9.1 Repeat syntax

The syntax for repeats is

```
\repeat variant repeatcount repeatbody
```

If you have alternative endings, you may add

```

\alternative { alternative1
               alternative2
               alternative3 ... }

```

where each *alternative* is a music expression.

Normal notation repeats are used like this:

```

c'1
\repeat volta 2 { c'4 d' e' f' }
\repeat volta 2 { f' e' d' c' }

```




With alternative endings:

```
c'1
\repeat volta 2 {c'4 d' e' f'}
\alternative { {d'2 d'} {f' f} }
```



Folded repeats look like this:

```
c'1
\repeat fold 2 {c'4 d' e' f'}
\alternative { {d'2 d'} {f' f} }
```



If you don't give enough alternatives for all of the repeats, then the first alternative is assumed to be repeated often enough to equal the specified number of repeats.

```
\context Staff {
  \relative c' {
    \partial 4
    \repeat volta 4 { e | c2 d2 | e2 f2 | }
    \alternative { { g4 g g } { a | a a a | b2. } }
  }
}
```



3.9.2 Repeats and MIDI

For instructions on how to unfold repeats for MIDI output, see the example file ‘input/test/unfold-all-repeats.ly’.

BUGS

Notice that timing information is not remembered at the start of an alternative, so you have to reset timing information after a repeat, e.g. using a bar-check (See Section 3.2.3 [Bar check], page 52), setting `Score.measurePosition` or entering `\partial`. Slurs or ties are also not repeated.

It is possible to nest `\repeats`, although this probably is only meaningful for unfolded repeats.

Folded repeats offer little more over simultaneous music.

3.9.3 Manual repeat commands

The property `repeatCommands` can be used to control the layout of repeats. Its value is a Scheme list of repeat commands, where each repeat command can be

```
'start-repeat
    Print a |: bar line

'end-repeat
    Print a :| bar line

(volta . text)
    Print a volta bracket saying text.

(volta . #f)
    Stop a running volta bracket

c''4
    \property Score.repeatCommands = #'((volta "93") end-repeat)
c''4 c''4
    \property Score.repeatCommands = #'((volta #f))
c''4 c''4
```



Repeats brackets are `VoltaBracket` objects.

3.9.4 Tremolo repeats

To place tremolo marks between notes, use `\repeat` with tremolo style.

```
\score {
  \context Voice \notes\relative c' {
    \repeat "tremolo" 8 { c16 d16 }
    \repeat "tremolo" 4 { c16 d16 }
    \repeat "tremolo" 2 { c16 d16 }
    \repeat "tremolo" 4 c16
  }
}
```



Tremolo beams are `Beam` objects. Single stem tremolos are `StemTremolo`. The single stem tremolo *must* be entered without `{` and `}`.

BUGS

Only powers of two and undotted notes are supported repeat counts.

3.9.5 Tremolo subdivisions

Tremolo marks can be printed on a single note by adding ‘:[*length*]

 after the note. The length must be at least 8. A *length* value of 8 gives one line across the note stem. If the length is omitted, then then the last value (stored in `Voice.tremoloFlags`) is used.

```
c'2:8 c':32 | c': c': |
```



BUGS

Tremolos in this style do not carry over into the MIDI output.

3.9.6 Measure repeats

In the `percent` style, a note pattern can be repeated. It is printed once, and then the pattern is replaced with a special sign. Patterns of a one and two measures are replaced by percent-like signs, patterns that divide the measure length are replaced by slashes.

```
\context Voice { \repeat "percent" 4 { c'4 }
  \repeat "percent" 2 { c'2 es'2 f'4 fis'4 g'4 c''4 }
}
```



The signs are represented by these objects: `RepeatSlash` and `PercentRepeat` and `DoublePercentRepeat`.

BUGS

You can not nest percent repeats, e.g. by filling in the first measure with slashes, and repeating that measure with percents.

3.10 Rhythmic music

Sometimes you might want to show only the rhythm of a melody. This can be done with the rhythmic staff. All pitches of notes on such a staff are squashed, and the staff itself looks has a single staff line:

```
\context RhythmicStaff {
  \time 4/4
  c4 e8 f g2 | r4 g r2 | g1:32 | r1 |
}
```



3.10.1 Percussion staves

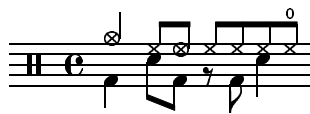
To typeset more than one piece of percussion to be played by the same musician one typically uses a multiline staff where each staff position refers to a specific piece of percussion.

LilyPond is shipped with a bunch of scheme functions which allows you to do this fairly easily.

The system is based on the general midi drum-pitches. In order to use the drum pitches you include ‘`ly/drumpitch-init.ly`’. This file defines the pitches from the scheme variable `drum-pitch-names` - which definition can be read in ‘`scm/drums.scm`’. You see that each piece of percussion has a full name and an abbreviated name - and you may freely select whether to refer to the full name or the abbreviation in your music definition.

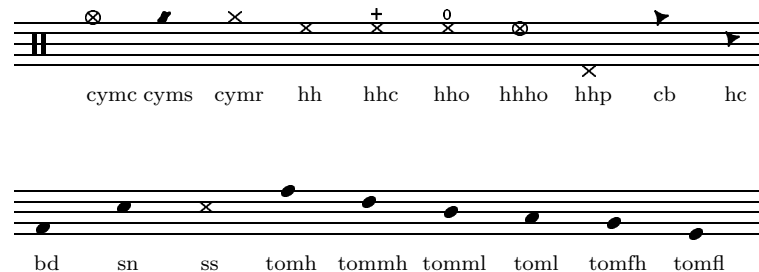
To typeset the music on a staff you apply the scheme function `drums->paper` to the percussion music. This function takes a list of percussion instrument names, notehead scripts and staff positions (that is: pitches relative to the C-clef) and uses this to transform the input music by moving the pitch, changing the notehead and (optionally) adding a script:

```
\include "drumpitch-init.ly"
up = \notes { crashcymbal4 hihat8 halfopenhihat hh hh hh openhihat }
down = \notes { bassdrum4 snare8 bd r bd sn4 }
\score {
  \apply #(drums->paper 'drums) \context Staff <
    \clef percussion
    \context Voice = up { \voiceOne \up }
    \context Voice = down { \voiceTwo \down }
  >
}
```



In the above example the music was transformed using the list ‘`drums`’. Currently the following lists are defined in ‘`scm/drums.scm`’:

`'drums` To typeset a typical drum kit on a five-line staff.

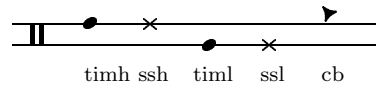


Notice that the scheme supports six different toms. If you are using fewer toms then you simply select the toms that produce the desired result - i.e. to get toms on the three middle lines you use `tommh`, `tomml` and `tomfh`.

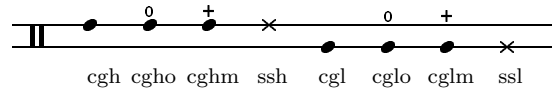
Because the general midi contain no rimshots we use the sidestick for this purpose instead.

`'timbales`

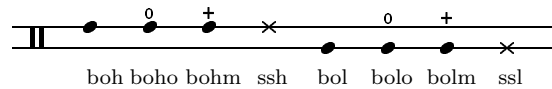
To typeset timbales on a two line staff.



`'congas` To typeset congas on a two line staff.

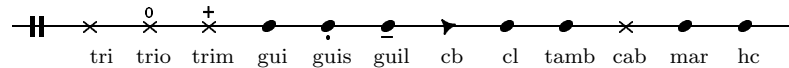


`'bongos` To typeset bongos on a two line staff.



`'percussion`

To typeset all kinds of simple percussion on one line staves.

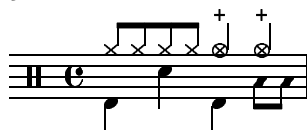


If you don't like any of the predefined lists you can define your own list at the top of your file:

```

#(define mydrums '(
  (bassdrum      default  #f      ,(make-pitch -1 2 0))
  (snare         default  #f      ,(make-pitch 0 1 0))
  (hihat         cross    #f      ,(make-pitch 0 5 0))
  (pedalhihat    xcircle  "stopped" ,(make-pitch 0 5 0))
  (lowtom        diamond  #f      ,(make-pitch -1 6 0))
))
\include "drumpitch-init.ly"
up = \notes { hh8 hh hh hh hhp4 hhp }
down = \notes { bd4 sn bd toml8 toml }
\score {
  \apply #(drums->paper 'mydrums) \context Staff <
    \clef percussion
    \context Voice = up { \voiceOne \up }
    \context Voice = down { \voiceTwo \down }
  >
}

```



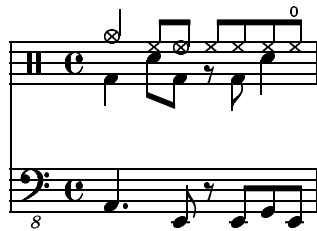
To use a modified existing list instead of building your own from scratch you can append your modifications to the start of the existing list:

```
#(define mydrums (append '(
  (bassdrum default #f ,(make-pitch -1 2 0))
  (lowtom   diamond #f ,(make-pitch -1 6 0))
) drums ))
```

3.10.1.1 Percussion staves with normal staves

When you include ‘drumpitch-init.ly’ then the default pitches are overridden so that you after the inclusion cannot use the common dutch pitch names anymore. Hence you might want to reinclude ‘nederlands.ly’ after the drum-pattern-definitions:

```
\include "drumpitch-init.ly"
up = \notes { crashcymbal4 hihat8 halfopenhihat hh hh hh openhihat }
down = \notes { bassdrum4 snare8 bd r bd sn4 }
\include "nederlands.ly"
bass = \notes \transpose c, { a4. e8 r e g e }
\score {
  <
    \apply #(drums->paper 'drums) \context Staff = drums <
      \clef percussion
      \context Voice = up { \voiceOne \up }
      \context Voice = down { \voiceTwo \down }
    >
    \context Staff = bass { \clef "F_8" \bass }
  >
}
```



3.10.1.2 Percussion midi output

In order to produce correct midi output you need to produce two score blocks - one for the paper and one for the midi. To use the percussion channel you set the property `instrument` to ‘drums’. Because the drum-pitches themselves are similar to the general midi pitches all you have to do is to insert the voices with none of the scheme functions to get the correct midi output:

```
\score {
  \apply #(drums->paper 'mydrums) \context Staff <
    \clef percussion
```

```

        \context Voice = up { \voiceOne \up }
        \context Voice = down { \voiceTwo \down }
    >
    \paper{}
}
\score {
    \context Staff <
        \property Staff.instrument = #'drums
        \up \down
    >
    \midi{}
}

```

BUGS

This scheme is to be considered a temporary implementation. Even though the scheme will probably keep on working then the future might bring some other way of typesetting drums, and probably there will be made no great efforts in keeping things downwards compatible.

3.11 Piano music

Piano music is an odd type of notation. Piano staves are two normal staves coupled with a brace. The staves are largely independent, but sometimes voices can cross between the two staves. The `PianoStaff` is especially built to handle this cross-staffing behavior. In this section we discuss the `PianoStaff` and some other pianistic peculiarities.

3.11.1 Automatic staff changes

Voices can switch automatically between the top and the bottom staff. The syntax for this is

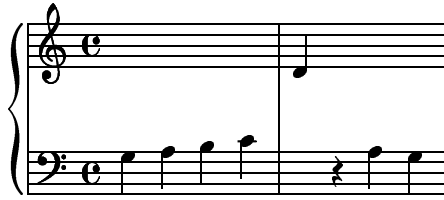
```
\autochange Staff \context Voice { ...music... }
```

The autochanger switches on basis of pitch (central C is the turning point), and it looks ahead skipping over rests to switch rests in advance. Here is a practical example:

```

\score { \notes \context PianoStaff <
    \context Staff = "up" {
        \autochange Staff \context Voice = VA < \relative c' {
            g4 a b c d r4 a g } > }
    \context Staff = "down" {
        \clef bass
        s1*2
    } > }

```



Spacer rests are used to prevent the bottom staff from terminating too soon.

3.11.2 Manual staff switches

Voices can be switched between staves manually, using the following command:

```
\translator Staff = staffname music
```

The string *staffname* is the name of the staff. It switches the current voice from its current staff to the Staff called *staffname*. Typically *staffname* is "up" or "down".

3.11.3 Pedals

Piano pedal instruction can be expressed using `\sustainDown`, `\sustainUp`, `\unaCorda`, `\treCorde`, `\sostenutoDown` and `\sostenutoUp`.

These identifiers are shorthands for spanner commands of the types `Sustain`, `UnaCorda` and `Sostenuto`:

```
c''4 \spanrequest \start "Sustain" c''4
c''4 \spanrequest \stop "Sustain"
```



The symbols that are printed can be modified by setting `pedalXStrings`, where *X* is one of the pedal types: `Sustain`, `Sostenuto` or `UnaCorda`. Refer to the generated documentation of `SustainPedal`, for example, for more information.

Pedals can also be indicated by a sequence of brackets, by setting the `pedal-type` property of `SustainPedal` objects:

```
\property Staff.SustainPedal \override #'pedal-type = #'bracket
c''4 \sustainDown d''4 e''4 a'4
\sustainUp \sustainDown
f'4 g'4 a'4 \sustainUp
```



A third style of pedal notation is a mixture of text and brackets, obtained by setting `pedal-type` to `mixed`:

```
\property Staff.SustainPedal \override #'pedal-type = #'mixed
c''4 \sustainDown d''4 e''4 c'4
\sustainUp \sustainDown
```



```
f'4 g'4 a'4 \sustainUp
```



The default `*Ped` style for sustain and damper pedals corresponds to `\pedal-type = #'text`. However, `mixed` is the default style for a sostenuto pedal:

```
c''4 \sostenutoDown d''4 e''4 c'4 f'4 g'4 a'4 \sostenutoUp
```



For fine-tuning of the appearance of a pedal bracket, the properties `edge-width`, `edge-height`, and `shorten-pair` of `PianoPedalBracket` objects (see the detailed documentation of `PianoPedalBracket`) can be modified. For example, the bracket may be extended to the end of the note head.

```
\property Staff.PianoPedalBracket \override
  #'shorten-pair = #'(0 . -1.0)
c''4 \sostenutoDown d''4 e''4 c'4
f'4 g'4 a'4 \sostenutoUp
```



3.11.4 Arpeggio

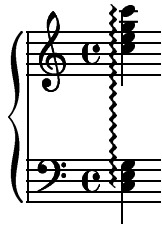
You can specify an arpeggio sign on a chord by attaching an `\arpeggio` to a note of the chord.

```
\context Voice <c\arpeggio e g c>
```



When an arpeggio crosses staves in piano music, you attach an arpeggio to the chords in both staves, and set `PianoStaff.connectArpeggios`.

```
\context PianoStaff <
  \property PianoStaff.connectArpeggios = ##t
  \context Voice = one { <c'\arpeggio e g c> }
  \context Voice = other { \clef bass <c,,\arpeggio e g>}
>
```



This command creates `Arpeggio` objects. Cross staff arpeggios are `PianoStaff.Arpeggio`.

To add an arrow head to explicitly specify the direction of the arpeggio, you should set the arpeggio object property `arpeggio-direction`.

```
\context Voice {
  \property Voice.Arpeggio \set #'arpeggio-direction = #1
  <c\arpeggio e g c>
  \property Voice.Arpeggio \set #'arpeggio-direction = #-1
  <c\arpeggio e g c>
}
```



A square bracket on the left indicates that the player should not arpeggiate the chord. To draw these brackets, set the `molecule-callback` property of `Arpeggio` or `PianoStaff.Arpeggio` objects to `\arpeggioBracket`, and use `\arpeggio` statements within the chords as before.

```
\context PianoStaff <
  \property PianoStaff.connectArpeggios = ##t
  \property PianoStaff.Arpeggio \override
    #'molecule-callback = \arpeggioBracket
  \context Voice = one { <c'\arpeggio e g c> }
  \context Voice = other { \clef bass <c,,\arpeggio e g>}
>
```



BUGS

It is not possible to mix connected arpeggios and unconnected arpeggios in one `PianoStaff` at the same time.

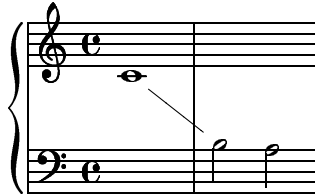
3.11.5 Voice follower lines

Whenever a voice switches to another staff a line connecting the notes can be printed automatically. This is enabled if the property `PianoStaff.followVoice` is set to true:

```

\context PianoStaff <
  \property PianoStaff.followVoice = ##t
  \context Staff \context Voice {
    c1
    \translator Staff=two
    b2 a
  }
  \context Staff=two {\clef bass \skip 1*2 }
>

```



The associated object is `VoiceFollower`.

3.12 Tablatures

Tablature notation is used for notating music for plucked string instruments. It notates pitches not by using note heads, but by indicating on which string and fret a note must be played. LilyPond offers limited support for tablature.

3.12.1 Tablatures basic

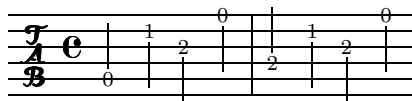
Tablature can be typeset with Lilypond by using the `TabStaff` and `TabVoice` contexts. As tablature is a recent feature in Lilypond, most of the guitar special effects such as bend are not yet supported.

With the `TabStaff`, the string number associated to a note is given as a backslash followed by the string number, e.g. `c4\3` for a C quarter on the third string. By default, string 1 is the highest one, and the tuning defaults to the standard guitar tuning (with 6 strings).

```

\context TabStaff <
  \notes {
    a,4\5 c'\2 a\3 e'\1
    e\4 c'\2 a\3 e'\1
  }
>

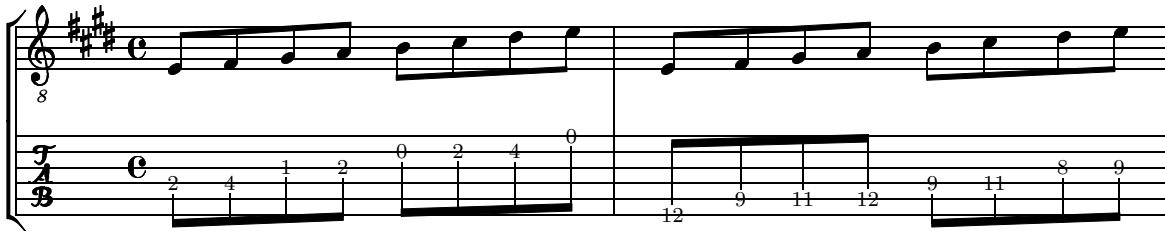
```



If you do not specify a string number then lilypond automatically selects one. The selection is controlled by the translator property `minimumFret`. – LilyPond simply selects the first string that does not give a fret number less than `minimumFret`. Default is 0.

Notice that LilyPond does not handle chords in any special way, and hence the automatic string selector may easily select the same string to two notes in a chord.

```
e8 fis gis a b cis' dis' e'
\property TabStaff.minimumFret = #8
e8 fis gis a b cis' dis' e'
```



3.12.2 Non-guitar tablatures

There are many ways to customize Lilypond tablatures.

First you can change the number of strings, by setting the number of lines in the `TabStaff` (the `line-count` property of `TabStaff` can only be changed using `\outputproperty`, for more information, see Section 3.16.2 [Tuning per object], page 107. You can change the strings tuning. A string tuning is given as a Scheme list with one integer number for each string, the number being the pitch of an open string.

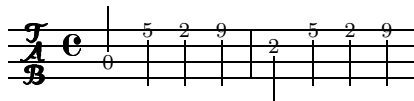
(The numbers specified for `stringTuning` are the numbers of semitons to subtract — or add — starting the specified pitch by default middle C, in string order: thus the notes are e, a, d & g)

```
\context TabStaff <

  \outputproperty #(make-type-checker 'staff-symbol-interface)
    #'line-count = #4
  \property TabStaff.stringTunings = #'(-5 -10 -15 -20)

  \notes {
    a,4 c' a e' e c' a e'
  }

>
```



Finally, it is possible to change the Scheme function to format the tablature note text. The default is *fret-number-tablature-format*, which uses the fret number, but for some instruments that may not use this notation, just create your own *tablature-format* function. This function takes three argument: the string number, the string tuning and the note pitch.

3.12.3 Tablature in addition to normal staff

It is possible to typeset both tablature and a "normal" staff, as commonly done in many parts.

A common trick for that is to put the notes in a variables, and to hide the fingering information (which correspond to the string number) for the standard staff.

```
part = \notes {
  a,4-2 c'-5 a-4 e'-6
  e-3 c'-5 a-4 e'-6
}
\score {
  \context StaffGroup <
    \context Staff <
      % Hide fingering number
      \property Staff.Fingering \override #'transparent = ##t

      \part
    >
    \context TabStaff <
      \property Staff.Stem \override #'direction = #1

      \part
    >
  >
}
```

3.13 Chords

LilyPond has support for both entering and printing chords.

```
twoWays = \notes \transpose c'' {
  \chords {
    c1 f:sus4 bes/f
  }
  <c e g>
  <f bes c'>
  <f bes d'>
}

\score {
  < \context ChordNames \twoWays
    \context Voice \twoWays > }

  C      F4      Bb/F  C      F4      F4/6/no3/no5
```



This example also shows that the chord printing routines do not try to be intelligent. If you enter `f bes d`, it does not interpret this as an inversion.

As you can see chords really are a set of pitches. They are internally stored as simultaneous music expressions. This means you can enter chords by name and print them as notes, enter them as notes and print them as chord names, or (the most common case) enter them by name, and print them as name.

3.13.1 Chords mode

Chord mode is a mode where you can input sets of pitches using common names. It is introduced by the keyword `\chords`. It is similar to note mode, but words are also looked up in a chord modifier table (containing `maj`, `dim`, etc). Dashes and carets are used to indicate chord additions and subtractions, so articulation scripts can not be entered in Chord mode.

Throughout these examples, chords have been shifted around the staff using `\transpose`.

```
\transpose c'' {
  \chords {
    c1 c:3-      c:7      c:8
    c:9 c:9-.5+.7+ c:3-.5-
  }
}
```



The second type of modifier that may appear after the `:` is a named modifier. Named modifiers are listed in the file `'chord-modifiers.ly'`. The available modifiers are `m` and `min` which lower the 3rd half a step, `'aug'` which raises the 5th, `'dim'` which lowers the 5th, `'maj'` which adds a raised 7th, and `'sus'` which replaces the 5th with a 4th.

```
\transpose c'' {
  \chords {
    c1:m c:min7 c:maj c:aug c:dim c:sus
  }
}
```



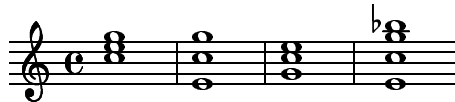
Chord subtractions are used to eliminate notes from a chord. The notes to be subtracted are listed after a `^` character, separated by dots.

```
\transpose c'' {
  \chords {
    c1^3 c:7^5.3 c:8^7
  }
}
```



Chord inversions can be specified by appending ‘/’ and the name of a single note to a chord. In a chord inversion, the inverted note is transposed down until it is the lowest note in the chord. If the note is not in the chord, a warning will be printed.

```
\transpose c''' {
  \chords {
    c1 c/e c/g c:7/e
  }
}
```



Bass notes can be added by ‘/+’ and the name of a single note to a chord. This has the effect of adding the specified note to the chord, lowered by an octave, so it becomes the lowest note in the chord.

```
\transpose c''' {
  \chords {
    c1 c/+c c/+g c:7/+b
  }
}
```



The formal syntax for named chords is as follows:

```
tonic [duration] [-modifiers] [~subtractions] [/inversion] [/+bass].
```

tonic should be the tonic note of the chord, and *duration* is the chord duration in the usual notation. There are two kinds of modifiers. One type is formed by *chord additions*. Additions are obtained by listing intervals separated by dots. An interval is written by its number with an optional + or - to indicate raising or lowering by half a step. Chord additions have two effects: they add the specified interval and all lower odd numbered intervals to the chord, and they may lower or raise the specified interval.

BUGS

Implementation details are gory. For example `c:4` not only adds a fourth, but also removes the third.

3.13.2 Printing named chords

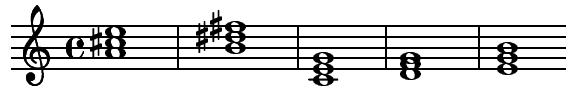
For displaying printed chord names, use the `ChordNames` context. The chords may be entered either using the notation described above, or directly using simultaneous music.

```

scheme = \notes {
  \chords {a1 b c} <d f g> <e g b>
}
\score {
  \notes<
    \context ChordNames \scheme
    \context Staff \transpose c'' \scheme
  >
}

```

A B C D_m^{4/no5}E_m



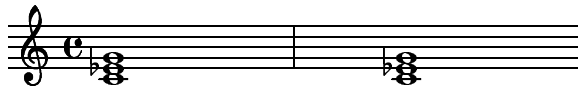
You can make the chord changes stand out by setting `ChordNames.chordChanges` to `true`. This will only display chord names when there's a change in the chords scheme and at the start of a new line.

```

scheme = \chords {
  c1:m c:m \break c:m c:m d
}
\score {
  \notes <
    \context ChordNames {
      \property ChordNames.chordChanges = ##t
      \scheme }
    \context Staff \transpose c'' \scheme
  >
\paper{linewidth= 9.\cm}
}

```

C_m



C_m

D



LilyPond examines chords specified as lists of notes to determine a name to give the chord. LilyPond will not try to identify chord inversions or an added bass note, which may result in strange chord names when chords are entered as a list of pitches:

```

scheme = \notes {
  <c'1 e' g'>
  <e' g' c''>
  <e e' g' c''>
}

```

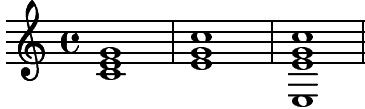


```

\score {
  <
    \context ChordNames \scheme
    \context Staff \scheme
  >
}

C      Em6-/no5E10-/13-/no3/no5/no7/no9/no11

```



By default, a chord name system proposed by Harald Banter (See Chapter 8 [Literature], page 147) is used. The system is very regular and predictable. Typical American style chord names may be selected by setting the `style` property of the `ChordNames.ChordName` object to `'american`. Similarly `'jazz` selects Jazz chordnames.

Routines that determine the names to be printed are written in Scheme, and may be customized by the user. The code can be found in `'scm/chord-name.scm'`. Here's an example showing the differences in chord name styles:


```

scheme = \chords {
  c1 c:5^3 c:4^3 c:5+
  c:m7+ c:m5-.7
  c:5-.7 c:5+.7
  c:9^7
}

\score {
  \notes <
    \context ChordNames = banter \scheme
    \context ChordNames = american {
      \property ChordNames.ChordName \override
        #'style = #'american \scheme }
    \context ChordNames = jazz {
      \property ChordNames.ChordName \override
        #'style = #'jazz \scheme }
    \context Staff \transpose c'' \scheme
  >
}

C      Cno3  C4      C5+      Cmmaj7  Cm5-/7  C5-/7      C5+/7  C9/no7
C      C5    Csus    Caug    Cm(maj7)  C79      C7b5    Caug7  Cadd9
C      C5    Csus    Caug    Cm▲      C79      C7b5    Caug7  Cadd9

```



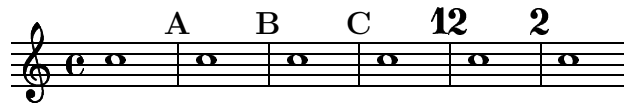
3.14 Writing parts

Orchestral music involves some special notation, both in the full score, as in the individual parts. This section explains how to tackle common problems in orchestral music.

3.14.1 Rehearsal marks

To print a rehearsal mark, use the `\mark` command.

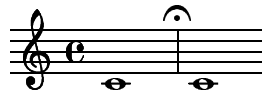
```
\relative c'' {
  c1 \mark "A"
  c1 \mark \default
  c1 \mark \default
  c1 \mark "12"
  c1 \mark \default
  c1
}
```



As you can see, the mark is incremented automatically if you use `\mark \default`. The value to use is stored in the property `rehearsalMark` is used and automatically incremented. The object is `RehearsalMark` in `Score` context. See `input/test/boxed-molecule.ly` if you need boxes around the marks.

The `\mark` command can also be used to put signs like coda, segno and fermatas on a barline. The trick is to use the text markup mechanism to access the fermata symbol.

```
c1 \mark #'(music "scripts-ufermata")
c1
```



The problem is that marks that occur at a line break are typeset only at the beginning of the next line, opposite to what you want for the fermata. This can be corrected by the following property setting

```
\property Score.RehearsalMark \override
  #'visibility-lambda = #begin-of-line-invisible
```

3.14.2 Bar numbers

Bar numbers are printed by default at the start of the line. The number itself is a property that can be set by modifying the `currentBarNumber` property, i.e.

```
\property Score.currentBarNumber = #217
```

To typeset Bar Numbers at regular intervals instead of at the beginning of each line, you need to change the grob property `break-visibility` as well as the translator property

`barNumberVisibility`, as illustrated in the following example which also adds a box around the bar numbers:

```
\property Score.BarNumber \override #'break-visibility =
  #end-of-line-invisible
\property Score.barNumberVisibility = #(every-nth-bar-number-visible 5)
\property Score.BarNumber \override #'molecule-callback =
  #(make-molecule-boxer 0.1 0.25 0.25 Text_item::brew_molecule)
\property Score.BarNumber \override #'font-relative-size = #0
```



See also internals document, `BarNumber`.

BUGS

Barnumbers can collide with the `StaffGroup`, if there is one at the top. To solve this, You have to twiddle with the `padding` property of `BarNumber` if your score starts with a `StaffGroup`.

3.14.3 Instrument names

In scores, the instrument name is printed before the staff. This can be done by setting `Staff.instrument` and `Staff.instr`. This will print a string before the start of the staff. For the first start, `instrument` is used, for the next ones `instr` is used.

```
\property Staff.instrument = "ploink " { c''4 }
```



You can also use markup texts to construct more complicated instrument names:

```
#(define text-flat
  '(((font-relative-size . -2 ) (music "accidentals--1"))))

\score { \notes {
  \property Staff.instrument = #'(((kern . 0.5) (lines
    "2 Clarinetti" (columns "      (B" ,text-flat ")"))))
    c'' 4 }
}
```



BUGS

When you put a name on a grand staff or piano staff the width of the brace is not taken into account. You must add extra spaces to the end of the name to avoid a collision.

3.14.4 Transpose

A music expression can be transposed with `\transpose`. The syntax is

```
\transpose pitch musicexpr
```

This means that middle C in *musicexpr* is transposed to *pitch*.

`\transpose` distinguishes between enharmonic pitches: both `\transpose cis'` or `\transpose des'` will transpose up half a tone. The first version will print sharps and the second version will print flats.

```
mus =\notes { \key d \major cis d fis g }
\score { \notes \context Staff {
  \clef "F" \mus
  \clef "G"
  \transpose g'' \mus
  \transpose f'' \mus
}}
```



If you want to use both `\transpose` and `\relative`, then you must use `\transpose` first. `\relative` will have no effect music that appears inside a `\transpose`.

3.14.5 Multi measure rests

Multi measure rests are entered using 'R'. It is specifically meant for full bar rests and for entering parts: the rest can expand to fill a score with rests, or it can be printed as a single multimeasure rest. This expansion is controlled by the property `Score.skipBars`. If this is set to true, Lily will not expand empty measures, and the appropriate number is added automatically.

```
\time 3/4 r2. | R2. | R2.*2
\property Score.skipBars = ##t R2.*17 R2.*4
```



Notice that the `R2.` is printed as a whole rest, centered in the measure.

The object for this object is `MultiMeasureRest`.

BUGS

Currently, there is no way to automatically condense multiple rests into a single multi-measure rest. Multi measure rests do not take part in rest collisions.

3.14.6 Automatic part combining

Automatic part combining is used to merge two parts of music onto a staff in an intelligent way. It is aimed primarily at typesetting orchestral scores. When the two parts are identical

for a period of time, only one is shown. In places where the two parts differ, they are typeset as separate voices, and stem directions are set automatically. Also, solo and *a due* parts can be identified and marked.

The syntax for part combining is

```
\partcombine context musicexpr1 musicexpr2
```

where the pieces of music *musicexpr1* and *musicexpr2* will be combined into one context of type *context*. The music expressions must be interpreted by contexts whose names should start with *one* and *two*.

The most useful function of the part combiner is to combine parts into one voice, as common for wind parts in orchestral scores:

```
\context Staff <
  \context Voice=one \partcombine Voice
    \context Thread=one \relative c'' {
      g a ( ) b r
    }
  \context Thread=two \relative c'' {
    g r4 r f
  }
>
```



Notice that the first *g* appears only once, although it was specified twice (once in each part). Stem, slur and tie directions are set automatically, depending whether there is a solo or unisono. The first part (with context called *one*) always gets up stems, and ‘solo’, while the second (called *two*) always gets down stems and ‘Solo II’.

If you just want the merging parts, and not the textual markings, you may set the property *soloADue* to false.

```
\context Staff <
  \property Staff.soloADue = ##f
  \context Voice=one \partcombine Voice
    \context Thread=one \relative c'' {
      b4 a c g
    }
  \context Thread=two \relative c'' {
    d,2 a4 g'
  }
>
```



There are a number of other properties that you can use to tweak the behavior of part combining, refer to the automatically generated documentation of *Thread_devnull_engraver*

and `Voice_devnull_engraver`. Look at the documentation of the responsible engravers, `Thread_devnull_engraver`, `Voice_devnull_engraver` and `A2_engraver`.

BUGS

In `soloADue` mode, when the two voices play the same notes on and off, the part combiner may typeset `a2` more than once in a measure.



3.14.7 Hara kiri staves

In orchestral scores, staff lines that only have rests are usually removed. This saves some space. LilyPond also supports this through the `hara kiri`¹ staff. This staff commits suicide when it finds itself to be empty after the line-breaking process. It will not disappear when it contains normal rests, you must use multi measure rests.

The `hara kiri` staff is specialized version of the `Staff` context. It is available as the context identifier `\HaraKiriStaffContext`. Observe how the second staff in this example disappears in the second line.

```
\score {
  \notes \relative c' <
    \context Staff = SA { e4 f g a \break c1 }
    \context Staff = SB { c4 d e f \break R1 }
  >
  \paper {
    linewidth = 6.\cm
    \translator { \HaraKiriStaffContext }
  }
}
```



¹ Hara kiri, also called Seppuku, is the ritual suicide of the Japanese Samourai warriors.

3.14.8 Sound output for transposing instruments

When you want to make a MIDI file from a score containing transposed and untransposed instruments, you have to instruct LilyPond the pitch offset (in semitones) for the transposed instruments. This is done using the `transposing` property. It does not affect printed output.

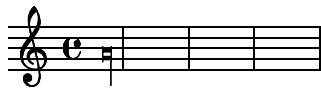
```
\property Staff.instrument = #"Cl. in B-flat"
\property Staff.transposing = #-2
```

3.15 Ancient notation

3.15.1 Ancient note heads

To get a longa note head, you have to use mensural note heads. This is accomplished by setting the `style` property of the `NoteHead` object to `mensural`. There is also a note head style `baroque` which gives mensural note heads for `\longa` and `\breve` but standard note heads for shorter notes.

```
\property Voice.NoteHead \set #'style = #'mensural
a'\longa
```



3.15.2 Ancient clefs

LilyPond supports a variety of clefs, many of them ancient.

For modern clefs, see section Section 3.3.3 [Clef], page 55. For the percussion clef, see section Section 3.10.1 [Percussion staves], page 78. For the TAB clef, see section Section 3.12 [Tablatures], page 85.

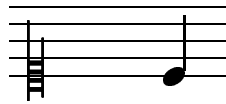
The following table shows all ancient clefs that are supported via the `\clef` command. Some of the clefs use the same glyph, but differ only with respect to the line they are printed on. In such cases, a trailing number in the name is used to enumerate these clefs. Still, you can manually force a clef glyph to be typeset on an arbitrary line, as described in section Section 3.3.3 [Clef], page 55. The note printed to the right side of each clef denotes the `c'` with respect to the clef.

modern style mensural C clef (glyph: `clefs-neo_mensural_c'`)

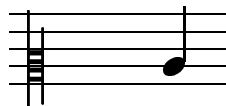
Supported clefs: `neo_mensural_c1`, `neo_mensural_c2`, `neo_mensural_c3`, `neo_mensural_c4`



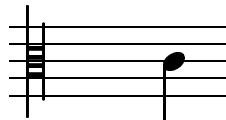
`petrucci style mensural C clef (glyph: clefs-petrucci_c1)`
 Supported clefs: `petrucci_c1` for 1st staffline



`petrucci style mensural C clef (glyph: clefs-petrucci_c2)`
 Supported clefs: `petrucci_c2` for 2nd staffline



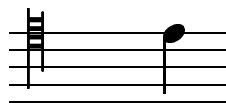
`petrucci style mensural C clef (glyph: clefs-petrucci_c3)`
 Supported clefs: `petrucci_c3` for 3rd staffline



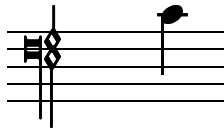
`petrucci style mensural C clef (glyph: clefs-petrucci_c4)`
 Supported clefs: `petrucci_c4` for 4th staffline



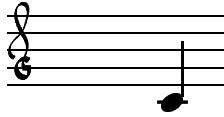
`petrucci style mensural C clef (glyph: clefs-petrucci_c5)`
 Supported clefs: `petrucci_c5` for 5th staffline



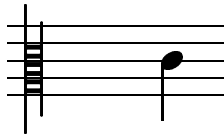
`petrucci style mensural F clef (glyph: clefs-petrucci_f)`
 Supported clefs: `petrucci_f`



petrucci style mensural G clef (glyph: clefs-petrucci_g)
Supported clefs: petrucci_g



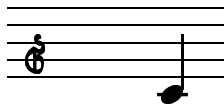
historic style mensural C clef (glyph: clefs-mensural_c')
Supported clefs: mensural_c1, mensural_c2, mensural_c3, mensural_c4



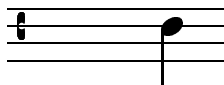
historic style mensural F clef (glyph: clefs-mensural_f)
Supported clefs: mensural_f



historic style mensural G clef (glyph: clefs-mensural_g)
Supported clefs: mensural_g

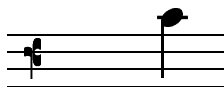


Editio Vaticana style do clef (glyph: clefs-vaticana_do)
Supported clefs: vaticana_do1, vaticana_do2, vaticana_do3



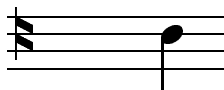
Editio Vaticana style fa clef (glyph: clefs-vaticana_fa)

Supported clefs: vaticana_fa1, vaticana_fa2



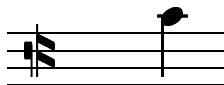
Editio Medicaea style do clef (glyph: clefs-medicaea_do)

Supported clefs: medicaea_do1, medicaea_do2, medicaea_do3



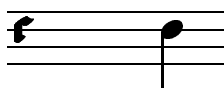
Editio Medicaea style fa clef (glyph: clefs-medicaea_fa)

Supported clefs: medicaea_fa1, medicaea_fa2



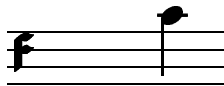
historic style hufnagel do clef (glyph: clefs-hufnagel_do)

Supported clefs: hufnagel_do1, hufnagel_do2, hufnagel_do3



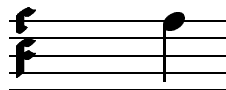
historic style hufnagel fa clef (glyph: clefs-hufnagel_fa)

Supported clefs: hufnagel_fa1, hufnagel_fa2



historic style hufnagel combined do/fa clef (glyph: clefs-hufnagel_do_fa)

Supported clefs: hufnagel_do_fa



Modern style means “as is typeset in current editions of transcribed mensural music”.

Petrucchi style means “inspired by printings published by the famous engraver Petrucci (1466-1539)”.

Historic style means “as was typeset or written in contemporary historic editions (other than those of Petrucci)”.

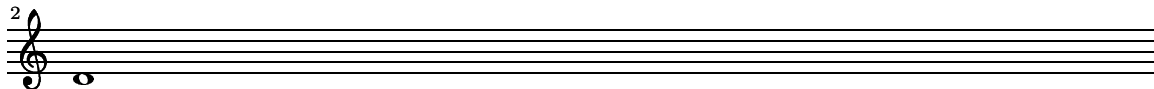
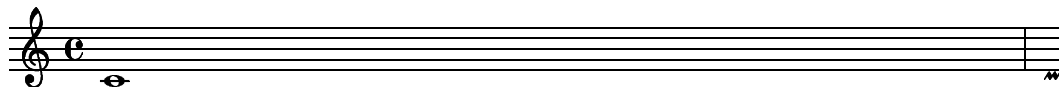
Editio XXX style means “as is/was printed in Editio XXX”.

Petrucchi used C clefs with differently balanced left-side vertical beams, depending on which staffline it was printed.

3.15.3 Custodes

A *custos* (plural: *custodes*; latin word for ‘guard’) is a staff context symbol that appears at the end of a staff line. It anticipates the pitch of the first note(s) of the following line and thus helps the player or singer to manage line breaks during performance, thus enhancing readability of a score.

```
\score {
  \notes { c'1 \break
    \property Staff.Custos \set #'style = #'mensural
    d' }
  \paper {
    \translator {
      \StaffContext
      \consists Custos_engraver
    }
  }
}
```



Custodes were frequently used in music notation until the 17th century. There were different appearances for different notation styles. Nowadays, they have survived only in special forms of musical notation such as via the *editio vaticana* dating back to the beginning of the 20th century.

For typesetting custodes, just put a `Custos_engraver` into the `Staff` context when declaring the `\paper` block. In this block, you can also globally control the appearance of

the `custos` symbol by setting the `custos style` property. Currently supported styles are `vaticana`, `medicaea`, `hufnagel` and `mensural`.

```
\paper {
  \translator {
    \StaffContext
    \consists Custos_engraver
    Custos \override #'style = #'mensural
  }
}
```

The property can also be set locally, for example in a `\notes` block:

```
\notes {
  \property Staff.Custos \override #'style = #'vaticana
  c'1 d' e' d' \break c' d' e' d'
}
```

3.15.4 Ligatures

In musical terminology, a ligature is a coherent graphical symbol that represents at least two different notes. Ligatures originally appeared in the manuscripts of Gregorian chant notation roughly since the 9th century as an allusion to the accent symbols of greek lyric poetry to denote ascending or descending sequences of notes. Both, the shape and the exact meaning of ligatures changed tremendously during the following centuries: In early notation, ligatures were used for monophonic tunes (Gregorian chant) and very soon denoted also the way of performance in the sense of articulation. With upcoming multiphony, the need for a metric system arose, since multiple voices of a piece have to be synchronized some way. New notation systems were invented, that used the manifold shapes of ligatures to now denote rhythmical patterns (e.g. black mensural notation, mannered notation, *ars nova*). With the invention of the metric system of the white mensural notation, the need for ligatures to denote such patterns disappeared. Nevertheless, ligatures were still in use in the mensural system for a couple of decades until they finally disappeared during the late 16th / early 17th century. Still, ligatures have survived in contemporary editions of Gregorian chant such as the *Editio Vaticana* from 1905/08.

Syntactically, ligatures are simply enclosed by `\[` and `\]`. Some ligature styles (such as *Editio Vaticana*) may need additional input syntax specific for this particular type of ligature. By default, the `LigatureBracket` engraver just marks the start and end of a ligature by small square angles:

```
\score {
  \notes \transpose c'' {
    \[ g c a f d' \]
    a g f
    \[ e f a g \]
  }
}
```



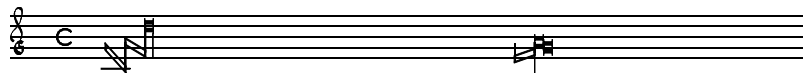
To select a specific style of ligatures, a proper ligature engraver has to be added to the **Voice** context, as explained in the following subsections. Currently, Lilypond only supports white mensural ligatures with certain limitations. Support for *Editio Vaticana* will be added in the future.

3.15.4.1 White mensural ligatures

Lilypond has limited support for white mensural ligatures. The implementation is still experimental; it currently may output strange warnings or even crash in some cases or produce weird results on more complex ligatures. To engrave white mensural ligatures, in the paper block the **MensuralLigature** engraver has to be put into the **Voice** context (and you probably want to remove the **LigatureBracket** engraver). There is no additional input language to describe the shape of a white mensural ligature. The shape is rather determined solely from the pitch and duration of the enclosed notes. While this approach may take a new user quite a while to get accustomed, it has a great advantage: this way, lily has full musical information about the ligature. This is not only required for correct MIDI output, but also allows for automatic transcription of the ligatures.

Example:

```
\score {
  \notes \transpose c'' {
    \property Score.timing = ##f
    \property Score.defaultBarType = "empty"
    \property Voice.NoteHead \set #'style = #'neo_mensural
    \property Staff.TimeSignature \set #'style = #'neo_mensural
    \clef "petrucci_g"
    \[ g\longa c\breve a\breve f\breve d'\longa \]
    s4
    \[ e1 f1 a\breve g\longa \]
  }
  \paper {
    \translator {
      \VoiceContext
      \remove Ligature_bracket_engraver
      \consists Mensural_ligature_engraver
    }
  }
}
```



Without replacing **Ligature_bracket_engraver** with **Mensural_ligature_engraver**, the same music transcribes to the following:

```

\score {
  \notes \transpose c'' {
    \property Score.timing = ##f
    \property Score.defaultBarType = "empty"
    \property Voice.NoteHead \set #'style = #'neo_mensural
    \property Staff.TimeSignature \set #'style = #'neo_mensural
    \clef "petrucci_g"
    \[ g\longa c\breve a\breve f\breve d'\longa \]
    s4
    \[ e1 f1 a\breve g\longa \]
  }
}

```



3.15.5 Figured bass

LilyPond has limited support for figured bass:

```

<
  \context FiguredBass
  \figures {
    <_! 3+ 5- >4
    < [4 6] 8 >
  }
  \context Voice { c4 g8 }
>

```



The support for figured bass consists of two parts: there is an input mode, introduced by `\figures`, where you can enter bass figures as numbers, and there is a context called `FiguredBass` that takes care of making `BassFigure` objects.

In figures input mode, a group of bass figures is delimited by `<` and `>`. The duration is entered after the `>`.

```

<4 6>
6
4

```

Accidentals are added to the numbers if you alterate them by appending `-`, `!` and `+`.

```

<4- 6+ 7!>
7!
6#
4b

```

Spaces or dashes may be inserted by using `_`. Brackets are introduced with `[` and `]`.

```
< [4 6] 8 [_ 12]>
[12]
 8
[6]
 4
```

Although the support for figured bass may superficially resemble chord support, it works much simpler: in figured bass simply stores the numbers, and then prints the numbers you entered. There is no conversion to pitches, and no realizations of the bass are played in the MIDI file.

3.16 Tuning output

LilyPond tries to take as much formatting as possible out of your hands. Nevertheless, there are situations where it needs some help, or where you want to override its decisions. In this section we discuss ways to do just that.

Formatting is internally done by manipulating so called objects (graphic objects). Each object carries with it a set of properties (object properties) specific to that object. For example, a stem object has properties that specify its direction, length and thickness.

The most direct way of tuning the output is by altering the values of these properties. There are two ways of doing that: first, you can temporarily change the definition of a certain type of object, thus affecting a whole set of objects. Second, you can select one specific object, and set a object property in that object.

3.16.1 Tuning groups of objects

A object definition is a Scheme association list, that is stored in a context property. By assigning to that property (using plain `\property`), you can change the resulting objects.

```
c'4 \property Voice.Stem = #'()
```



The `\property` assignment effectively empties the definition of the Stem object. One of the effects is that the recipe of how it should be printed is erased, with the effect of rendering it invisible. The above assignment is available as a standard identifier, for the case that you find this useful:

```
\property Voice.Stem = \turnOff
```

This mechanism is fairly crude, since you can only set, but not modify, the definition of a object. For this reason, there is a more advanced mechanism.

The definition of a object is actually a list of default object properties. For example, the definition of the Stem object (available in `'scm/grob-description.scm'`), defines the following values for `Stem`

```
(thickness . 0.8)
(beamed-lengths . (0.0 2.5 2.0 1.5))
(Y-extent-callback . ,Stem::height)
```

...

You can add a property on top of the existing definition, or remove a property, thus overriding the system defaults:

```
c'4 \property Voice.Stem \override #'thickness = #4.0
c'4 \property Voice.Stem \revert #'thickness
c'4
```



You should balance `\override` and `\revert`. If that's too much work, you can use the `\set` shorthand. It performs a revert followed by an override. The following example gives exactly the same result as the previous one.

```
c'4 \property Voice.Stem \set #'thickness = #4.0
c'4 \property Voice.Stem \set #'thickness = #0.8
c'4
```



If you use `\set`, you must explicitly restore the default.

Formally the syntax for these constructions is

```
\property context.grobname \override symbol = value
\property context.grobname \set symbol = value
\property context.grobname \revert symbol
```

Here *symbol* is a Scheme expression of symbol type, *context* and *grobname* are strings and *value* is a Scheme expression.

If you revert a setting which was not set in the first place, then it has no effect. However, if the setting was set as a system default, it may remove the default value, and this may give surprising results, including crashes. In other words, `\override` and `\revert`, must be carefully balanced.

These are examples of correct nesting of `\override`, `\set`, `\revert`.

A clumsy but correct form:

```
\override \revert \override \revert \override \revert
```

Shorter version of the same:

```
\override \set \set \revert
```

A short form, using only `\set`. This requires you to know the default value:

```
\set \set \set \set to default value
```

If there is no default (i.e. by default, the object property is unset), then you can use

```
\set \set \set \revert
```

For the digirati, the object description is an Scheme association list. Since a Scheme list is a singly linked list, we can treat it as a stack, and `\override` and `\revert` are just push and pop operations. This pushing and popping is also used for overriding automatic beaming settings.

BUGS

LilyPond will hang or crash if *value* contains cyclic references. The backend is not very strict in type-checking object properties. If you `\revert` properties that are expected to be set by default, LilyPond may crash.

3.16.2 Tuning per object

Tuning a single object is most often done with `\property`. The form,

```
\once \property ...
```

applies a setting only during one moment in the score: notice how the original setting for stem thickness is restored automatically in the following example

```
c4
\once \property Voice.Stem \set #'thickness = #4
c4
c4
```



A second way of tuning objects is the more arcane `\outputproperty` feature. The syntax is as follows:

```
\outputproperty predicate symbol = value
```

Here *predicate* is a Scheme function taking a object argument, and returning a boolean. This statement is processed by the `Output_property_engraver`. It instructs the engraver to feed all objects that it sees to *predicate*. Whenever the predicate returns true, the object property *symbol* will be set to *value*.

This command is only single shot, in contrast to `\override` and `\set`.

You will need to combine this statement with `\context` to select the appropriate context to apply this to.

In the following example, all note heads occurring at current staff level, are shifted up and right by setting their `extra-offset` property.

```
\relative c'' { c4
  \context Staff \outputproperty
  #(make-type-checker 'note-head-interface)
  #'extra-offset = #'(0.5 . 0.75)
  <c8 e g> }
```



In this example, the predicate checks the `text` object property, to shift only the 'm.d.' text, but not the fingering instruction "2".

```
#(define (make-text-checker text)
  (lambda (grob) (equal? text (ly-get-grob-property grob 'text))))
```

```

\score {
  \notes\relative c''' {
    \property Voice.Stem \set #'direction = #1
    \outputproperty #(make-text-checker "m.d.")
    #'extra-offset = #'(-3.5 . -4.5)
    a^2^"m.d."
  }
}

```



BUGS

If possible, avoid this feature: the semantics are not very clean, and the syntax and semantics are up for rewrite.

3.16.3 Font selection

The most common thing to change about the appearance of fonts is their size. The font size of a **Voice**, **Staff** or **Thread** context, can be easily changed by setting the **fontSize** property for that context:



This command will not change the size of variable symbols, such as beams or slurs. You can use this command to get smaller symbol for cue notes, but that involves some more subtleties. An elaborate example of those is in `'input/test/cue-notes.ly'`.

The font used for printing a object can be selected by setting **font-name**, e.g.

```

\property Staff.TimeSignature
\set #'font-name = #"cmr17"

```

You may use any font which is available to T_EX, such as foreign fonts or fonts that do not belong to the Computer Modern font family. Font selection for the standard fonts, T_EX's Computer Modern fonts, can also be adjusted with a more fine-grained mechanism. By setting the object properties described below, you can select a different font. All three mechanisms work for every object that supports **font-interface**.

font-family

A symbol indicating the general class of the typeface. Supported are **roman** (Computer Modern), **braces** (for piano staff braces), **music** (the standard music font), **ancient** (the ancient notation font) **dynamic** (font for dynamic signs) and **typewriter**.

font-shape

A symbol indicating the shape of the font, there are typically several font shapes available for each font family. Choices are **italic**, **caps** and **upright**

font-series

A symbol indicating the series of the font. There are typically several font series for each font family and shape. Choices are **medium** and **bold**.

font-relative-size

A number indicating the size relative the standard size. For example, with 20pt staff height, relative size -1 corresponds to 16pt staff height, and relative size +1 corresponds to 23 pt staff height.

font-design-size

A number indicating the design size of the font.

This is a feature of the Computer Modern Font: each point size has a slightly different design. Smaller design sizes are relatively wider, which enhances readability.

For any of these properties, the value `*` (i.e. the *symbol*, `*`, entered as `#'*`), acts as a wildcard. This can be used to override default setting, which are always present. For example:

```
\property Lyrics.LyricText \override #'font-series = #'bold
\property Lyrics.LyricText \override #'font-family = #'typewriter
\property Lyrics.LyricText \override #'font-shape = #'*
```

There are also pre-cooked font selection qualifiers. These are selected through the object property **font-style**. For example, the style **finger** selects family **number** and relative size -3. Styles available include **volta**, **finger**, **tuplet**, **timesig**, **mmrest**, **script**, **large**, **Large** and **dynamic**. The style sheets and tables for selecting fonts are located in `'scm/font.scm'`. Refer to this file for more information.

The size of the font may be scaled with the object property **font-magnification**. For example, 2.0 blows up all letters by a factor 2 in both directions.

BUGS

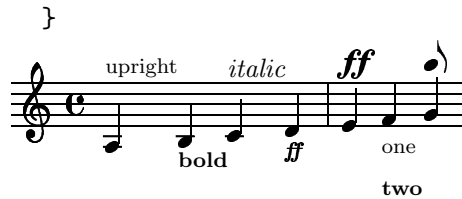
Relative size is not linked to any real size.

There is no style sheet provided for other fonts besides the T_EX family, and the style sheet can not be modified easiyl.

3.16.4 Text markup

LilyPond has an internal mechanism to typeset texts. You can form text markup expressions by composing scheme expressions in the following way.

```
\relative c' {
  \fatText
  a^#"upright"
  b_#'(bold "bold")
  c^#'(italic "italic")
  d_#'((bold italic) "ff")
  e^#'(dynamic "ff")
  f_#'(lines "one" (bold "two"))
  g^#'(music "noteheads-2" ((raise . 2.4) "flags-u3"))
```



Normally, the Scheme markup text is stored in the `text` property of a object. Formally, it is defined as follows:

```
text: string | (head? text+)
head: markup | (markup+)
markup-item: property | abbrev
property: (key . value)
abbrev: columns lines roman music bold italic named super sub
       overstrike text finger volta timesig mmrest mark script
       large Large dynamic
```

The markup is broken down and converted into a list of object properties, which are prepended to the property list. The *key-value* pair is a object property. A list of properties available is included in the generated documentation for `text-interface`.

The following abbreviations are defined:

<code>columns</code>	horizontal mode: set all text on one line (default)
<code>lines</code>	vertical mode: set every text on a new line
<code>roman</code>	select roman font
<code>music</code>	selects the Feta font (the standard font for music notation glyphs), and uses named lookup
<code>bold</code>	select bold series
<code>italic</code>	select italic shape
<code>named</code>	lookup by character name
<code>text</code>	plain text lookup (by character value)
<code>super</code>	superscript
<code>sub</code>	subscript
<code>overstrike</code>	the next text or character overstrikes this one
<code>finger</code>	select fingering number fontstyle
<code>volta</code>	select volta number fontstyle
<code>timesig</code>	select time signature number fontstyle
<code>mmrest</code>	select multi measure rest number fontstyle
<code>mark</code>	select mark number fontstyle
<code>script</code>	select scriptsize roman fontstyle
<code>large</code>	select large roman fontstyle

Large select Large roman fontstyle

dynamic select dynamics fontstyle

One practical application of complicated markup is to fake a metronome marking:

```
#(define note '(columns
  (music "noteheads-2" ((kern . -0.1) "flags-stem"))))
#(define eight-note '(columns ,note ((kern . -0.1)
  (music ((raise . 3.5) "flags-u3")))))
#(define dotted-eight-note
  '(columns ,eight-note (music "dots-dot"))))

\score {
  \notes\relative c'' {
    a1^#((columns (font-relative-size . -1))
      ,dotted-eight-note " = 64")
  }
  \paper {
    linewidth = -1.
    \translator{
      \ScoreContext
      TextScript \override #'font-shape = #'upright
    }
  }
}
```



BUGS

The syntax and semantics of markup texts are not clean, and both syntax and semantics are slated for a rewrite.

LilyPond does not do kerning, and there generally spaces texts slightly too wide.

3.17 Global layout

The global layout determined by three factors: the page layout, the line breaks and the spacing. These all influence each other: The choice of spacing determines how densely each system of music is set, where line breaks are chosen, and thus ultimately how many pages a piece of music takes. In this section we will explain how the lilypond spacing engine works, and how you can tune its results.

Globally spoken, this procedure happens in three steps: first, flexible distances (“springs”) are chosen, based on durations. All possible line breaking combination are tried, and the one with the best results—a layout that has uniform density and requires as little stretching or cramping as possible—is chosen. When the score is processed by \TeX , page are filled with systems, and page breaks are chosen whenever the page gets full.

3.17.1 Vertical spacing

The height of each system is determined automatically by lilypond, to keep systems from bumping into each other, some minimum distances are set. By changing these, you can put staves closer together, and thus put more systems onto one page.

Normally staves are stacked vertically. To make staves maintain a distance, their vertical size is padded. This is done with the property `minimumVerticalExtent`. It takes a pair of numbers, so if you want to make it smaller from its, then you could set

```
\property Staff.minimumVerticalExtent = #'(-4 . 4)
```

This sets the vertical size of the current staff to 4 staff-space on either side of the center staff line. The argument of `minimumVerticalExtent` is interpreted as an interval, where the center line is the 0, so the first number is generally negative. you could also make the staff larger at the bottom by setting it to `(-6 . 4)`. The default value is `(-6 . 6)`.

Vertical alignment of staves is handled by the `VerticalAlignment` object, which lives at `Score` level.

The piano staves are handled a little differently: to make cross-staff beaming work correctly, it necessary that the distance between staves is fixed. This is also done with a `VerticalAlignment` object, created in `PianoStaff`, but a forced distance is set. This is done with the object property `#'forced-distance`. If you want to override this, use a `\translator` block as follows:

```
\translator {
  \PianoStaffContext
  VerticalAlignment \override #'forced-distance = #9
}
```

This would bring the staves together at a distance of 9 staff spaces, and again this is measured from the center line of each staff.

3.17.2 Horizontal Spacing

The spacing engine translates differences in durations into stretchable distances (“springs”) of differing lengths. Longer durations get more space, shorter durations get less. The basis for assigning spaces to durations, is that the shortest durations get a fixed amount of space, and the longer durations get more: doubling a duration adds a fixed amount of space to the note.

For example, the following piece contains lots of half, quarter and 8th notes, the eighth note is followed by 1 note head width. The The quarter note is followed by 2 NHW, the half by 3 NHW, etc.

```
c2 c4. c8 c4. c8 c4. c8 c8 c4 c4 c4
```



These two amounts of space are `shortest-duration-space` `spacing-increment`, object properties of `SpacingSpanner`. Normally `spacing-increment` is set to 1.2, which is the

width of a note head, and `shortest-duration-space` is set to 2.0, meaning that the shortest note gets 2 noteheads of space. For normal notes, this space is always counted from the left edge of the symbol, so the short notes in a score is generally followed by one note head width of space.

If one would follow the above procedure exactly, then adding a single 32th note to a score that uses 8th and 16th notes, would widen up the entire score a lot. The shortest note is no longer a 16th, but a 64th, thus adding 2 noteheads of space to every note. To prevent this, the shortest duration for spacing is not the shortest note in the score, but the most commonly found shortest note. Notes that are even shorter this are followed by a space that is proportional to their duration relative to the common shortest note. So if we were to add only a few 16th notes to the example above, they would be followed by half a NHW:

```
c2 c4. c8 c4. [c16 c] c4. c8 c8 c8 c4 c4 c4
```



The most common shortest duration is determined as follows: in every measure, the shortest duration is determined. The most common short duration, is taken as the basis for the spacing, with the stipulation that this shortest duration should always be equal to or shorter than 1/8th note. The shortest duration is printed when you run lilypond with `--verbose`. These durations may also be customized. If you set the `common-shortest-duration` in `SpacingSpanner`, then this sets the base duration for spacing. The maximum duration for this base (normally 1/8th), is set through `base-shortest-duration`.

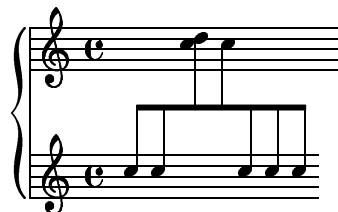
In the introduction it was explained that stem directions influence spacing. This is controlled with `stem-spacing-correction` in `NoteSpacing`. The `StaffSpacing` object contains the same property for controlling the stem/barline spacing. In the following example shows these corrections, once with default settings, and once with exaggerated corrections.



BUGS

Spacing is determined on a score wide basis. If you have a score that changes its character (measured in durations) half way during the score, the part containing the longer durations will be spaced too widely.

Generating optically pleasing spacing is black magic. LilyPond tries to deal with a number of frequent cases. Here is an example that is not handled correctly, due to the combination of chord collisions and kneed stems.



3.17.3 Font size

The Feta font provides musical symbols at seven different sizes. These fonts are 11 point, 13 point, 16 point, 19 pt, 20 point, 23 point, and 26 point. The point size of a font is the height of the five lines in a staff when displayed in the font.

Definitions for these sizes are the files ‘`paperSZ.ly`’, where `SZ` is one of 11, 13, 16, 19, 20, 23 and 26. If you include any of these files, the identifiers `paperEleven`, `paperThirteen`, `paperSixteen`, `paperNineteen`, `paperTwenty`, `paperTwentythree`, and `paperTwentysix` are defined respectively. The default `\paper` block is also set. These files should be imported at toplevel, i.e.

```
\include "paper26.ly"
\score { ... }
```

The font definitions are generated using a Scheme function. For more details, see the file ‘`scm/font.scm`’.

3.17.4 Line breaking

Line breaks are normally computed automatically. They are chosen such that it looks neither cramped nor loose, and that consecutive lines have similar density.

Occasionally you might want to override the automatic breaks; you can do this by specifying `\break`. This will force a line break at this point. Line breaks can only occur at places where there are bar lines. If you want to have a line break where there is no bar line, you can force an invisible bar line by entering `\bar ""`. Similarly, `\noBreak` forbids a line break at a certain point.

If you want linebreaks at regular intervals, you can use the following:

```
< \repeat 7 unfold { s1 * 4 \break }
    real music
>
```

This makes the following 28 measures (assuming 4/4 time) be broken every 4 measures.

3.17.5 Page layout

The most basic settings influencing the spacing are `linewidth` and `indent`, both set in the `\paper` block. They control the indentation of the first line of music, and the lengths of the lines. If `linewidth` set to a negative value, a single unjustified line is produced. A similar effect for scores that are longer than one line, can be produced by setting `raggedright` to true in the `\paper` block.

The page layout process happens outside lilypond. Ly2dvi sets page layout instructions. Ly2dvi responds to the following variables in the `\paper` block. The variable `textheight` sets the total height of the music on each page. The spacing between systems is controlled with `interscoreline`, its default is 16pt. The distance between the score lines will stretch in order to fill the full page `interscorelinefill` is set to a positive number. In that case `interscoreline` specifies the minimum spacing.

If the variable `lastpagefill` is defined (that is, it gets any value assigned in the `\paper` block), systems are evenly distributed vertically on the last page. This might produce ugly

results in case there are not enough systems on the last page. Note that `lilypond-book` ignores `lastpagefill`. See Chapter 6 [Integrating text and music with `lilypond-book`], page 136 for more information.

Page breaks are normally computed by `TeX`, so they are not under direct control of LilyPond. However, you can insert a commands into the `.tex` output to instruct `TeX` where to break pages. You can insert a `\newpage` from within `lilypond`. This is done by setting the `between-systems-strings` on the `NonMusicalPaperColumn` where the system is broken.

To change the paper size, you must first set the `papersize` paper variable variable. Set it to the strings `a4`, `letter`, or `legal`. After this specification, you must set the font as described above. If you want the default font, then use the 20 point font.

```
\paper{ papersize = "a4" }
\include "paper16.ly"
```

The file `paper16.ly` will now include a file named `a4.ly`, which will set the paper variables `hsize` and `vsize` (used by Lilypond and `ly2dvi`)

3.18 Sound

LilyPond can produce MIDI output. The performance lacks lots of interesting effects, such as swing, articulation, slurring, etc., but it is good enough for proof-hearing the music you have entered. Ties, dynamics and tempo changes are interpreted.

Dynamic marks, crescendi and decrescendi translate into MIDI volume levels. Dynamic marks translate to a fixed fraction of the available MIDI volume range, crescendi and decrescendi make the the volume vary linearly between their two extremities. The fractions be adjusted by overriding the `absolute-volume-alist` defined in `scm/midi.scm`.

For each type of musical instrument (that MIDI supports), a volume range can be defined. This gives you basic equalizer control, which can enhance the quality of the MIDI output remarkably. You can add instruments and ranges or change the default settings by overriding the `instrument-equalizer-alist` defined in `scm/midi.scm`.

Both loudness controls are combined to produce the final MIDI volume.

3.18.1 MIDI block

The MIDI block is analogous to the paper block, but it is somewhat simpler. The `\midi` block can contain:

- a `\tempo` definition
- context definitions

Assignments in the `\midi` block are not allowed.

Context definitions follow precisely the same syntax as within the `\paper` block. Translation modules for sound are called performers. The contexts for MIDI output are defined in `ly/performer-init.ly`.

3.18.2 MIDI instrument names

The MIDI instrument name is set by the `Staff.midiInstrument` property or, if that property is not set, the `Staff.instrument` property. The instrument name should be chosen from the list in Section A.4 [MIDI instruments], page 159.

BUGS

If the selected string does not exactly match, then LilyPond uses the default (Grand Piano). It is not possible to select an instrument by number.

4 Advanced Topics

When translating the input to notation, there are number of distinct phases. We list them here:

The purpose of LilyPond is explained informally by the term ‘music typesetter’. This is not a fully correct name: Not only does the program print musical symbols, it also makes aesthetic decisions. Symbols and their placements are *generated* from a high-level musical description. In other words, LilyPond would be best described to be a ‘music compiler’ or ‘music to notation compiler’.

LilyPond is linked to GUILE, GNU’s Scheme library for extension programming. The Scheme library provides the glue that holds together the low-level routines and separate modules which are written in C++.

When lilypond is run to typeset sheet music, the following happens:

- **GUILE initialization:** Various scheme files are read.
- **Parsing:** First standard `ly` initialization files are read, then the user ‘`ly`’ file is read.
- **Interpretation:** The music in the file is processed ‘in playing order’, i.e., the order that you use to read sheet music, or the order in which notes are played. The result of this step is a typesetting specification.
- **Typesetting:** The typesetting specification is solved: positions and formatting is calculated.
- The visible results (“virtual ink”) are written to the output file.

During these stages different types of data play the the main role: During parsing, **Music** objects are created. During the interpretation, **contexts** are constructed, and with these contexts a network of **graphical objects** (‘grobs’) is created. These grobs contain unknown variables, and the network forms a set of equations. After solving the equations and filling in these variables, the printed output is written to an output file.

These threemanship of tasks (parsing, translating, typesetting) and data-structures (music, context, graphical objects) permeates the entire design of the program.

Parsing

The `ly` file is read and converted to a list of **Scores**, which each contain **Music** and paper/midi-definitions. Here **Music**, **Pitch**, and **Duration** objects are created.

Interpreting music

All music events are ‘read’ in the same order as they would be played (or read from paper). At every step of the interpretation, musical events are delivered to interpretation contexts, which use them to build **Grobs** (or MIDI objects for MIDI output).

In this stage **Music_iterators** do a traversal of the **Music** structure. The music events thus encountered are reported to **Translators**, a set of objects that collectively form interpretation contexts.

Prebreaking

At places where line breaks may occur, clefs and bars are prepared for a possible line break.

Preprocessing

In this stage, all information that is needed to determine line breaking is computed.

Break calculation

The lines and horizontal positions of the columns are determined.

Breaking

Relations between all grobs are modified to reflect line breaks: When a spanner, e.g. a slur, crosses a line break, then the spanner is ‘broken into pieces’; for every line that the spanner is in, a copy of the grob is made. A substitution process redirects all grob references so that each spanner grob will only reference other grobs in the same line.

Outputting

All vertical dimensions and spanning objects are computed, and all grobs are output, line by line. The output is encoded in the form of **Molecules**

The data types that are mentioned here are all discussed in this section.

4.1 Interpretation context

Interpretation contexts are objects that only exist during a run of LilyPond. During the interpretation phase of LilyPond (when it prints **interpreting music** to standard output), the music expression in a **\score** block is interpreted in time order. This is the same order that humans hear and play the music.

During this interpretation, the interpretation context holds the state for the current point within the music. It contains information like

- What notes are playing at this point?
- What symbols will be printed at this point?
- What is the current key signature, time signature, point within the measure, etc.?

Contexts are grouped hierarchically: A **Voice** context is contained in a **Staff** context (because a staff can contain multiple voices at any point), a **Staff** context is contained in **Score**, **StaffGroup**, or **ChoirStaff** context.

Contexts associated with sheet music output are called *notation contexts*, those for sound output are called *performance contexts*. The default definitions of the standard notation and performance contexts can be found in ‘ly/engraver-init.ly’ and ‘ly/performer-init.ly’, respectively.

4.1.1 Creating contexts

Contexts for a music expression can be selected manually, using the following music expression.

```
\context contexttype [= contextname] musicexpr
```

This instructs lilypond to interpret *musicexpr* within the context of type *contexttype* and with name *contextname*. If this context does not exist, it will be created.

```
\score {
  \notes \relative c'' {
    c4 <d4 \context Staff = "another" e4> f
  }
}
```



In this example, the `c` and `d` are printed on the default staff. For the `e`, a context `Staff` called **another** is specified; since that does not exist, a new context is created. Within **another**, a (default) `Voice` context is created for the `e4`. When all music referring to a context is finished, the context is ended as well. So after the third quarter, **another** is removed.

4.1.2 Default contexts

Most music expressions don't need an explicit `\context` declaration: they inherit the notation context from their parent. Each note is a music expression, and as you can see in the following example, only the sequential music enclosing the three notes has an explicit context.

```
\score { \notes \context Voice = goUp { c'4 d' e' } }
```



There are some quirks that you must keep in mind when dealing with defaults:

First, every top level music is interpreted by the `Score` context; in other words, you may think of `\score` working like

```
\score {
  \context Score music
}
```

Second, contexts are created automatically to be able to interpret the music expressions. Consider the following example.

```
\score { \context Score \notes { c'4 ( d' )e' } }
```



The sequential music is interpreted by the `Score` context initially (notice that the `\context` specification is redundant), but when a note is encountered, contexts are setup to accept that note. In this case, a `Thread`, `Voice`, and `Staff` context are created. The rest of the

sequential music is also interpreted with the same Thread, Voice, and Staff context, putting the notes on the same staff, in the same voice.

This is a convenient mechanism, but do not expect opening chords to work without `\context`. For every note, a separate staff is instantiated.

```
\score { \notes <c'4 es'> }
```



Of course, if the chord is preceded by a normal note in sequential music, the chord will be interpreted by the Thread of the preceding note:

```
\score { \notes { c'4 <c'4 es'> } }
```



4.1.3 Context properties

Notation contexts have properties. These properties are from the `‘.ly’` file using the following expression:

```
\property contextname.propname = value
```

Sets the *propname* property of the context *contextname* to the specified Scheme expression *value*. All *propname* and *contextname* are strings, which are typically unquoted.

Properties that are set in one context are inherited by all of the contained contexts. This means that a property valid for the **Voice** context can be set in the **Score** context (for example) and thus take effect in all **Voice** contexts.

If you don't wish to specify the name of the context in the `\property`-expression itself, you can refer to the abstract context name, **Current**. The **Current** context is the latest used context. This will typically mean the **Thread** context, but you can force another context with the `\property`-command. Hence the expressions

```
\property contextname.propname = value
```

and

```
\context contextname
```

```
\property Current.propname = value
```

do the same thing. The main use for this is in macros – allowing the specification of a property-setting without restriction to a specific context.

Properties can be unset using the following expression:

```
\property contextname.propname \unset
```

This removes the definition of *propname* in *contextname*. If *propname* was not defined in *contextname* (but was inherited from a higher context), then this has no effect.

BUGS

The syntax of `\unset` is asymmetric: `\property \unset` is not the inverse of `\property \set`.

4.1.4 Engravers and performers

[TODO]

Basic building blocks of translation are called engravers; they are special C++ classes.

4.1.5 Changing context definitions

The most common way to define a context is by extending an existing context. You can change an existing context from the paper block by first initializing a translator with an existing context identifier:

```
\paper {
  \translator {
    context-identifier
  }
}
```

Then you can add and remove engravers using the following syntax:

```
\remove engravername
\consists engravername
```

Here *engravername* is a string, the name of an engraver in the system.

```
\score {
  \notes {
    c'4 c'4
  }
  \paper {
    \translator {
      \StaffContext
      \remove Clef_engraver
    }
  }
}
```



You can also set properties in a translator definition. The syntax is as follows:

```
propname = value
propname \set grob-propname = pvalue
propname \override grob-propname = pvalue
propname \revert grob-propname
```

propname is a string, *grob-propname* a symbol, *value* and *pvalue* are Scheme expressions. These types of property assignments happen before interpretation starts, so a `\property` command will override any predefined settings.

To simplify editing translators, all standard contexts have standard identifiers called *nameContext*, e.g. *StaffContext*, *VoiceContext*; see ‘`ly/engraver-init.ly`’.

4.1.6 Defining new contexts

If you want to build a context from scratch, you must also supply the following extra information:

- A name, specified by `\name contextname`.
- A cooperation module. This is specified by `\type typename`.

This is an example:

```
\translator
  \type "Engraver_group_engraver"
  \name "SimpleStaff"
  \alias "Staff"
  \consists "Staff_symbol_engraver"
  \consists "Note_head_engraver"
  \consistsend "Axis_group_engraver"
```

The argument of `\type` is the name for a special engraver that handles cooperation between simple engravers such as *Note_head_engraver* and *Staff_symbol_engraver*. Alternatives for this engraver are the following:

Engraver_group_engraver

The standard cooperation engraver.

Score_engraver

This is a cooperation module that should be in the top level context.

Other modifiers are

- `\alias alternate-name`: This specifies a different name. In the above example, `\property Staff.X = Y` will also work on *SimpleStaffs*
- `\consistsend engravename`: Analogous to `\consists`, but makes sure that *engravename* is always added to the end of the list of engravers.

Some engraver types need to be at the end of the list; this insures they stay there even if a user adds or removes engravers. End-users generally don't need this command.

- `\accepts contextname`: Add *contextname* to the list of contexts this context can contain in the context hierarchy. The first listed context is the context to create by default.
- `\denies`: The opposite of `\accepts`. Added for completeness, but is never used in practice.

- `\name contextname`: This sets the type name of the context, e.g. **Staff**, **Voice**. If the name is not specified, the translator won't do anything.

In the `\paper` block, it is also possible to define translator identifiers. Like other block identifiers, the identifier can only be used as the very first item of a translator. In order to define such an identifier outside of `\score`, you must do

```
\paper {
  foo = \translator { ... }
}
\score {
  \notes {
    ...
  }
  \paper {
    \translator { \foo ... }
  }
}
```

4.2 Syntactic details

This section describes details that were too boring to be put elsewhere.

4.2.1 Identifiers

All of the information in a LilyPond input file is internally represented as a Scheme value. In addition to normal Scheme data types (such as pair, number, boolean, etc.), LilyPond has a number of specialized data types,

- Input
- c++-function
- Music
- Identifier
- Translator_def
- Duration
- Pitch
- Score
- Music_output_def
- Moment (rational number)

LilyPond also includes some transient object types. Objects of these types are built during a LilyPond run, and do not ‘exist’ per se within your input file. These objects are created as a result of your input file, so you can include commands in the input to manipulate them, during a LilyPond run.

- Grob: short for ‘Graphical object’.
- Molecule: Device-independent page output object, including dimensions. Produced by some Grob functions.

- `Translator`: An object that produces audio objects or Grobs. This is not yet user-accessible.
- `Font_metric`: An object representing a font.

4.2.2 Music expressions

Music in LilyPond is entered as a music expression. Notes, rests, lyric syllables are music expressions, and you can combine music expressions to form new ones, for example by enclosing a list of expressions in `\sequential { }` or `< >`. In the following example, a compound expression is formed out of the quarter note `c` and a quarter note `d`:

```
\sequential { c4 d4 }
```

The two basic compound music expressions are simultaneous and sequential music.

```
\sequential { musicexprlist }
\simultaneous { musicexprlist }
```

For both, there is a shorthand:

```
{ musicexprlist }
```

for sequential and

```
< musicexprlist >
```

for simultaneous music. In principle, the way in which you nest sequential and simultaneous to produce music is not relevant. In the following example, three chords are expressed in two different ways:

```
\notes \context Voice {
  <a c'> <b d'> <c' e'>
  < { a b c' } { c' d' e' } >
}
```



Other compound music expressions include

```
\repeat expr
\transpose pitch expr
\apply func expr
\context type = id expr
\times fraction expr
```

4.2.3 Manipulating music expressions

The `\apply` mechanism gives you access to the internal representation of music. You can write Scheme-functions that operate directly on it. The syntax is

```
\apply #func music
```

This means that `func` is applied to `music`. The function `func` should return a music expression.

This example replaces the text string of a script. It also shows a dump of the music it processes, which is useful if you want to know more about how music is stored.

```
#(define (testfunc x)
  (if (equal? (ly-get-mus-property x 'text) "foo")
      (ly-set-mus-property! x 'text "bar"))
  ;; recurse
  (ly-set-mus-property! x 'elements
    (map testfunc (ly-get-mus-property x 'elements)))
  (display x)
  x)

\score {
  \notes
  \apply #testfunc { c'4_"foo" }
}
```



For more information on what is possible, see the automatically generated documentation.

Directly accessing internal representations is dangerous: The implementation is subject to changes, so you should avoid this feature if possible.

A final example is a function that reverses a piece of music in time:

```
#(define (reverse-music music)
  (let* ((elements (ly-get-mus-property music 'elements))
        (reversed (reverse elements))
        (span-dir (ly-get-mus-property music 'span-direction)))
    (ly-set-mus-property! music 'elements reversed)
    (if (dir? span-dir)
        (ly-set-mus-property! music 'span-direction (- span-dir)))
    (map reverse-music reversed)
    music))

music = \notes { c'4 d'4( e'4 f'4 }

\score {
  \context Voice {
    \music
    \apply #reverse-music \music
  }
}
```



More examples are given in the distributed example files in `input/test/`.

4.2.4 Span requests

Notational constructs that start and end on different notes can be entered using span requests. The syntax is as follows:

```
\spanrequest startstop type
```

This defines a spanning request. The *startstop* parameter is either -1 (`\start`) or 1 (`\stop`) and *type* is a string that describes what should be started. Much of the syntactic sugar is a shorthand for `\spanrequest`, for example

```
c'4-\spanrequest \start "slur"
c'4-\spanrequest \stop "slur"
```



Among the supported types are `crescendo`, `decrescendo`, `beam`, `slur`. This is an internal command. Users are encouraged to use the shorthands which are defined in the initialization file '`spanners.ly`'.

4.2.5 Assignments

Identifiers allow objects to be assigned to names during the parse stage. To assign an identifier, use *name=value*. To refer to an identifier, precede its name with a backslash: '`\name`'. *value* is any valid Scheme value or any of the input-types listed above. Identifier assignments can appear at top level in the LilyPond file, but also in `\paper` blocks.

An identifier can be created with any string for its name, but you will only be able to refer to identifiers whose names begin with a letter, being entirely alphabetical. It is impossible to refer to an identifier whose name is the same as the name of a keyword.

The right hand side of an identifier assignment is parsed completely before the assignment is done, so it is allowed to redefine an identifier in terms of its old value, e.g.

```
foo = \foo * 2.0
```

When an identifier is referenced, the information it points to is copied. For this reason, an identifier reference must always be the first item in a block.

```
\paper {
  foo = 1.0
  \paperIdent % wrong and invalid
}

\paper {
  \paperIdent % correct
  foo = 1.0
}
```

4.2.6 Lexical modes

To simplify entering notes, lyrics, and chords, LilyPond has three special input modes in addition to the default mode: note, lyrics, and chords mode. These input modes change the way that normal, unquoted words are interpreted: For example, the word `cis` may be interpreted as a C-sharp, as a lyric syllable ‘cis’ or as a C-sharp major triad respectively.

A mode switch is entered as a compound music expression

```
\notes musicexpr
\chords musicexpr
\lyrics musicexpr
```

In each of these cases, these expressions do not add anything to the meaning of their arguments. They just instruct the parser in what mode to parse their arguments.

Different input modes may be nested.

4.2.7 Ambiguities

The grammar contains a number of ambiguities. We hope to resolve them at some time.

- The assignment

```
foo = bar
```

is interpreted as the string identifier assignment. However, it can also be interpreted as making a string identifier `\foo` containing "bar", or a music identifier `\foo` containing the syllable ‘bar’. The former interpretation is chosen.

- If you do a nested repeat like

```
\repeat ...
\repeat ...
\alternative
```

then it is ambiguous to which `\repeat` the `\alternative` belongs. This is the classic if-then-else dilemma. It may be solved by using braces.

4.3 Lexical details

Even more boring details, now on the lexical side of the input parser.

4.3.1 Direct Scheme

LilyPond internally uses GUILE, a Scheme-interpreter. Scheme is a language from the LISP family. You can learn more about Scheme at <http://www.scheme.org>. It is used to represent data throughout the whole program. The hash-sign (#) accesses GUILE directly: The code following the hash-sign is evaluated as Scheme. The boolean value *true* is `#t` in Scheme, so for LilyPond *true* looks like `##t`.

LilyPond contains a Scheme interpreter (the GUILE library) for internal use. In some places, Scheme expressions also form valid syntax: Wherever it is allowed,

`#scheme`

evaluates the specified Scheme code. Example:

```
\property Staff.TestObject \override #'foobar = #(+ 1 2)
```

`\override` expects two Scheme expressions. The first one is a symbol (`foobar`), the second one an integer (namely, 3).

In-line Scheme may be used at the top level. In this case the result is discarded.

Scheme is a full-blown programming language, and a full discussion is outside the scope of this document. Interested readers are referred to the website <http://www.schemers.org/> for more information on Scheme.

4.3.2 Reals

Formed from an optional minus sign and a sequence of digits followed by a *required* decimal point and an optional exponent such as `-1.2e3`. Reals can be built up using the usual operations: `+`, `-`, `*`, and `/`, with parentheses for grouping.

A real constant can be followed by one of the dimension keywords: `\mm` `\pt`, `\in`, or `\cm`, for millimeters, points, inches and centimeters, respectively. This converts the number that is the internal representation of that dimension.

4.3.3 Strings

Begins and ends with the `"` character. To include a `"` character in a string write `\"`. Various other backslash sequences have special interpretations as in the C language. A string that contains no spaces can be written without the quotes. Strings can be concatenated with the `+` operator.

4.4 Output details

LilyPond's default output format is `TeX`. Using the option `-f` (or `--format`) other output formats can be selected also, but currently none of them reliably work.

At the beginning of the output file, various global parameters are defined. It also contains a large `\special` call to define PostScript routines to draw items not representable with `TeX`, mainly slurs and ties. A DVI driver must be able to understand such embedded PostScript, or the output will be rendered incompletely.

Then the file `'lilyponddefs.tex'` is loaded to define the macros used in the code which follows. `'lilyponddefs.tex'` includes various other files, partially depending on the global parameters.

Now the music is output system by system (a `'system'` consists of all staves belonging together). From `TeX`'s point of view, a system is an `\hbox` which contains a lowered `\vbox` so that it is centered vertically on the baseline of the text. Between systems, `\interscoreline` is inserted vertically to have stretchable space. The horizontal dimension of the `\hbox` is given by the `linewidth` parameter from LilyPond's `\paper` block (using the natural line width if its value is `-1`).

After the last system LilyPond emits a stronger variant of `\interscoreline` only if the macro `\lilypondpaperlastpagefill` is not defined (flushing the systems to the top of the page). You can avoid that manually by saying

```
\def\lilypondpaperlastpagefill{1}
```

or by setting the variable `lastpagefill` in LilyPond's `\paper` block.

It is possible to fine-tune the vertical offset further by defining the macro `\lilypondscoreshift`. Example:

```
\def\lilypondscoreshift{0.25\baselineskip}
```

`\baselineskip` is the distance from one text line to the next.

The code produced by LilyPond can be used by both \TeX and \LaTeX .

Here an example how to embed a small LilyPond file `foo.ly` into running \LaTeX text without using the `lilypond-book` script (see Chapter 6 [Integrating text and music with `lilypond-book`], page 136).

```
\documentclass{article}

\def\lilypondpaperlastpagefill{}
\lineskip 5pt
\def\lilypondscoreshift{0.25\baselineskip}

\begin{document}
This is running text which includes an example music file
\input{foo.tex}
right here.
\end{document}
```

The file 'foo.tex' has been simply produced with

```
lilypond foo.ly
```

It is important to set the `indent` parameter to zero in the `\paper` block of 'foo.ly'.

The call to `\lineskip` assures that there is enough vertical space between the LilyPond box and the surrounding text lines.

5 Invoking LilyPond

Usage:

```
lilypond [option]... file...
```

When invoked with a filename that has no extension, LilyPond will try to add ‘.ly’ as an extension first. To have LilyPond read from stdin, use a dash - for *file*.

When LilyPond processes ‘filename.ly’ it will produce ‘filename.tex’ as output (or ‘filename.ps’ for PostScript output). If ‘filename.ly’ contains more than one \score block, then LilyPond will output the rest in numbered files, starting with ‘filename-1.tex’. Several files can be specified; they will each be processed independently.¹

5.1 Command line options

The following options are supported:

-e, --evaluate=*expr*

Evaluate the Scheme *expr* before parsing any ‘.ly’ files. Multiple -e options may be given, they will be evaluated sequentially. The function `ly-set-option` allows for access to some internal variables. Use -e ‘(ly-option-usage)’ for more information.

-f, --format=*format*

Output format for sheet music. Choices are `tex` (for T_EX output, to be processed with plain T_EX, or through ly2dvi), `pdftex` for PDF_TE_X input, `ps` (for PostScript), `scm` (for a Scheme dump), `sk` (for Sketch) and `as` (for ASCII-art).

This option is only for developers. Only the T_EX output of these is usable for real work. More information can be found at <http://lilypond.org/wiki?OutputFormats>.

-h, --help

Show a summary of usage.

--include, -I=*directory*

Add *directory* to the search path for input files.

-i, --init=*file*

Set init file to *file* (default: ‘init.ly’).

-m, --no-paper

Disable T_EX output. If you have a \midi definition midi output will be generated.

-M, --dependencies

Output rules to be included in Makefile.

-o, --output=*FILE*

Set the default output file to *FILE*.

¹ The status of GUILF is not reset across invocations, so be careful not to change any default settings from within Scheme .

`-v,--version`
 Show version information

`-V,--verbose`
 Be verbose: show full paths of all files read, and give timing information.

`-w,--warranty`
 Show the warranty with which GNU LilyPond comes. (It comes with **NO WARRANTY!**)

5.2 Environment variables

For processing both the T_EX and the PostScript output, you must have appropriate environment variables set. The following scripts do this:

- ‘buildscripts/out/lilypond-profile’ (for sh shells)
- ‘buildscripts/out/lilypond-login’ (for C-shells)

They should normally be sourced as part of your login process. If these scripts are not run from the system wide login process, then you must run it yourself.

If you use sh, bash, or a similar shell, then add the following to your ‘.profile’

```
. lilypond-profile
```

If you use csh, tcsh or a similar shell, then add the following to your ‘~/.login’

```
source lilypond-login
```

These scripts set the following variables

TEXMF To make sure that T_EX and lilypond find data files (among others ‘.tex’, ‘.mf’ and ‘.tfm’), you have to set **TEXMF** to point to the lilypond data file tree. A typical setting would be

```
{/usr/share/lilypond/1.6.0,{!!/usr/share/texmf}}
```

GS_LIB For processing PostScript output (obtained with `-f ps`) with Ghostscript you have to set **GS_LIB** to point to the directory containing LilyPond PS files.

GS_FONTPATH

For processing PostScript output (obtained with `-f ps`) with Ghostscript you have to set **GS_FONTPATH** to point to the directory containing LilyPond PFA files.

When you print direct PS output, remember to send the PFA files to the printer as well.

The LilyPond binary itself recognizes the following environment variables

LILYPONDPREFIX

This specifies a directory where locale messages and data files will be looked up by default. The directory should contain subdirectories called ‘ly/’, ‘ps/’, ‘tex/’, etc.

LANG This selects the language for the warning messages of LilyPond.

5.3 Reporting bugs

Since there is no finder's fee which doubles every year, there is no need to wait for the prize money to grow. So send a bug report today!

LilyPond development moves quickly, so if you have a problem, it is wise to check if it has been fixed in a newer release. If you think you found a bug, please send in a bugreport. When you send us a bugreport, we have to diagnose the problem and if possible, duplicate it. To make this possible, it is important that you include the following information in your report:

- A sample input which causes the error. Please have mercy on the developers, send a *small* sample file.
- The version number of lilypond.
- A description of the platform you use (i.e., operating system, system libraries, whether you downloaded a binary release)
- If necessary, send a description of the bug itself. If you include output a ly2dvi run, please use `--verbose` option of ly2dvi.

You can send the report to `bug-lilypond@gnu.org`. This is a mailinglist, but you don't have to be subscribed to it to post.

5.4 Website

If you are reading this manual in print, it is possible that the website contains updates to the manual. You can find the lilypond website at <http://www.lilypond.org/>.

5.5 Invoking ly2dvi

Nicely titled output is created through a separate program: 'ly2dvi' is a script that uses LilyPond and LaTeX to create a nicely titled piece of sheet music, in DVI format or PostScript.

```
ly2dvi [option]... file...
```

To have ly2dvi read from stdin, use a dash - for *file*.

Ly2dvi supports the following options:

`-k, --keep`

Keep the temporary directory including LilyPond and ly2dvi output files. The temporary directory is created in the current directory as `ly2dvi.dir`.

`-d, --dependencies`

Write makefile dependencies for every input file.

`-h, --help`

Print usage help.

`-I, --include=dir`

Add *dir* to LilyPond's include path.

- `-m, --no-paper`
Produce MIDI output only.
- `--no-lily`
Do not run LilyPond; useful for debugging ly2dvi.
- `-o, --output=file`
Generate output to *file*. The extension of *file* is ignored.
- `-P, --postscript`
Also generate PostScript output, using dvips. The postscript uses the standard TeX bitmap fonts for your printer.
- `-p, --pdf` Also generate Portable Document Format (PDF). This option will generate a PS file using scalable fonts, and will run the PS file through `ps2pdf` producing a PDF file.

If you use `lilypond-book` or your own wrapper files, don't use `\usepackage[[T1]{fontenc}` in the file header but don't forget `\usepackage[latin1]{inputenc}` if you use any other non-anglosaxian characters.
- `--preview`
Also generate a picture of the first system of the score.
- `-s, --set=key=val`
Add `key= val` to the settings, overriding those specified in the files. Possible keys: `language`, `latexheaders`, `latexpackages`, `latexoptions`, `papersize`, `pagenumber`, `linewidth`, `orientation`, `textheight`.
- `-v, --version`
Show version information
- `-V, --verbose`
Be verbose
- `-w, --warranty`
Show the warranty with which GNU LilyPond comes. (It comes with **NO WARRANTY!**)

5.5.1 Titling layout

Ly2dvi extracts the following header fields from the LY files to generate titling:

- `title` The title of the music. Centered on top of the first page.
- `subtitle` Subtitle, centered below the title.
- `poet` Name of the poet, left flushed below the subtitle.
- `composer` Name of the composer, right flushed below the subtitle.
- `meter` Meter string, left flushed below the poet.
- `opus` Name of the opus, right flushed below the composer.

arranger	Name of the arranger, right flushed below the opus.
instrument	Name of the instrument, centered below the arranger
dedication	[docme]
piece	Name of the piece, left flushed below the instrument
head	A text to print in the header of all pages. It is not called header , because \header is a reserved word in LilyPond.
copyright	A text to print in the footer of the first page. Default is to print the standard footer also on the first page.
footer	A text to print in the footer of all but the last page.
tagline	Line to print at the bottom of last page. The default text is “Lily was here, <i>version-number</i> ”.

5.5.2 Additional parameters

Ly2dvi responds to several parameters specified in a **\paper** section of the LilyPond file. They can be overridden by supplying a **--set** command line option.

language	Specify LaTeX language: the babel package will be included. Default: unset. Read from the \header block.
latexheaders	Specify additional LaTeX headers file. Normally read from the \header block. Default value: empty
latexpackages	Specify additional LaTeX packages file. This works cumulative, so you can add multiple packages using multiple -s=latexpackages options. Normally read from the \header block. Default value: geometry .
latexoptions	Specify additional options for the LaTeX \documentclass . You can put any valid value here. This was designed to allow ly2dvi to produce output for double-sided paper, with balanced margins and pagenumbers on alternating sides. To achieve this specify twoside
orientation	Set orientation. Choices are portrait or landscape . Is read from the \paper block, if set.
textheight	The vertical extension of the music on the page. It is normally calculated automatically, based on the paper size.

linewidth	The music line width. It is normally read from the <code>\paper</code> block.
papersize	The paper size (as a name, e.g. <code>a4</code>). It is normally read from the <code>\paper</code> block.
pagenumber	If set to <code>no</code> , no page numbers will be printed. If set to a positive integer, start with this value as the first page number.
fontenc	The font encoding, should be set identical to the <code>font-encoding</code> property in the score.

5.5.3 Environment variables

LANG	selects the language for the warning messages of Ly2dvi and LilyPond.
GUILE_MAX_SEGMENT_SIZE	is an option for GUILE, the scheme interpreter; it sets the size of the chunks of memory allocated by GUILE. By increasing this from its default 8388608, the performance of LilyPond on large scores is slightly improved.

6 Integrating text and music with lilypond-book

If you want to add pictures of music to a document, you can simply do it the way you would do with other types of pictures. You write LilyPond code, process it separately to embedded PostScript or `png`, and include it as a picture into your LaTeX or `html` source.

`lilypond-book` provides you with a way to automate this process: This program extracts snippets of music from your document, runs LilyPond on them, and substitutes the resulting pictures back. The line width and font size definitions for the music are adjusted to match the layout of your document.

It can work on LaTeX, `html` or `texinfo` documents. A tutorial on using `lilypond-book` is in Section 2.11 [Integrating text and music], page 42.

6.1 Integrating Texinfo and music

You specify the LilyPond code like this:

```
@lilypond[options, go, here]
  YOUR LILYPOND CODE
@end lilypond
@lilypond[options, go, here]{ YOUR LILYPOND CODE }
@lilypondfile[options, go, here]{filename}
```

We show two simple examples here. First a complete block:

```
@lilypond[26pt]
  c' d' e' f' g'2 g'
@end lilypond
```

produces this music:



Then the short version:

```
@lilypond[11pt]{<c' e' g'>}
```

and its music:



`lilypond-book` knows the default margins and a few paper sizes. One of these commands should be in the beginning of the document:

- `@afourpaper`
- `@afourlatex`
- `@afourwide`
- `@smallbook`

`@pagesizes` are not yet supported.

When producing `texinfo`, `lilypond-book` also generates bitmaps of the music, so you can make a HTML document with embedded music.

6.2 Integrating LaTeX and music

You specify LilyPond code like this:

```
\begin[options, go, here]{lilypond}
  YOUR LILYPOND CODE
\end{lilypond}
\lilypondfile[options, go,here]{filename}
```

or

```
\lilypond{ YOUR LILYPOND CODE }
```

We show some examples here.

```
\begin[26pt]{lilypond}
  c' d' e' f' g'2 g'2
\end{lilypond}
```

produces this music:



Then the short version:

```
\lilypond[11pt]{<c' e' g'>}
```

and its music:



You can use whatever commands you like in the document preamble, the part of the document before `\begin{document}`. `lilypond-book` will send it to LaTeX to find out how wide the text is and adjust the linewidth variable in the paper definition of your music according to that.

After `\begin{document}` you must be a little more careful when you use commands that change the width of the text and how many columns there are. `lilypond-book` knows about the `\onecolumn` and `\twocolumn` commands and the `multicols` environment from the `multicol` package.

The music will be surrounded by `\preLilypondExample` and `\postLilypondExample`. The variables are defined to nothing by default, and the user can redefine them to whatever he wants.

6.3 Integrating HTML and music

You specify LilyPond code like this:

```
<lilypond relative1 verbatim>
  \key c \minor r8 c16 b c8 g as c16 b c8 d | g,4
```

```

</lilypond>
produces
<lilypond relative1 verbatim>
  \key c \minor r8 c16 b c8 g as c16 b c8 d | g,4
</lilypond>

```



Inline picture:

Some music in <lilypond a b c/> a line of text.

6.4 Music fragment options

The commands for lilypond-book have room to specify options. These are all of the options:

eps This will create the music as eps graphics and include it into the document with the `\includegraphics` command. It works in LaTeX only.

This enables you to place music examples in the running text (and not in a separate paragraph). To avoid that LaTeX places the music on a line of its own, there should be no empty lines between the normal text and the LilyPond environment. For inline music, you probably also need a smaller music font size (e.g. 11 pt or 13 pt)

verbatim CONTENTS is copied into the source enclosed in a verbatim block, followed by any text given with the `intertext` option, then the actual music is displayed. This option does not work with the short version of the LilyPond blocks:

```
@lilypond{ CONTENTS } and \lilypond{ CONTENTS }
```

smallverbatim

Like `verbatim`, but in a smaller font.

intertext="text"

Used in conjunction with `verbatim` option: This puts *text* between the code and the music (without indentation).

filename="filename"

Save the LilyPond code to *filename*. By default, a hash value of the code is used.

11pt



13pt



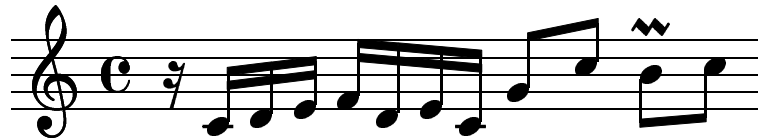
16pt



20pt



26pt

**singleline**

Produce a single, naturally spaced, unjustified line (i.e., linewidth = -1).

multilineThe opposite of **singleline**: Justify and break lines.**linewidth=***sizeunit*Set linewidth to *size*, where *unit* = cm, mm, in, or pt. This option affects LilyPond output, not the text layout.**notime**

Don't print time signature.

fragment**nofragment**

Override lilypond-book auto detection of what type of code is in the LilyPond block, voice contents or complete code.

indent=*sizeunit*Set indentation of the first music system to *size*, where *unit* = cm, mm, in, or pt. This option affects LilyPond, not the text layout. For single-line fragments the default is to use no indentation.**noindent**

Set indentation of the first music system to zero. This option affects LilyPond, not the text layout.

noquote

By default, lilypond-book puts both LaTeX and texinfo output into a quotation block. Using this option prevents this; no indentation will be used.

printfilenamePrints the file name before the music example. Useful in conjunction with `\lilypondfile`.

`relative`, `relative N`

Use relative octave mode. By default, notes are specified relative central C. The optional integer argument specifies how many octaves higher (positive number) or lower (negative number) to place the starting note.

6.5 Invoking lilypond-book

When you run `lilypond-book` it will generate lots of small files that LilyPond will process. To avoid all the garbage in your source directory, you should either change to a temporary directory, or use the `--outdir` command line options:

```
cd out && lilypond-book ../yourfile.tex
lilypond-book --outdir=out yourfile.tex
```

For LaTeX input, the file to give to LaTeX has extension `.latex`. Texinfo input will be written to a file with extension `.texi`.

If you use `--outdir`, you should also `cd` to that directory before running LaTeX or `makeinfo`. This may seem a little kludgy, but both LaTeX and `makeinfo` expect picture files (the music) to be in the current working directory. Moreover, if you do this, LaTeX will not clutter your normal working directory with output files.

If you want to add titling from the `\header` section of the files, you should add the following to the top of your LaTeX file:

```
\input titledefs.tex
\def\preLilypondExample{\def\mustmakelilypondtitle{}}
```

`lilypond-book` accepts the following command line options:

`'-f format', '--format=format'`

Specify the document type to process: `html`, `latex` or `texi` (the default). `lilypond-book` usually figures this out automatically.

Note that the `texi` document type produces a DVI file; to convert a texinfo document to `html`, you should use the additional format `texi-html` instead of `texi` to convert lilypond fragments to PNG images.

`'--default-music-fontsize=szpt'`

Set the fontsize to use for LilyPond if no fontsize is given as option.

`'--force-music-fontsize=szpt'`

Force all LilyPond code to use this fontsize, overriding options given to `\begin{lilypond}`.

`'-I dir', '--include=dir'`

Add *DIR* to the include path.

`'-M', '--dependencies'`

Write dependencies to `'filename.dep'`.

`'--dep-prefix=pref'`

Prepend *pref* before each `'-M'` dependency.

`'-n', '--no-lily'`

Don't run LilyPond, but do generate the `.ly` files.

```

'--no-music'
    Strip all LilyPond blocks from the file.

'--no-pictures'
    Don't generate pictures when processing Texinfo.

'--read-lys'
    Don't write ly files. This way you can do
        lilypond-book file.tely
        convert-ly
        lilypond-book --read-lys

'--outname=file'
    The name of LaTeX file to output. If this option is not given, the output name
    is derived from the input name.

'--outdir=dir'
    Place generated files in dir.

'--version'
    Print version information.

'--help'  Print a short help message.

```

6.6 Bugs

The LaTeX `\includeonly{...}` command is ignored.

The Texinfo command `pagesize` is on the TODO list for LilyPond 1.8, but changing the linewidth in other ways will not give you a straight right margin.

Almost all LaTeX commands that change margins and line widths are ignored.

There is no way to automatically apply `convert-ly` only to fragments inside a lilypond-book file.

`lilypond-book` processes all music fragments in one big run. The state of the GUILF interpreter is not reset between fragments; this means that global GUILF definitions, e.g., done with `#(define ...)` and `#(set! ...)` can leak from one fragment into the next fragment.

7 Converting from other formats

7.1 Invoking convert-ly

Convert-ly sequentially applies different conversions to upgrade a Lilypond input file. It uses `\version` statements in the file to detect the old version number. For example, to upgrade all lilypond files in the current directory and its subdirectories, use

```
convert-ly -e --to=1.3.150 'find . -name '*.ly' -print'
```

The program is invoked as follows:

```
convert-ly [option]... file...
```

The following options can be given:

`-a, --assume-old`

If version number cannot be determined, apply all conversions.

`-e, --edit`

Do an inline edit of the input file. Overrides `--output`.

`-f, --from=from-patchlevel`

Set the level to convert from. If this is not set, convert-ly will guess this, on the basis of `\version` strings in the file.

`-o, --output=file`

Set the output file to write.

`-n, --no-version`

Normally, convert-ly adds a `\version` indicator to the output. Specifying this option suppresses this.

`-s, --show-rules`

Show all known conversions and exit.

`--to=to-patchlevel`

Set the goal version of the conversion. It defaults to the latest available version.

`-h, --help`

Print usage help

BUGS

Not all language changes are handled. Only one output options can be specified.

7.2 Invoking midi2ly

Midi2ly translates a MIDI input file to a LilyPond source file. MIDI (Music Instrument Digital Interface) is a standard for digital instruments: it specifies cabling, a serial protocol and a file format.

The MIDI file format is a de facto standard format for exporting music from other programs, so this capability may come in useful when you want to import files from a program that has no converter for its native format.

It is possible to record a MIDI file using a digital keyboard, and then convert it to '.ly'. However, human players are not rhythmically exact enough to make a MIDI to LY conversion trivial. midi2ly tries to compensate for these timing errors, but is not very good at this. It is therefore not recommended to use midi2ly for human-generated midi files.

Hackers who know about signal processing are invited to write a more robust midi2ly. midi2ly is written in Python, using a module written in C to parse the MIDI files.

It is invoked as follows:

```
    midi2ly [option]... midi-file
```

The following options are supported by midi2ly:

- b, --no-quantify,
Write exact durations, e.g.: 'a4*385/384'.
- D, --debug,
Print lots of debugging stuff.
- h, --help,
Show a summary of usage.
- I, --include=*dir*,
Add *dir* to search path.
- k, --key=*acc[:minor]*,
Set default key. *acc* > 0 sets number of sharps; *acc* < 0 sets number of flats. A minor key is indicated by ":1".
- n, --no-silly,
Assume no plets or double dots, assume smallest (reciprocal) duration 16.
- o, --output=*file*,
Set *file* as default output.
- p, --no-plets,
Assume no plets.
- q, --quiet,
Be quiet.
- s, --smallest=*N*,
Assume no shorter (reciprocal) durations than *N*.
- v, --verbose,
Be verbose.
- w, --warranty,
Show the warranty with which midi2ly comes. (It comes with **NO WARRANTY!**)
- x, --no-double-dots,
Assume no double dotted notes.

7.3 Invoking etf2ly

ETF (Enigma Transport Format) is a format used by Coda Music Technology’s Finale product. etf2ly will convert part of an ETF file to a ready-to-use LilyPond file.

It is invoked as follows:

```
etf2ly [option]... etf-file
```

The following options are supported by etf2ly.

-h,--help

this help

-o,--output=FILE

set output filename to FILE

-v,--version

version information

BUGS

The list of articulation scripts is incomplete. Empty measures confuse etf2ly.

7.4 Invoking abc2ly

ABC is a fairly simple ASCII based format. It is described at the abc site:

<http://www.gre.ac.uk/~c.walshaw/abc2mtex/abc.txt>

abc2ly translates from ABC to LilyPond. It is invoked as follows:

```
abc2ly [option]... abc-file
```

The following options are supported by abc2ly:

-h,--help

this help

-o,--output=file

set output filename to *file*.

-v,--version

print version information.

There is a rudimentary facility for adding lilypond code to the ABC source file. If you say:

```
%%LY voices \property Voice.autoBeaming=##f
```

This will cause the text following the keyword “voices” to be inserted into the current voice of the lilypond output file.

Similarly:

```
%%LY slyrics more words
```

will cause the text following the “slyrics” keyword to be inserted into the current line of lyrics.

BUGS

The ABC standard is not very “standard”. For extended features (eg. polyphonic music) different conventions exist.

Multiple tunes in one file cannot be converted.

ABC synchronizes words and notes at the beginning of a line; abc2ly does not.

abc2ly ignores the ABC beaming.

7.5 Invoking pmx2ly

PMX is a MusiXTeX preprocessor written by Don Simons. More information on PMX is available from the following site:

<http://icking-music-archive.sunsite.dk/Misc/Music/musixtex/software/pmx/>. ■

pmx2ly converts from PMX to LilyPond input. The program is invoked as follows:

```
pmx2ly [option]... pmx-file
```

The following options are supported by pmx2ly:

```
-h,--help          this help
-o,--output=FILE   set output filename to FILE
-v,--version       version information
```

7.6 Invoking musedata2ly

Musedata (<http://www.musedata.org/>) is an electronic library of classical music scores, currently comprising about 800 composition dating from 1700 to 1825. The music is encoded in so-called Musedata format. musedata2ly converts a set of musedata files to one .ly file, and will include a \header field if a ‘.ref’ file is supplied. It is invoked as follows:

```
musedata2ly [option]... musedata-files
```

The following options are supported by musedata2ly:

```
-h,--help          print help
-o,--output=file   set output filename to file
-v,--version       version information
-r,--ref=reffile   read background information from ref-file ref file
```

BUGS

musedata2ly converts only a small subset musedata.

7.7 Invoking mup2ly

MUP (Music Publisher) is a shareware music notation program by Arkkra Enterprises. It is also the name of the input format. Mup2ly will convert part of a Mup file to a ready-to-use LilyPond file. Mup2ly is invoked as follows:

```
mup2ly [option]... mup-file
```

The following options are supported by mup2ly:

```
-d,--debug
    show what constructs are not converted, but skipped.

D, --define=name [=exp]
    define macro name with opt expansion exp

-E,--pre-process
    only run the pre-processor

-h,--help
    print help

-o,--output=file
    write output to file

-v,--version
    version information

-w,--warranty
    print warranty and copyright.
```

BUGS

Currently, only plain notes (pitches, durations), voices and staves are converted.

8 Literature

If you need to know more about music notation, here are some interesting titles to read. The source archive includes a more elaborate BibTeX bibliography of over 100 entries in ‘Documentation/bibliography/’. It is also available online from the lilypond website.

Banter 1987

Harald Banter, Akkord Lexikon. Schott’s Söhne 1987. Mainz, Germany ISBN 3-7957-2095-8

Comprehensive overview of commonly used chords. Suggests (and uses) a unification for all different kinds of chord names.

Gerou 1996

Tom Gerou and Linda Lusk, Essential Dictionary of Music Notation. Alfred Publishing, Van Nuys CA ISBN 0-88284-768-6

A concise, alphabetically ordered list of typesetting and music (notation) issues which covers most of the normal cases.

Hader 1948,

Karl Hader, Aus der Werkstatt eines Notenstechers. Waldheim–Eberle Verlag, Vienna 1948.

Hader was the chief-engraver of the Waldheim-Eberle music publishers. This beautiful booklet was intended as an introduction for laymen on the art of engraving. It contains a step by step, in-depth explanation of how to cut and stamp music into zinc plates. It also contains a few compactly formulated rules on musical orthography. This book is out of print.

Read 1968

Gardner Read, Music Notation: a Manual of Modern Practice. Taplinger Publishing, New York (2nd edition).

A standard work on music notation.

Ross 1987,

Ted Ross, Teach yourself the art of music engraving and processing. Hansen House, Miami, Florida 1987

This book is about music engraving, i.e. professional typesetting. It contains directions on stamping, use of pens and notational conventions. The sections on reproduction technicalities, and history are also interesting.

Stone 1980

Kurt Stone, Music Notation in the Twentieth Century Norton, New York 1980.

This book describes music notation for modern serious music, but starts out with a thorough overview of existing traditional notation practices.

Wanske 1988,

Helene Wanske, Musiknotation — Von der Syntax des Notenstichs zum EDV-gesteuerten Notensatz. Schott-Verlag, Mainz 1988. ISBN 3-7957-2886-x

A book in two parts: 1. A very thorough overview of engraving practices of various craftsmen. It includes detailed specs of characters, dimensions etc. 2.

a thorough overview of a anonymous (by now antiquated) automated system. EDV means E(lektronischen) D(aten)v(erarbeitung), electronic data processing.

Index

,		\context	118
,	46	\cr	73
(\decr	73
(begin * * * *)	61	\defaultAccidentals	63
(end * * * *)	61	\dorian	55
,		\duration	47
,	46	\dynamicDown	74
.		\dynamicUp	74
.	48	\f	73
/		\ff	73
/	89	\fff	73
/+	89	\ffff	73
<		\forgetAccidentals	65
<	124	\fp	73
>		\glissando	73
>	124	\grace	28, 70
?		\header	24
?	46	\header in LaTeX documents	140
[\in	128
[60	\ionian	55
]		\key	55
]	60	\locrian	55
\		\lydian	55
\!	30	\lyrics	17, 127
\"!	73	\major	55
\<	30, 73	\mark	92
\>	73	\mf	73
\aeolian	55	\minor	55
\alternative	74	\mixolydian	55
\arpeggio	83	\mm	128
\bar	57	\modernAccidentals	64
\chords	127	\modernCautionaries	64
\clef	55	\modernVoiceAccidentals	64
\cm	128	\modernVoiceCautionaries	64
		\mp	73
		\noResetKey	65
		\notes	52, 127
		\once	107
		\outputproperty	107
		\override	105
		\p	73
		\partial	19, 56
		\phrygian	55
		\pianoAccidentals	65
		\pianoCautionaries	65
		\pitch	45
		\pp	73
		\ppp	73
		\property	21
		\property	120
		\pt	128
		\rc	73
		\rced	73
		\relative	51
		\repeat	74

<code>\revert</code>	105
<code>\rfz</code>	73
<code>\script</code>	69
<code>\sequential</code>	124
<code>\set</code>	105
<code>\sf</code>	73
<code>\sff</code>	73
<code>\sfz</code>	73
<code>\simultaneous</code>	20, 124
<code>\sp</code>	73
<code>\spp</code>	73
<code>\start</code>	126
<code>\stop</code>	126
<code>\tempo</code>	67
<code>\time</code>	56
<code>\times</code>	50
<code>\translator</code>	29
<code>\transpose</code>	94
<code>\unset</code>	120
<code>\voiceAccidentals</code>	63

|

.....	52, 57
-------	--------

~

~	29, 48
---------	--------

A

<code>A2_engraver</code>	96
<code>ABC</code>	144
accent	69
accessing Scheme	127
Accidentals	63
adjusting output	5
Adjusting slurs	67
adjusting staff symbol	55
ambiguities	127
anacrusis	19, 56
<code>Arkkra</code>	146
arpeggio	28
<code>Arpeggio</code>	83
<code>Arpeggio</code>	84
articulations	68
Articulations	68
ASCII-art output	130
assignments	24
<code>Assignments</code>	126
<code>aug</code>	88
auto-knee-gap	61
autobeam	62
<code>autoBeamSettings</code>	61
automatic beam generation	62
automatic beaming, turning off	19
Automatic beams	60
automatic beams, tuning	61

automatic part combining	94
Automatic staff changes	81

B

balance	3
bar check	52
Bar check	52
Bar lines	57
bar numbers	92
<code>Bar_line_engraver</code>	58
<code>barCheckSynchronize</code>	52
<code>BarLine</code>	58
barlines, putting symbols on	92
<code>BarNumber</code>	93
<code>base-shortest-duration</code>	113
<code>BassFigure</code>	104
Basso continuo	104
<code>Batch</code>	2
<code>Beam</code>	60, 76
beams, kneed	61
beams, manual	60
beats per minute	67
between staves, distance	112
blackness	3
breaking lines	114
breaking pages	114
<code>BreathingSign</code>	67
broken arpeggio	83
bugreport	5
bugs	131

C

cautionary accidental	46
<code>CCARH</code>	145
<code>ChoirStaff</code>	118
chord modifier	20
chord names	89
<code>ChordNames</code>	89
chords	20, 89
<code>Chords</code>	87
Chords mode	88
chords, starting with	120
Clef	56
coda	69, 92
Coda Technology	144
command line options	130
<code>common-shortest-duration</code>	113
concatenate	128
condensing rests	94
context	21
context definition	115, 121
context selection	118
context variables	21
craftsmanship	3
crescendo	30, 73
cross staff	84

cross staff voice	29
cross staff voice, manual	29
cue notes	108
Current	120
currentBarNumber	92
Custodes	101
Custos	101

D

decrescendo	73
defaultBarType	58
Denemo	51
dim	88
dimensions	128
diminuendo	74
direction, of dynamics	74
distance between staves	112
documents, adding music to	136
DoublePercentRepeat	77
downbow	69
drums	78
duration	47
DVI driver	9
DVI file	8
dvilj	9
dvips	9
DynamicLineSpanner	33, 74
dynamics	30
Dynamics	73
DynamicText	74

E

easy notation	50
editor	53
emacs	53
Emacs	53
emacs mode	53
encoding music	2
engraver	117, 121
Engraver_group_engraver	122
engraving	2
enigma	144
entering notes	45
ETF	144
evaluating Scheme	127
expanding repeats	75
explicit context	120
extending lilypond	5
extra-offset	32, 107

F

FDL, GNU Free Documentation License	163
fermata	69
fermatas	92
FiguredBass	104
file searching	130
Finale	144
finding graphical objects	31
fingering	69
Fingering	69
fingering instructions	31
flageolet	69
follow voice	84
followVoice	84
font	3
font magnification	109
font selection	109
font size	108
font size, setting	114
font-interface	109
font-style	109
foot marks	69
footer	134
forte	30
four bar music	114
free software	2

G

ghostscript	51, 131
Ghostsript	9
Glissando	73
grace notes	28, 70
grace slash	71
grammar	127
GrandStaff	65
graphical interface	51
graphical object descriptions	31
grob properties	33
GS_FONTPATH	131
GS_LIB	131
GUI	2, 51
GUILE	127
GVim	53

H

Hairpin	74
Hal Leonard	50
hara kiri	61
header	134
html	136
HTML, music in	42
hufnagel	97

I

identifier assignment	20, 24
Identifiers	123
idiom	5
indent	114
index	5
input format	4
input mode	127
installing LilyPond	131
instrument names	116
internal documentation	5, 31
interpretation context	21
interpreting music	117
interscoreline	114
interscorelinefill	114
Invisible rest	47
Invoking LilyPond	130

K

KDE	53
KDVI	53
Key	55
keySignature	55
KeySignature	55
knead beams	61

L

LANG	131
lastpagefill	115
latex	136
LaTeX, music in	42
Lexical modes	127
LigatureBracket	102, 103
Ligatures	102
Lily was here	24
lilypond-book and titling	140
lilypond-internals	5
lilypond-mode for emacs	53
LILYPONDPREFIX	131
line breaks	114
line-column-location	54
line-location	53
linewidth	114
LISP	127
loudness	30
lpr	9
ly2dvi	8
lyric mode	17
lyrics	17, 20, 62

M

magnification	109
maj	88
manual beaming	19
manual staff switches	82
marcato	69
mark	92
markup text	109
master	3
measure lines	57
measure numbers	92
measure repeats	77
measure, partial	56
Medicaea, Editio	97
mensural	97
Mensural ligatures	103
MensuralLigature	103
meter	56
metronome mark	111
metronome marking	67
MIDI	51, 130, 142
MIDI block	115
mode, chords	20
mode, input	127
modifier, chord	20
mordent	69
Multi measure rests	94
MultiMeasureRest	94
MUP	146
Musedata	145
Music entry	51
music expressions	4, 124
music properties	33
Music Publisher	146
music representation	4
musical symbols	3
MusiXTeX	145

N

named modifier	20
NEdit	53, 54
neutral-direction	61
Non-guitar tablatures	86
NonMusicalPaperColumn	115
notation context	21
Note entry	45
note names, Dutch	45
Note specification	45
NoteCollision	60
NoteColumn	60
NoteEdit	51
NoteSpacing	113

O

object description	105
open	69
optical spacing	3
options, command line	130
organ pedal marks	69
ornaments	28, 68, 70
Ornaments	68
output format, setting	130
overview of manual	5

P

padding	93
page breaks	114
page layout	114, 134
page size	115
paper file	114
paper size	115
paper types, engravers, and pre-defined translators	123
papersize	115
parenthesized accidental	46
part combiner	94
Partial	56
partial measure	56
PDF	9, 133
PDFTeX output	130
Pedals	82
percent repeats	77
PercentRepeat	77
percussion	78
Petrucci	97
phrasing	22
phrasing marks	67
phrasing slurs	67
PhrasingSlur	67
PianoStaff	65, 81, 112
Pitch names	45
pitchs	45
PMX	145
poind and click	53
polyphony	58
portato	69
PostScript	9, 131
PostScript output	130
prall	69
prall, down	69
prall, up	69
prallmordent	69
prallprall	69
prebreaking	117
preprocessing	118
printing chord names	89
Printing output	9
printing postscript	131
properties	5
properties, unsetting	120

property types	33
'property-init.ly'	63

R

r	48
R	94
real numbers	128
regular line breaks	114
regular rhythms	3
regular spacing	3
Rehearsal marks	92
RehearsalMark	92
Relative	51
relative mode and transposing	29
relative octave specification	51
reminder accidental	46
repeat bars	57
repeatCommands	58
repeatCommands	76
repeats	74
RepeatSlash	77
reporting bugs	131
Rest	46
RestCollision	60
Rests	46
reverseturn	69
reverting object properties	33
RoseGarden	51

S

s	48
Scalable fonts	133
Scheme	5, 127
Scheme dump	130
Scheme, in-line code	127
Score	51, 56, 65, 92, 112, 118, 120
Score_engraver	122
Script	69
scripts	68, 69
search path	130
segno	69, 92
sequencer	51
sequential music	124
Sequential music	124
setting context variables	21
setting object properties	32
sharing software	2
shorten measures	56
signature line	24
Simons, Don	145
Simultaneous music	124
size	108
Sketch output	130
Skip	47
skipTypesetting	53
slash	71

Slur	32
Slur	66
slur attachments	31
Slurs	66
snippets	5
Sostenuto	82
Sound	115
Space note	47
spacing	113
SpacingSpanner	112, 113
Span requests	126
staccatissimo	69
staccato	69
Staff	55, 58, 64, 65, 96, 101, 108, 118, 123
staff distance	112
staff lines, setting number of	55
Staff notation	54
staff size, setting	114
staff switch	29
staff switch, manual	29, 82
staff switching	84
Staff.instrument	116
Staff.midiInstrument	116
StaffGroup	58, 93, 118
StaffSymbol	55
StaffSymbol, using \property	55
starting with chords	120
Stem	71, 105
stem-spacing-correction	113
stemBoth	29
stemLeftBeamCount	61
stemRightBeamCount	61
StemTremolo	76
stopped	69
string	128
subdivideBeams	61
subscript	69
superscript	69
sus	88
Sustain	82
switches	130
syllables, entering	20
Syntactic details	123
syntax coloring	53

T

Tablature in addition to normal staff	87
Tablatures basic	85
TabStaff	85, 86
TabVoice	85
tag line	24
Tempo	67
tenuto	69
texi	136
texinfo	136
Texinfo, music in	42
TEXMF	131

text markup	109
Text scripts	70
Text spanners	68
text-interface	110
textheight	114
TextScript	70
TextSpanner	68
Thread	65, 108, 120
Thread_devnull_engraver	96
thumb marking	69
tie	29
Tie	48
Tie	49
ties	48
Time signature	56
TimeSignature	56
titles	134
titling and lilypond-book	140
tonic	20
translator definition	121
translator properties	33
Transpose	94
transposing	29
transposing	97
transposition of pitches	94
tremolo beams	76
tremolo marks	77
tremoloFlags	77
trill	29, 69
triplets	28, 50
tuning automatic beaming	61
tuning graphical objects	31
tuplet	28
tuplet formatting	50
TupletBracket	50
tupletNumberFormatFunction	50
tuplets	50
turn	69
tutorial	5
typography	2

U

UnaCorda	82
undoing object properties	33
unfolded \repeat	29
UNIX	2
upbeat	56
upbow	69

V

variables	5
Vaticana, Editio	97
vertical spacing	112
VerticalAlignment	112
Viewing music	8
Vim	53
Voice 32, 58, 65, 66, 68, 70, 103, 108, 118, 120, 123	
Voice.autoBeaming	62
Voice_engraver	96
VoiceFollower	85
VoltaBracket	76

W

whichBar	58
White mensural ligatures	103
whole rests for a full measure	94

X

xdvi	8
Xdvi	51, 53
<i>XEDITOR</i>	54
XEmacs	53

Appendix A Refman appendix

A.1 Lyrics mode definition

The definition of lyrics mode is ludicrous, and this will remain so until the authors of LilyPond acquire a deeper understanding of character encoding, or someone else steps up to fix this.

A word in Lyrics mode begins with: an alphabetic character, `_`, `?`, `!`, `:`, `'`, the control characters `^A` through `^F`, `^Q` through `^W`, `^Y`, `^^`, any 8-bit character with ASCII code over 127, or a two-character combination of a backslash followed by one of `'`, `'`, `"`, or `^`.

Subsequent characters of a word can be any character that is not a digit and not white space. One important consequence of this is that a word can end with `}`, which may be confusing. However, LilyPond will issue a warning. Any `_` character which appears in an unquoted word is converted to a space. This provides a mechanism for introducing spaces into words without using quotes. Quoted words can also be used in Lyrics mode to specify words that cannot be written with the above rules.

A.2 American Chords

```
\include "english.ly"

scheme = \chords {
  c          % Major triad
  cs:m       % Minor triad
  df:m5-     % Diminished triad
  c:5^3      % Root-fifth chord
  c:4^3      % Suspended fourth triad
  c:5+       % Augmented triad
  c:2^3      % "2" chord
  c:m5-.7-   % Diminished seventh
  c:7+       % Major seventh
  c:7.4^3    % Dominant seventh suspended fourth
  c:5+.7     % Augmented dominant seventh
  c:m5-.7    % "Half" diminished seventh
  c:5-.7     % Dominant seventh flat fifth
  c:5-.7+    % Major seventh flat fifth
  c:m7+      % Minor-major seventh
  c:m7       % Minor seventh
  c:7        % Dominant seventh
  c:6        % Major sixth
  c:m6       % Minor sixth
  c:9^7      % Major triad w/added ninth
  c:6.9^7    % Six/Nine chord
  c:9        % Dominant ninth
  c:7+.9     % Major ninth
```

```

    c:m7.9    % Minor ninth
}

\score {
  \notes <
    \context ChordNames \scheme
    \context Staff \transpose c'' \scheme
  >
  \paper {
    linewidth = 5.7\in
    \translator {
      \ChordNamesContext
      ChordName \override #'word-space = #1
      ChordName \override #'style = #'american
    }
  }
}

```

C C_{♯m} D_{♭dim} C₅ C_{sus} C_{aug} C₂ C_{°7} C_{maj7} C_{7sus4} C_{aug7} C_{°7}



C_{7b5} C_{maj7b5} C_{m(maj7)} C_{m7} C₇ C₆ C_{m6} C_{add9} C_{6/9} C₉ C_{maj9} C_{m9}



A.3 Jazz chords

Similarly, Jazz style chord names are implemented as a variation on American style names:

```

scheme = \chords {
  % major chords
  c
  c:6 % 6 = major triad with added sixth
  c:maj % triangle = maj
  c:6.9^7 % 6/9
  c:9^7 % add9

  % minor chords
  c:m % m = minor triad
  c:m.6 % m6 = minor triad with added sixth
  c:m.7+ % m triangle = minor major seventh chord
  c:3-.6.9^7 % m6/9
  c:m.7 % m7
}

```

```

c:3-.9 % m9
c:3-.9^7 % madd9

% dominant chords
c:7 % 7 = dominant
c:7.5+ % +7 = augmented dominant
c:7.5- % 7b5 = hard diminished dominant
c:9 % 7(9)
c:9- % 7(b9)
c:9+ % 7(#9)
c:13^9.11 % 7(13)
c:13-^9.11 % 7(b13)
c:13^11 % 7(9,13)
c:13.9-^11 % 7(b9,13)
c:13.9+^11 % 7(#9,13)
c:13-^11 % 7(9,b13)
c:13-.9-^11 % 7(b9,b13)
c:13-.9+^11 % 7(#9,b13)

% half diminished chords
c:m5-.7 % slashed o = m7b5
c:9.3-.5- % o/7(pure 9)

% diminished chords
c:m5-.7- % o = diminished seventh chord
}

\score {
  \notes <
    \context ChordNames \scheme
    \context Staff \transpose c'' \scheme
  >
  \paper {
    linewidth = 5.7\in
    \translator {
      \ChordNamesContext
      ChordName \override #'word-space = #1
      ChordName \override #'style = #'jazz
    }
  }
}

```

C C₆ C[▲] C_{6/9} C_{add9} C_m C_{m6} C_m[▲] C_{m6}^{6135maj79} C_{m7} C_{m9} C_m^{add9}

Chord progression 1: C7, C^{aug}7, C⁷_{b5}, C⁹, C⁷^{maj}₇^b₉, C⁷^{maj}₇[#]₉, C⁷^{add}₁₃, C⁷^{add}₁₃^b

Chord progression 2: C⁹^{add}₁₃, C⁷^{add}₁₃, C⁷^{add}₁₃, C⁹^{add}₁₃^b, C⁷^{add}₁₃^b, C⁷^{add}₁₃^b, C⁷, C⁷, C⁷^{6135maj}₇⁹, C⁷














A.4 MIDI instruments

"acoustic grand"	"contrabass"	"lead 7 (fifths)"
"bright acoustic"	"tremolo strings"	"lead 8 (bass+lead)"
"electric grand"	"pizzicato strings"	"pad 1 (new age)"
"honky-tonk"	"orchestral strings"	"pad 2 (warm)"
"electric piano 1"	"timpani"	"pad 3 (polysynth)"
"electric piano 2"	"string ensemble 1"	"pad 4 (choir)"
"harpsichord"	"string ensemble 2"	"pad 5 (bowed)"
"clav"	"synthstrings 1"	"pad 6 (metallic)"
"celesta"	"synthstrings 2"	"pad 7 (halo)"
"glockenspiel"	"choir aahs"	"pad 8 (sweep)"
"music box"	"voice oohs"	"fx 1 (rain)"
"vibraphone"	"synth voice"	"fx 2 (soundtrack)"
"marimba"	"orchestra hit"	"fx 3 (crystal)"
"xylophone"	"trumpet"	"fx 4 (atmosphere)"
"tubular bells"	"trombone"	"fx 5 (brightness)"
"dulcimer"	"tuba"	"fx 6 (goblins)"
"drawbar organ"	"muted trumpet"	"fx 7 (echoes)"
"percussive organ"	"french horn"	"fx 8 (sci-fi)"
"rock organ"	"brass section"	"sitar"
"church organ"	"synthbrass 1"	"banjo"
"reed organ"	"synthbrass 2"	"shamisen"
"accordion"	"soprano sax"	"koto"
"harmonica"	"alto sax"	"kalimba"
"concertina"	"tenor sax"	"bagpipe"
"acoustic guitar (nylon)"	"baritone sax"	"fiddle"
"acoustic guitar (steel)"	"oboe"	"shanai"
"electric guitar (jazz)"	"english horn"	"tinkle bell"
"electric guitar (clean)"	"bassoon"	"agogo"
"electric guitar (muted)"	"clarinet"	"steel drums"
"overdriven guitar"	"piccolo"	"woodblock"
"distorted guitar"	"flute"	"taiko drum"
"guitar harmonics"	"recorder"	"melodic tom"
"acoustic bass"	"pan flute"	"synth drum"
"electric bass (finger)"	"blown bottle"	"reverse cymbal"
"electric bass (pick)"	"skakuhachi"	"guitar fret noise"

























"fretless bass"	"whistle"	"breath noise"
"slap bass 1"	"ocarina"	"seashore"
"slap bass 2"	"lead 1 (square)"	"bird tweet"
"synth bass 1"	"lead 2 (sawtooth)"	"telephone ring"
"synth bass 2"	"lead 3 (calliope)"	"helicopter"
"violin"	"lead 4 (chiff)"	"applause"
"viola"	"lead 5 (charang)"	"gunshot"
"cello"	"lead 6 (voice)"	

A.5 The Feta font

The following symbols are available in the Feta font and may be accessed directly using text markup such as `g^#'(music "scripts-segno")`, see Section 3.16.4 [Text markup], page 109.

 rests-0	 rests-1	 rests-0o	 rests-1o	 rests-3	 rests-2	 rests-1	 rests-2
 rests-2classical	 rests-3	 rests-4	 rests-5	 rests-6	 rests-7	 rests-3neo_mensural	 rests-2neo_mensural
 rests-1neo_mensural	 rests-0neo_mensural	 rests-1neo_mensural	 rests-2neo_mensural				
 rests-3neo_mensural	 rests-4neo_mensural	 accidentals-1	 accidentals-0				
 accidentals-1	 accidentals-2	 accidentals-2	 accidentals-rightparen				
 accidentals-leftparen	 dots-dot	 noteheads-1	 noteheads-0				
 noteheads-1	 noteheads-2	 noteheads-0diamond	 noteheads-1diamond				
 noteheads-2diamond	 noteheads-0triangle	 noteheads-1triangle	 noteheads-2triangle				
 noteheads-0slash	 noteheads-1slash	 noteheads-2slash	 noteheads-0cross				
 noteheads-1cross	 noteheads-2cross	 noteheads-2xcircle	 noteheads-ledgerending				
 scripts-ufermata	 scripts-dfermata	 scripts-thumb	 scripts-sforzato				

 scripts-staccato	 scripts-ustaccatissimo	 scripts-dstaccatissimo	 scripts-tenuto
 scripts-uportato	 scripts-dportato	 scripts-umarcato	 scripts-dmarcato
 scripts-open	 scripts-stopped	 scripts-upbow	 scripts-downbow
 scripts-reverseturn	 scripts-turn	 scripts-trill	 scripts-upedalheel
 scripts-dpedalheel	 scripts-upedaltoe	 scripts-dpedaltoe	 scripts-flageolet
 scripts-segno	 scripts-coda	 scripts-rcomma	 scripts-lcomma
 scripts-arpeggio	 scripts-trill-element	 scripts-arpeggio-arrow-1	 scripts-arpeggio-arrow-1
 scripts-trilelement	 scripts-prall	 scripts-mordent	 scripts-prallprall
 scripts-prallmordent	 scripts-upprall	 scripts-downprall	 scripts-upmordent
 scripts-downmordent	 scripts-lineprall	 scripts-pralldown	 scripts-prallup
 flags-u3	 flags-u4	 flags-u5	 flags-u6
 flags-d3	 flags-ugrace	 flags-dgrace	 flags-d4
 flags-d5	 flags-d6	 flags-stem	 flags-dstem
 clefs-G	 clefs-G_change	 clefs-C	 clefs-C_change
 clefs-F	 clefs-F_change	 clefs-percussion	 clefs-percussion_change
 clefs-tab	 clefs-tab_change	 timesig-C4/4	 timesig-C2/2
 pedal-*	 pedal-	 pedal-d	 pedal-e
 pedal-	 pedal-P	 pedal-Ped	 accordion-accDiscant
 accordion-accDot	 accordion-accFreebase	 accordion-accStdbase	 accordion-accBayanbase

							
accordion-accSB	accordion-accBB		accordion-accOldEE	accordion-accOldEES			
							
solfa-0do	solfa-1do		solfa-2do	solfa-0re			
							
solfa-1re	solfa-2ro	solfa-0me	solfa-1me	solfa-2me	solfa-0fa	solfa-1fau	solfa-2fau
							
solfa-1fad	solfa-2fad	solfa-0la	solfa-1la	solfa-2la	solfa-0te	solfa-1te	solfa-2te

Appendix B 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. Secondly, 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.

B.0.1 ADDENDUM: How to use this License for your documents

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.