LilyPond

The music typesetter

Program usage

The LilyPond development team

Copyright © 1999–2008 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".

For LilyPond version 2.12.1

Table of Contents

1 Install 1
1.1 Precompiled binaries 1
Downloading 1
1.2 Compiling from source 1
1.2.1 Downloading source code 1
1.2.2 Requirements 1
Compilation 1
Running requirements 2
Requirements for building documentation 2
1.2.3 Building LilyPond 2
Compiling 3
Compiling for multiple platforms 3
Compiling outside the source tree 3
Useful `make` variables 3
1.2.4 Building documentation 3
Commands for building documentation 3
Building documentation without compiling LilyPond 4
1.2.5 Testing LilyPond 5
1.2.6 Problems 5
Bison 1.875 5
Solaris 6
FreeBSD 6
International fonts 6

2 Setup 7
2.1 Setup for specific Operating Systems 7
2.1.1 Setup for MacOS X 7
2.2 Text editor support 8
2.2.1 Emacs mode 8
2.2.2 Vim mode 8
2.2.3 jEdit 8
2.2.4 TexShop 8
2.2.5 TextMate 8
2.2.6 LilyKDE 9
2.3 Point and click 9

3 Running LilyPond 10
3.1 Normal usage 10
3.2 Command-line usage 10
3.2.1 Invoking `lilypond` 10
3.2.2 Command line options for `lilypond` 10
3.2.3 Environment variables 13
3.3 Error messages 14
3.4 Updating files with `convert-ly` 14
3.4.1 Command line options for `convert-ly` 15
3.4.2 Problems with `convert-ly` 15
3.5 Reporting bugs 16

4 `lilypond-book`: Integrating text and music 18
4.1 An example of a musicological document 18
4.2 Integrating music and text 21
4.2.1 LaTeX 21
4.2.2 Texinfo 22
4.2.3 HTML 23
4.2.4 DocBook 23
4.3 Music fragment options 24
4.4 Invoking `lilypond-book` 27
4.5 Filename extensions 29
4.6 Alternative methods of mixing text and music 30
Many quotes from a large score 30
Inserting LilyPond output into OpenOffice.org 30
Inserting LilyPond output into other programs 30

5 Converting from other formats 31
5.1 Invoking `midi2ly` 31
5.2 Invoking `musicxml2ly` 32
5.3 Invoking `abc2ly` 33
5.4 Invoking `etf2ly` 34
5.5 Generating LilyPond files 34

Appendix A GNU Free Documentation License 35

Appendix B LilyPond index 41

1 Install

There are two sets of releases for LilyPond: stable releases, and unstable development releases. Stable versions have an even-numbered 'minor' version number (i.e. 2.8, 2.10, 2.12, etc). Development versions have an odd-numbered 'minor' version number (i.e. 2.7, 2.9, 2.11, etc).

Building LilyPond is a very involved process, so we **highly** recommend using the precompiled binaries.

1.1 Precompiled binaries

Downloading

Check out http://lilypond.org/web/install/ for up to date information on binary packages for your platform. If your operating system is not covered on that general page, please see the complete list at http://download.linuxaudio.org/lilypond/binaries/

We currently create binaries for

```
darwin-ppc  - MacOS X powerpc
darwin-x86  - MacOS X intel
freebsd-64  - FreeBSD 6.x, x86_64
freebsd-x86 - FreeBSD 4.x, x86
linux-64    - Any GNU/Linux distribution, x86_64
linux-ppc   - Any GNU/Linux distribution, powerpc
linux-x86   - Any GNU/Linux distribution, x86
mingw       - Windows x86
```

Known issues and warnings

If you have MacOS 10.3 or 10.4 and you would like to use Python scripts such as `convert-ly` and `lilypond-book`, see Section "Setup for MacOS X" in *Application Usage*.

1.2 Compiling from source

1.2.1 Downloading source code

Download source

- tarballs from http://lilypond.org/download/ by HTTP.
- tarballs from http://download.linuxaudio.org/lilypond/ by HTTP.
- GIT from git.sv.gnu.org

  ```
  git clone git://git.sv.gnu.org/lilypond.git
  ```

  The repository does not contain generated files. To create '`configure`', run

  ```
  ./autogen.sh
  ```

For information on packaging, see http://lilypond.org/devel.

1.2.2 Requirements

Compilation

In addition to the packages needed for running LilyPond (see below), you need the following extra packages for building.

When installing a binary package FOO, you may need to install the FOO-devel, libFOO-dev or FOO-dev package too.

- FontForge 20060125 or newer.

- MetaFont (mf-nowin, mf, mfw or mfont binaries) and MetaPost (mpost binary), usually packaged with a LaTeX distribution like tetex or texlive.
- t1utils (version 1.33 or newer recommended).
- New Century Schoolbook fonts, as PFB files. These are shipped with X11 and Ghostscript, and are named '`c059033l.pfb`' '`c059036l.pfb`', '`c059013l.pfb`' and '`c059016l.pfb`'.
- GUILE (version 1.8.2 or newer). If you are installing binary packages, you may need to install guile-devel or guile-dev or libguile-dev too.
- Texinfo (version 4.11 or newer).
- The GNU c++ compiler (version 3.4 or newer. 4.x is strongly recommended).
- Python (version 2.4 or newer)
- GNU Make (version 3.78 or newer).
- gettext (version 0.17 or newer).
- Flex.
- Perl.
- GNU Bison.
- All packages required for running, including development packages with header files and libraries.

Running requirements

Running LilyPond requires proper installation of the following software

- Freetype (version 2.1.10 or newer).
- FontConfig (version 2.2).
- Pango (version 1.12 or newer).
- GUILE (version 1.8.2 or newer), or patch 1.8.1 with `http://lilypond.org/vc/gub.darcs/patches/guile`
- Python (version 2.4 or newer).
- Ghostscript (version 8.15 or newer. 8.50 recommended)
- Dejaview. (This is normally installed by default)

International fonts are required to create music with international text or lyrics.

Requirements for building documentation

You can view the documentation online at `http://lilypond.org/doc/`, but you can also build it locally. This process requires a successful compile of LilyPond, and some additional tools and packages:

- The netpbm utilities
- ImageMagick
- International fonts (see input/regression/utf-8.ly for hints about which font packages are necessary for your platform)
- Ghostscript, 8.50 with the patch from `http://bugs.ghostscript.com/show_bug.cgi?id=688154` and the patch from `http://bugs.ghostscript.com/show_bug.cgi?id=688017`, or use a release of Ghostscript which includes these patches, for example 8.60 or newer.
- Texi2HTML 1.79 or newer is strongly recommended to build documentation in HTML; support for building HTML documentation using `makeinfo` from GNU Texinfo is deprecated.

1.2.3 Building LilyPond

Compiling

To install GNU LilyPond, type

```
gunzip -c lilypond-x.y.z | tar xf -
cd lilypond-x.y.z
./configure # run with --help for applicable options
make
su -c 'make install'
```

If you are not root, you should choose a `--prefix` argument that points into your home directory, e.g.

```
./configure --prefix=$HOME/usr
```

Compiling for multiple platforms

If you want to build multiple versions of LilyPond with different configuration settings, you can use the `--enable-config=CONF` option of `configure`. You should use `make conf=CONF` to generate the output in '`out-CONF`'. For example, suppose you want to build with and without profiling, then use the following for the normal build

```
./configure --prefix=$HOME/usr/ --enable-checking
make
make install
```

and for the profiling version, specify a different configuration

```
./configure --prefix=$HOME/usr/ --enable-profiling --enable-config=prof --disable-checl
make conf=prof
make conf=prof install
```

Compiling outside the source tree

It is possible to compile LilyPond in a build tree different from the source tree, with `--srcdir` option of `configure`:

```
mkdir lily-build && cd lily-build
sourcedir/configure --srcdir=sourcedir
```

Useful `make` variables

If a less verbose build output if desired, the variable `QUIET_BUILD` may be set to `1` on `make` command line, or in '`local.make`' at top of the build tree.

1.2.4 Building documentation

This requires a successful compile of LilyPond, or using an external LilyPond binary.

Commands for building documentation

The documentation is built by issuing

```
make web
```

After compilation, the HTML documentation tree is available in '`out-www/offline-root/`', and can be browsed locally.

The HTML and PDF files can be installed into the standard documentation path by issuing

```
make web-install
```

This also installs Info documentation with images if the installation prefix is properly set; otherwise, instructions for manual installation of Info documentation are printed on standard output.

It is also possible to build a documentation tree in 'out-www/online-root/', with special processing, so it can be used on a website with content negotiation for automatic language selection; this can be achieved by issuing

```
make WEB_TARGETS=online web
```

and both 'offline' and 'online' targets can be generated by issuing

```
make WEB_TARGETS="offline online" web
```

Several targets are available to clean the documentation build and help with maintaining documentation; an overview of these targets is available with

```
make help
```

from every directory in the build tree. Most targets for documentation maintenance are available from 'Documentation/'; for more information, see 'Documentation/user/README.txt' and 'Documentation/TRANSLATION'.

The makefile variable QUIET_BUILD may be set to 1 for a less verbose build output, just like for building the programs.

Known issues and warnings

The most time consuming task for building the documentation is running LilyPond to build images of music, and there cannot be several simultaneously running lilypond-book instances, so -j make option does not significantly speed up the build process. To help speed it up, the makefile variable *CPU_COUNT* may be set in 'local.make' or on the command line to the number of .ly files that LilyPond should process simultaneously, e.g. on a bi-processor or dual core machine

```
make -j3 CPU_COUNT=3 web
```

The recommended value of *CPU_COUNT* is one plus the number of cores or processors, but it is advisable to set it to a smaller value if your system has not enough RAM to run that many simultaneous LilyPond instances.

If source files have changed since last documentation build, output files that need to be rebuilt are normally rebuilt, even if you do not run make web-clean first. However, building dependencies in the documentation are so complex that rebuilding of some targets may not be triggered as they should be; a workaround is to force rebuilding by touching appropriate files, e.g.

```
touch Documentation/user/*.itely
touch input/lsr/*.ly
```

Building documentation without compiling LilyPond

The documentation can be built locally without compiling LilyPond binary, if LilyPond is already installed on your system.

From a fresh Git checkout, do

```
./autogen.sh   # ignore any warning messages
cp GNUmakefile.in GNUmakefile
make -C python
nice make LILYPOND_EXTERNAL_BINARY=/path/to/bin/lilypond web
```

Please note that this may break sometimes – for example, if a new feature is added with a test file in input/regression, even the latest development release of LilyPond will fail to build the docs.

You may build the manual without building all the 'input/*' stuff: change directory, for example to 'Documentation/user', issue make web, which will build documentation in a subdirectory 'out-www' from the source files in current directory. In this case, if you also want to browse the documentation in its post-processed form, change back to top directory and issue

```
make out=www WWW-post
```

Known issues and warnings

You may also need to create a script for `pngtopnm` and `pnmtopng`. On GNU/Linux, I use this:

```
export LD_LIBRARY_PATH=/usr/lib
exec /usr/bin/pngtopnm "$@"
```

On MacOS X, I use this:

```
export DYLD_LIBRARY_PATH=/sw/lib
exec /sw/bin/pngtopnm "$@"
```

1.2.5 Testing LilyPond

LilyPond comes with an extensive suite that exercises the entire program. This suite can be used to automatically check the impact of a change. This is done as follows

```
make test-baseline
## apply your changes, compile
make check
```

This will leave an HTML page '`out/test-results/index.html`'. This page shows all the important differences that your change introduced, whether in the layout, MIDI, performance or error reporting.

To rerun tests, use

```
make test-redo            ## redo files differing from baseline
make test-clean           ## remove all test results
```

and then run `make check` again.

For tracking memory usage as part of this test, you will need GUILE CVS; especially the following patch: http://lilypond.org/vc/gub.darcs/patches/guile-1.9-gcstats.patch.

For checking the coverage of the test suite, do the following

```
./buildscripts/build-coverage.sh
# uncovered files, least covered first
python ./buildscripts/coverage.py  --summary out-cov/*.cc
# consecutive uncovered lines, longest first
python ./buildscripts/coverage.py  --uncovered out-cov/*.cc
```

1.2.6 Problems

For help and questions use lilypond-user@gnu.org. Send bug reports to bug-lilypond@gnu.org.

Bugs that are not fault of LilyPond are documented here.

Bison 1.875

There is a bug in bison-1.875: compilation fails with "parse error before 'goto'" in line 4922 due to a bug in bison. To fix, please recompile bison 1.875 with the following fix

```
$ cd lily; make out/parser.cc
$ vi +4919 out/parser.cc
# append a semicolon to the line containing "__attribute__ ((__unused__))
# save
$ make
```

Solaris

Solaris7, ./configure

'`./configure`' needs a POSIX compliant shell. On Solaris7, '`/bin/sh`' is not yet POSIX compliant, but '`/bin/ksh`' or bash is. Run configure like

```
CONFIG_SHELL=/bin/ksh ksh -c ./configure
```

or

```
CONFIG_SHELL=/bin/bash bash -c ./configure
```

FreeBSD

To use system fonts, dejaview must be installed. With the default port, the fonts are installed in '`usr/X11R6/lib/X11/fonts/dejavu`'.

Open the file '`$LILYPONDBASE/usr/etc/fonts/local.conf`' and add the following line just after the `<fontconfig>` line. (Adjust as necessary for your hierarchy.)

```
<dir>/usr/X11R6/lib/X11/fonts</dir>
```

International fonts

On MacOS X, all fonts are installed by default. However, finding all system fonts requires a bit of configuration; see this post on the `lilypond-user` mailing list.

On Linux, international fonts are installed by different means on every distribution. We cannot list the exact commands or packages that are necessary, as each distribution is different, and the exact package names within each distribution changes. Here are some hints, though:

```
Red Hat Fedora

    taipeifonts fonts-xorg-truetype ttfonts-ja fonts-arabic \
         ttfonts-zh_CN fonts-ja fonts-hebrew

Debian GNU/Linux

   apt-get install emacs-intl-fonts xfonts-intl-.* \
        ttf-kochi-gothic ttf-kochi-mincho \
        xfonts-bolkhov-75dpi xfonts-cronyx-100dpi xfonts-cronyx-75dpi
```

2 Setup

This chapter discusses various post-install configuration options for LilyPond and various other programs. This chapter may be safely treated as a reference: only read a section if it applies to you.

2.1 Setup for specific Operating Systems

This section explains how to perform additional setup for specific operating systems.

2.1.1 Setup for MacOS X

Using Python scripts on MacOS 10.3 or 10.4

LilyPond binaries for MacOS X do not provide Python, but Python 2.4 or newer is required by `convert-ly`. Therefore, if you use MacOS 10.3 or 10.4, you must install a newer Python version from http://python.org/download/, then edit the first line of `convert-ly` and `lilypond-book` as follows: if the Python binary you just installed is in your *PATH*, the first line should be

```
#!/usr/bin/env python
```

otherwise it should be

```
#!/path/to/newly_installed/python
```

MacOS X on the command line

The scripts — such as `lilypond-book`, `convert-ly`, `abc2ly`, and even `lilypond` itself — are included inside the `.app` file for MacOS X. They can be run from the command line by invoking them directly, e.g.

```
path/to/LilyPond.app/Contents/Resources/bin/lilypond
```

The same is true of the other scripts in that directory, including `lilypond-book`, `convert-ly`, `abc2ly`, etc.

Alternatively, you may create scripts which add the path automatically. Create a directory to store these scripts,

```
mkdir -p ~/bin
cd ~/bin
```

Create a file called `lilypond` which contains

```
exec path/to/LilyPond.app/Contents/Resources/bin/lilypond "$@"
```

Create similar files `lilypond-book`, `convert-ly`, and any other helper programs you use (`abc2ly`, `midi2ly`, etc). Simply replace the `bin/lilypond` with `bin/convert-ly` (or other program name) in the above file.

Make the file executable,

```
chmod u+x lilypond
```

Now, add this directory to your path. Modify (or create) a file called `.profile` in your home directory such that it contains

```
export PATH=$PATH:~/bin
```

This file should end with a blank line.

Note that *path/to* will generally be `/Applications/`.

2.2 Text editor support

There is support from different text editors for LilyPond.

2.2.1 Emacs mode

Emacs has a '`lilypond-mode`', which provides keyword autocompletion, indentation, LilyPond specific parenthesis matching and syntax coloring, handy compile short-cuts and reading LilyPond manuals using Info. If '`lilypond-mode`' is not installed on your platform, see below.

An Emacs mode for entering music and running LilyPond is contained in the source archive in the '`elisp`' directory. Do `make install` to install it to *elispdir*. The file '`lilypond-init.el`' should be placed to *load-path*'`/site-start.d/`' or appended to your '`~/.emacs`' or '`~/.emacs.el`'.

As a user, you may want add your source path (e.g. '`~/site-lisp/`') to your *load-path* by appending the following line (as modified) to your '`~/.emacs`'

```
(setq load-path (append (list (expand-file-name "~/site-lisp")) load-path))
```

2.2.2 Vim mode

For VIM, a '`vimrc`' is supplied, along with syntax coloring tools. A Vim mode for entering music and running LilyPond is contained in the source archive in `$VIM` directory.

The LilyPond file type is detected if the file '`~/.vim/filetype.vim`' has the following content

```
if exists("did_load_filetypes")
  finish
endif
augroup filetypedetect
  au! BufNewFile,BufRead *.ly           setf lilypond
augroup END
```

Please include this path by appending the following line to your '`~/.vimrc`'

```
set runtimepath+=/usr/local/share/lilypond/${LILYPOND_VERSION}/vim/
```

where ${LILYPOND_VERSION} is your LilyPond version. If LilyPond was not installed in '`/usr/local/`', then change this path accordingly.

2.2.3 jEdit

Created as a plugin for the jEdit text editor, LilyPondTool is the most feature-rich text-based tool for editing LilyPond scores. Its features include a Document Wizard with lyrics support to set up documents easier, and embedded PDF viewer with advanced point-and-click support. For screenshots, demos and installation instructions, visit `http://lilypondtool.organum.hu`

2.2.4 TexShop

The TexShop editor for MacOS X can be extended to run LilyPond, lilypond-book and convert-ly from within the editor, using the extensions available at `http://www.dimi.uniud.it/vitacolo/freesoftware.html`.

2.2.5 TextMate

There is a LilyPond bundle for TextMate. It may be installed by running

```
mkdir -p /Library/Application\ Support/TextMate/Bundles
cd /Library/Application\ Support/TextMate/Bundles
svn co http://macromates.com/svn/Bundles/trunk/Bundles/Lilypond.tmbundle/
```

2.2.6 LilyKDE

LilyKDE is a plugin for KDE's text editor Kate. It has a powerful Score Wizard to quickly setup a LilyPond document and an embedded PDF viewer.

LilyKDE can use Rumor, so music can entered by playing on a MIDI keyboard.

Other features are lyric hyphenation and running LilyPond on multiple files at once from within the KDE file manager.

2.3 Point and click

Point and click lets you find notes in the input by clicking on them in the PDF viewer. This makes it easier to find input that causes some error in the sheet music.

When this functionality is active, LilyPond adds hyperlinks to the PDF file. These hyperlinks are sent to the web-browser, which opens a text-editor with the cursor in the right place.

To make this chain work, you should configure your PDF viewer to follow hyperlinks using the '`lilypond-invoke-editor`' script supplied with LilyPond.

For Xpdf on UNIX, the following should be present in '`xpdfrc`'[1]

```
urlCommand      "lilypond-invoke-editor %s"
```

The program '`lilypond-invoke-editor`' is a small helper program. It will invoke an editor for the special `textedit` URIs, and run a web browser for others. It tests the environment variable `EDITOR` for the following patterns,

`emacs` this will invoke

```
emacsclient --no-wait +line:column file
```

`vim` this will invoke

```
gvim --remote +:line:normchar file
```

`nedit` this will invoke

```
nc -noask +line file'
```

The environment variable `LYEDITOR` is used to override this. It contains the command line to start the editor, where `%(file)s`, `%(column)s`, `%(line)s` is replaced with the file, column and line respectively. The setting

```
emacsclient --no-wait +%(line)s:%(column)s %(file)s
```

for `LYEDITOR` is equivalent to the standard emacsclient invocation.

The point and click links enlarge the output files significantly. For reducing the size of PDF and PS files, point and click may be switched off by issuing

```
\pointAndClickOff
```

in a '`.ly`' file. Point and click may be explicitly enabled with

```
\pointAndClickOn
```

Alternately, you may disable point and click with a command-line option:

```
lilypond -dno-point-and-click file.ly
```

> **Note:** You should always turn off point and click in any LilyPond files to be distributed to avoid including path information about your computer in the .pdf file, which can pose a security risk.

[1] On UNIX, this file is found either in '`/etc/xpdfrc`' or as '`.xpdfrc`' in your home directory.

3 Running LilyPond

This chapter details the technicalities of running LilyPond.

3.1 Normal usage

Most users run LilyPond through a GUI; see Section "First steps" in *Learning Manual* if you have not read this already.

3.2 Command-line usage

This section contains extra information about using LilyPond on the command-line. This may be desirable to pass extra options to the program. In addition, there are certain extra 'helper' programs (such as `midi2ly`) which are only available on the command-line.

By 'command-line', we mean the command line in the operating system. Windows users might be more familiar with the terms 'DOS shell' or 'command shell'; MacOS X users might be more familiar with the terms 'terminal' or 'console'. They should also consult Section 2.1.1 [Setup for MacOS X], page 7.

Describing how to use this part of an operating system is outside the scope of this manual; please consult other documentation on this topic if you are unfamiliar with the command-line.

3.2.1 Invoking `lilypond`

The `lilypond` executable may be called as follows from the command line.

```
lilypond [option]... file...
```

When invoked with a filename that has no extension, the '`.ly`' extension is tried first. To read input from stdin, use a dash (`-`) for *file*.

When '`filename.ly`' is processed it will produce '`filename.ps`' and '`filename.pdf`' as output. Several files can be specified; they will each be processed independently.[1]

If '`filename.ly`' contains more than one `\score` block, then the rest of the scores will be output in numbered files, starting with '`filename-1.pdf`'. In addition, the value of `output-suffix` will be inserted between the basename and the number. An input file containing

```
#(define output-suffix "violin")
\book { ... }
#(define output-suffix "cello")
\book { ... }
```

will output *base*'`-violin.pdf`' and *base*'`-cello-1.pdf`'.

3.2.2 Command line options for `lilypond`

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 expression will be evaluated in the `guile-user` module, so if you want to use definitions in *expr*, use

```
lilypond -e '(define-public a 42)'
```

on the command-line, and include

[1] The status of GUILE is not reset after processing a `.ly` file, so be careful not to change any system defaults from within Scheme.

```
#(use-modules (guile-user))
```

at the top of the `.ly` file.

`-f,--format=`*format*

which formats should be written. Choices for `format` are `svg`, `ps`, `pdf`, and `png`.

Example: `lilypond -fpng` *filename*`.ly`

`-d,--define-default=`*var*`=`*val*

This sets the internal program option *var* to the Scheme value *val*. If *val* is not supplied, then *#t* is used. To switch off an option, `no-` may be prefixed to *var*, e.g.

```
-dno-point-and-click
```

is the same as

```
-dpoint-and-click='#f'
```

Here are a few interesting options.

'`help`' Running `lilypond -dhelp` will print all of the `-d` options available.

'`paper-size`'

This option sets the default paper-size,

```
-dpaper-size=\"letter\"
```

Note that the string must be enclosed in escaped quotes (`\"`).

'`safe`' Do not trust the `.ly` input.

When LilyPond formatting is available through a web server, either the `--safe` or the `--jail` option **MUST** be passed. The `--safe` option will prevent inline Scheme code from wreaking havoc, for example

```
#(system "rm -rf /")
{
  c4^#(ly:export (ly:gulp-file "/etc/passwd"))
}
```

The `-dsafe` option works by evaluating in-line Scheme expressions in a special safe module. This safe module is derived from GUILE '`safe-r5rs`' module, but adds a number of functions of the LilyPond API. These functions are listed in '`scm/safe-lily.scm`'.

In addition, safe mode disallows `\include` directives and disables the use of backslashes in TEX strings.

In safe mode, it is not possible to import LilyPond variables into Scheme.

`-dsafe` does *not* detect resource overuse. It is still possible to make the program hang indefinitely, for example by feeding cyclic data structures into the backend. Therefore, if using LilyPond on a publicly accessible webserver, the process should be limited in both CPU and memory usage.

The safe mode will prevent many useful LilyPond snippets from being compiled. The `--jail` is a more secure alternative, but requires more work to set up.

'`backend`' the output format to use for the back-end. Choices for `format` are

`ps` for PostScript.

Postscript files include TTF, Type1 and OTF fonts. No subsetting of these fonts is done. When using oriental character sets, this can lead to huge files.

`eps` for encapsulated PostScript. This dumps every page (system) as a separate '`EPS`' file, without fonts, and as one collated '`EPS`' file with all pages (systems) including fonts.

This mode is used by default by `lilypond-book`.

`svg` for SVG (Scalable Vector Graphics). This dumps every page as a separate '`SVG`' file, with embedded fonts. You need a SVG viewer which supports embedded fonts, or a SVG viewer which is able to replace the embedded fonts with OTF fonts. Under UNIX, you may use Inkscape (version 0.42 or later), after copying the OTF fonts from the LilyPond directory (typically '`/usr/share/lilypond/VERSION/fonts/otf/`') to '`~/.fonts/`'.

`scm` for a dump of the raw, internal Scheme-based drawing commands.

`null` do not output a printed score; has the same effect as `-dno-print-pages`.

Example: `lilypond -dbackend=svg` *`filename`*`.ly`

'`preview`' Generate an output file containing the titles and the first system

'`print-pages`'
Generate the full pages, the default. `-dno-print-pages` is useful in combination with `-dpreview`.

`-h,--help`
Show a summary of usage.

`-H,--header=`*`FIELD`*
Dump a header field to file '`BASENAME.`*`FIELD`*'.

`--include, -I=`*`directory`*
Add *directory* to the search path for input files.

`-i,--init=`*`file`*
Set init file to *file* (default: '`init.ly`').

`-o,--output=`*`FILE`*
Set the default output file to *FILE*. The appropriate suffix will be added (e.g. `.pdf` for pdf)

`--ps` Generate PostScript.

`--png` Generate pictures of each page, in PNG format. This implies `--ps`. The resolution in DPI of the image may be set with

```
-dresolution=110
```

`--pdf` Generate PDF. This implies `--ps`.

`-j,--jail=`*`user`*`,`*`group`*`,`*`jail`*`,`*`dir`*
Run `lilypond` in a chroot jail.

The `--jail` option provides a more flexible alternative to `--safe` when LilyPond formatting is available through a web server or whenever LilyPond executes externally provided sources.

The `--jail` option works by changing the root of `lilypond` to *jail* just before starting the actual compilation process. The user and group are then changed to

match those provided, and the current directory is changed to *dir*. This setup guarantees that it is not possible (at least in theory) to escape from the jail. Note that for `--jail` to work `lilypond` must be run as root, which is usually accomplished in a safe way using `sudo`.

Setting up a jail is a slightly delicate matter, as we must be sure that LilyPond is able to find whatever it needs to compile the source *inside the jail*. A typical setup comprises the following items:

Setting up a separate filesystem
: A separate filesystem should be created for LilyPond, so that it can be mounted with safe options such as `noexec`, `nodev`, and `nosuid`. In this way, it is impossible to run executables or to write directly to a device from LilyPond. If you do not want to create a separate partition, just create a file of reasonable size and use it to mount a loop device. A separate filesystem also guarantees that LilyPond cannot write more space than it is allowed.

Setting up a separate user
: A separate user and group (say, `lily/lily`) with low privileges should be used to run LilyPond inside the jail. There should be a single directory writable by this user, which should be passed in *dir*.

Preparing the jail
: LilyPond needs to read a number of files while running. All these files are to be copied into the jail, under the same path they appear in the real root filesystem. The entire content of the LilyPond installation (e.g., '`/usr/share/lilypond`') should be copied.

  If problems arise, the simplest way to trace them down is to run LilyPond using `strace`, which will allow you to determine which files are missing.

Running LilyPond
: In a jail mounted with `noexec` it is impossible to execute any external program. Therefore LilyPond must be run with a backend that does not require any such program. As we already mentioned, it must be also run with superuser privileges (which, of course, it will lose immediately), possibly using `sudo`. It is a good idea to limit the number of seconds of CPU time LilyPond can use (e.g., using `ulimit -t`), and, if your operating system supports it, the amount of memory that can be allocated.

`-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**!)

3.2.3 Environment variables

`lilypond` recognizes the following environment variables:

`LILYPOND_DATADIR`
: 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.

`LILYPOND_GC_YIELD`
With this variable the memory footprint and performance can be adjusted. It is a percentage tunes memory management behavior. With higher values, the program uses more memory, with smaller values, it uses more CPU time. The default value is `70`.

3.3 Error messages

Different error messages can appear while compiling a file:

Warning Something looks suspect. If you are requesting something out of the ordinary then you will understand the message, and can ignore it. However, warnings usually indicate that something is wrong with the input file.

Error Something is definitely wrong. The current processing step (parsing, interpreting, or formatting) will be finished, but the next step will be skipped.

Fatal error
Something is definitely wrong, and LilyPond cannot continue. This happens rarely. The most usual cause is misinstalled fonts.

Scheme error
Errors that occur while executing Scheme code are caught by the Scheme interpreter. If running with the verbose option (`-V` or `--verbose`) then a call trace of the offending function call is printed.

Programming error
There was some internal inconsistency. These error messages are intended to help the programmers and debuggers. Usually, they can be ignored. Sometimes, they come in such big quantities that they obscure other output.

Aborted (core dumped)
This signals a serious programming error that caused the program to crash. Such errors are considered critical. If you stumble on one, send a bug-report.

If warnings and errors can be linked to some part of the input file, then error messages have the following form

```
filename:lineno:columnno: message
offending input line
```

A line-break is inserted in the offending line to indicate the column where the error was found. For example,

```
test.ly:2:19: error: not a duration: 5
  { c'4 e'
            5 g' }
```

These locations are LilyPond's best guess about where the warning or error occurred, but (by their very nature) warnings and errors occur when something unexpected happens. If you can't see an error in the indicated line of your input file, try checking one or two lines above the indicated position.

3.4 Updating files with `convert-ly`

The LilyPond input syntax is routinely changed to simplify it or improve it in different ways. As a side effect of this, the LilyPond interpreter often is no longer compatible with older input files. To remedy this, the program `convert-ly` can be used to deal with most of the syntax changes between LilyPond versions.

It uses `\version` statements in the input files to detect the old version number. In most cases, to upgrade your input file it is sufficient to run

```
convert-ly -e myfile.ly
```

MacOS X users may execute this command under the menu entry `Compile > Update syntax`.

If there are no changes to myfile.ly and file called myfile.ly.NEW is created, then myfile.ly is already updated.

3.4.1 Command line options for `convert-ly`

`convert-ly` always converts up to the last syntax change handled by it. This means that the `\version` number left in the file is usually lower than the version of `convert-ly` itself.

To upgrade LilyPond fragments in texinfo files, use

```
convert-ly --from=... --to=... --no-version *.itely
```

To see the changes in the LilyPond syntax between two versions, use

```
convert-ly --from=... --to=... -s
```

To upgrade many files at once, combine `convert-ly` with standard UNIX commands. This example will upgrade all `.ly` files in the current directory

```
for f in *.ly; do convert-ly -e $f; done;
```

In general, the program is invoked as follows:

`convert-ly [`*option*`]... `*file*`...`

The following options can be given:

`-e,--edit`
: Do an inline edit of the input file. Overrides `--output`.

`-f,--from=`*from-patchlevel*
: Set the version to convert from. If this is not set, `convert-ly` will guess this, on the basis of `\version` strings in the file.

`-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.

3.4.2 Problems with `convert-ly`

Not all language changes are handled. Only one output option can be specified. Automatically updating scheme and LilyPond scheme interfaces is quite unlikely; be prepared to tweak scheme code manually.

```
There are a few things that the convert-ly cannot handle.  Here's a list
of limitations that the community has complained about.

This bug report structure has been chosen because convert-ly has a
structure that doesn't allow to smoothly implement all needed changes.
Thus this is just a wishlist, placed here for reference.
```

```
1.6->2.0:
 Doesn't always convert figured bass correctly, specifically things like {<
>}.  Mats' comment on working around this:
   To be able to run convert-ly
   on it, I first replaced all occurrences of '{<' to some dummy like '{#'
   and similarly I replaced '>}' with '&}'.  After the conversion, I could
   then change back from '{ #' to '{ <' and from '& }' to '> }'.
 Doesn't convert all text markup correctly.  In the old markup syntax,
 it was possible to group a number of markup commands together within
parentheses, e.g.
   -#'((bold italic) "string")
   This will incorrectly be converted into
   -\markup{{\bold italic} "string"}
   instead of the correct
   -\markup{\bold \italic "string"}
2.0->2.2:
 Doesn't handle \partcombine
 Doesn't do \addlyrics => \lyricsto, this breaks some scores with multiple
stanzas.
2.0->2.4:
 \magnify isn't changed to \fontsize.
    - \magnify #m => \fontsize #f, where f = 6ln(m)/ln(2)
 remove-tag isn't changed.
    - \applyMusic #(remove-tag '. . .) => \keepWithTag #'. . .
 first-page-number isn't changed.
    - first-page-number no => print-first-page-number = ##f
 Line breaks in header strings aren't converted.
    - \\\\  as line break in \header strings => \markup \center-align <
      "First Line" "Second Line" >
 Crescendo and decrescendo terminators aren't converted.
    - \rced => \!
    - \rc => \!
2.2->2.4:
 \turnOff (used in \set Staff.VoltaBracket = \turnOff) is not properly
converted.
2.4.2->2.5.9
 \markup{ \center-align <{ ... }> } should be converted to:
 \markup{ \center-align {\line { ... }} }
 but now, \line is missing.
2.4->2.6
 Special LaTeX characters such as $~$ in text are not converted to UTF8.
2.8
 \score{} must now begin with a music expression.  Anything else
 (particularly \header{}) must come after the music.
```

3.5 Reporting bugs

If you have input that results in a crash or an erroneous output, then that is a bug. There is a list of current bugs on our Google bug tracker,

http://code.google.com/p/lilypond/issues/list

If you have discovered a bug which is not listed, please report the bug by following the directions on

http://lilypond.org/web/devel/participating/bugs

Please construct and submit minimal examples of bugs in reports. We do not have the resources to investigate reports which are not as small as possible.

4 lilypond-book: Integrating text and music

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. The pictures are created separately, yielding PostScript output or PNG images, and those are included into a LaTeX or HTML document.

`lilypond-book` provides a way to automate this process: This program extracts snippets of music from your document, runs `lilypond` on them, and outputs the document with pictures substituted for the music. The line width and font size definitions for the music are adjusted to match the layout of your document.

This is a separate program from `lilypond` itself, and is run on the command line; for more information, see Section 3.2 [Command-line usage], page 10. If you have MacOS 10.3 or 10.4 and you have trouble running `lilypond-book`, see Section 2.1.1 [Setup for MacOS X], page 7.

This procedure may be applied to LaTeX, HTML, Texinfo or DocBook documents.

4.1 An example of a musicological document

Some texts contain music examples. These texts are musicological treatises, songbooks, or manuals like this. Such texts can be made by hand, simply by importing a PostScript figure into the word processor. However, there is an automated procedure to reduce the amount of work involved in HTML, LaTeX, Texinfo and DocBook documents.

A script called `lilypond-book` will extract the music fragments, format them, and put back the resulting notation. Here we show a small example for use with LaTeX. The example also contains explanatory text, so we will not comment on it further.

Input

```
\documentclass[a4paper]{article}

\begin{document}

Documents for \verb+lilypond-book+ may freely mix music and text.
For example,

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

Options are put in brackets.

\begin[fragment,quote,staffsize=26,verbatim]{lilypond}
  c'4 f16
\end{lilypond}

Larger examples can be put into a separate file, and introduced with
\verb+\lilypondfile+.

\lilypondfile[quote,noindent]{screech-boink.ly}

(If needed, replace screech-boink.ly by any .ly file you put in the same
directory as this file.)
```

```
\end{document}
```

Processing

Save the code above to a file called 'lilybook.lytex', then in a terminal run

```
lilypond-book --output=out --pdf lilybook.lytex
lilypond-book (GNU LilyPond) 2.12.1

Reading lilybook.lytex...
..lots of stuff deleted..
Compiling lilybook.tex...
cd out
pdflatex lilybook
..lots of stuff deleted..
xpdf lilybook
(replace xpdf by your favorite PDF viewer)
```
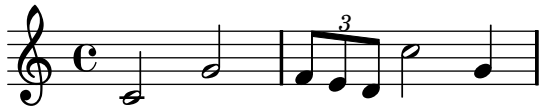
Running `lilypond-book` and `latex` creates a lot of temporary files, which would clutter up the working directory. To remedy this, use the `--output=dir` option. It will create the files in a separate subdirectory 'dir'.

Finally the result of the LaTeX example shown above.[1] This finishes the tutorial section.

[1] This tutorial is processed with Texinfo, so the example gives slightly different results in layout.
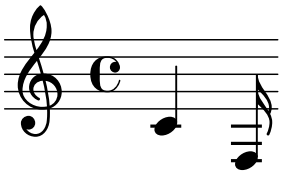
Output

Documents for `lilypond-book` may freely mix music and text. For example,



Options are put in brackets.

```
c'4 f16
```

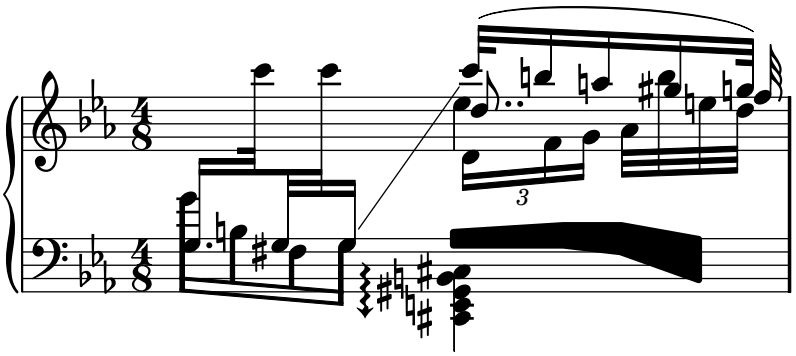


Larger examples can be put into a separate file, and introduced with `\lilypondfile`.

Screech and boink

Random complex notation

Han-Wen Nienhuys



4.2 Integrating music and text

Here we explain how to integrate LilyPond with various output formats.

4.2.1 LaTeX

LaTeX is the de-facto standard for publishing layouts in the exact sciences. It is built on top of the TeX typesetting engine, providing the best typography available anywhere.

See *The Not So Short Introduction to LaTeX* for an overview on how to use LaTeX.

Music is entered using

```
\begin[options,go,here]{lilypond}
  YOUR LILYPOND CODE
\end{lilypond}
```

or

```
\lilypondfile[options,go,here]{filename}
```

or
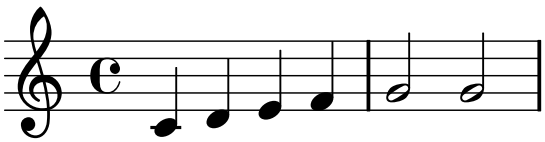
```
\lilypond{ YOUR LILYPOND CODE }
```

Additionally, `\lilypondversion` displays the current version of lilypond. Running `lilypond-book` yields a file that can be further processed with LaTeX.

We show some examples here. The `lilypond` environment

```
\begin[quote,fragment,staffsize=26]{lilypond}
  c' d' e' f' g'2 g'2
\end{lilypond}
```

produces



The short version

```
\lilypond[quote,fragment,staffsize=11]{<c' e' g'>}
```

produces



Currently, you cannot include `{` or `}` within `\lilypond{}`, so this command is only useful with the `fragment` option.

The default line width of the music will be adjusted by examining the commands in the document preamble, the part of the document before `\begin{document}`. The `lilypond-book` command sends these to LaTeX to find out how wide the text is. The line width for the music fragments is then adjusted to the text width. Note that this heuristic algorithm can fail easily; in such cases it is necessary to use the `line-width` music fragment option.

Each snippet will call the following macros if they have been defined by the user:

- `\preLilyPondExample` called before the music,
- `\postLilyPondExample` called after the music,
- `\betweenLilyPondSystem[1]` is called between systems if `lilypond-book` has split the snippet into several PostScript files. It must be defined as taking one parameter and will be passed the number of files already included in this snippet. The default is to simply insert a `\linebreak`.

Selected Snippets

Sometimes it is useful to display music elements (such as ties and slurs) as if they continued after the end of the fragment. This can be done by breaking the staff and suppressing inclusion of the rest of the LilyPond output.

In LaTeX, define `\betweenLilyPondSystem` in such a way that inclusion of other systems is terminated once the required number of systems are included. Since `\betweenLilypondSystem` is first called *after* the first system, including only the first system is trivial.

```
\def\betweenLilyPondSystem#1{\endinput}

\begin[fragment]{lilypond}
  c'1\( e'( c'~ \break c' d) e f\)
\end{lilypond}
```

If a greater number of systems is requested, a TeX conditional must be used before the `\endinput`. In this example, replace '2' by the number of systems you want in the output,

```
\def\betweenLilyPondSystem#1{
    \ifnum##1<2\else\endinput\fi
}
```

Remember that the definition of `\betweenLilyPondSystem` is effective until TeX quits the current group (such as the LaTeX environment) or is overridden by another definition (which is, in most cases, for the rest of the document). To reset your definition, write

```
\let\betweenLilyPondSystem\undefined
```

in your LaTeX source.

This may be simplified by defining a TeX macro

```
\def\onlyFirstNSystems#1{
    \def\betweenLilyPondSystem##1{\ifnum##1<#1\else\endinput\fi}
}
```

and then saying only how many systems you want before each fragment,

```
\onlyFirstNSystems{3}
\begin{lilypond}...\end{lilypond}
\onlyFirstNSystems{1}
\begin{lilypond}...\end{lilypond}
```

See also

There are specific `lilypond-book` command line options and other details to know when processing LaTeX documents, see Section 4.4 [Invoking lilypond-book], page 27.

4.2.2 Texinfo

Texinfo is the standard format for documentation of the GNU project. An example of a Texinfo document is this manual. The HTML, PDF, and Info versions of the manual are made from the Texinfo document.

In the input file, music is specified with

```
@lilypond[options,go,here]
  YOUR LILYPOND CODE
@end lilypond
```

or

```
@lilypond[options,go,here]{ YOUR LILYPOND CODE }
```

or

```
@lilypondfile[options,go,here]{filename}
```
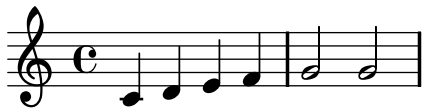
Additionally, `@lilypondversion` displays the current version of lilypond.

When `lilypond-book` is run on it, this results in a Texinfo file (with extension '`.texi`') containing `@image` tags for HTML, Info and printed output. `lilypond-book` generates images of the music in EPS and PDF formats for use in the printed output, and in PNG format for use in HTML and Info output.

We show two simple examples here. A `lilypond` environment

```
@lilypond[fragment]
c' d' e' f' g'2 g'
@end lilypond
```

produces



The short version

```
@lilypond[fragment,staffsize=11]{<c' e' g'>}
```

produces
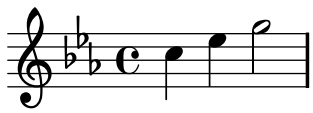


Contrary to LaTeX, `@lilypond{...}` does not generate an in-line image. It always gets a paragraph of its own.

4.2.3 HTML

Music is entered using

```
<lilypond fragment relative=2>
\key c \minor c4 es g2
</lilypond>
```

`lilypond-book` then produces an HTML file with appropriate image tags for the music fragments:



For inline pictures, use `<lilypond ... />`, where the options are separated by a colon from the music, for example

```
Some music in <lilypond relative=2: a b c/> a line of text.
```

To include separate files, say

```
<lilypondfile option1 option2 ...>filename</lilypondfile>
```

Additionally, `<lilypondversion/>` displays the current version of lilypond.

4.2.4 DocBook

For inserting LilyPond snippets it is good to keep the conformity of our DocBook document, thus allowing us to use DocBook editors, validation etc. So we don't use custom tags, only specify a convention based on the standard DocBook elements.

Common conventions

For inserting all type of snippets we use the `mediaobject` and `inlinemediaobject` element, so our snippets can be formatted inline or not inline. The snippet formatting options are always provided in the `role` property of the innermost element (see in next sections). Tags are chosen to allow DocBook editors format the content gracefully. The DocBook files to be processed with `lilypond-book` should have the extension '`.lyxml`'.

Including a LilyPond file

This is the most simple case. We must use the '`.ly`' extension for the included file, and insert it as a standard `imageobject`, with the following structure:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="music1.ly" role="printfilename" />
  </imageobject>
</mediaobject>
```

Note that you can use `mediaobject` or `inlinemediaobject` as the outermost element as you wish.

Including LilyPond code

Including LilyPond code is possible by using a `programlisting`, where the language is set to `lilypond` with the following structure:

```
<inlinemediaobject>
  <textobject>
    <programlisting language="lilypond" role="fragment verbatim staffsize=16 ragged-rig
\context Staff \with {
  \remove Time_signature_engraver
  \remove Clef_engraver}
  { c4( fis) }
    </programlisting>
  </textobject>
</inlinemediaobject>
```

As you can see, the outermost element is a `mediaobject` or `inlinemediaobject`, and there is a `textobject` containing the `programlisting` inside.

Processing the DocBook document

Running `lilypond-book` on our '`.lyxml`' file will create a valid DocBook document to be further processed with '`.xml`' extension. If you use dblatex, it will create a PDF file from this document automatically. For HTML (HTML Help, JavaHelp etc.) generation you can use the official DocBook XSL stylesheets, however, it is possible that you have to make some customization for it.

4.3 Music fragment options

In the following, a 'LilyPond command' refers to any command described in the previous sections which is handled by `lilypond-book` to produce a music snippet. For simplicity, LilyPond commands are only shown in LaTeX syntax.

Note that the option string is parsed from left to right; if an option occurs multiple times, the last one is taken.

The following options are available for LilyPond commands:

`staffsize=`*ht*
: Set staff size to *ht*, which is measured in points.

`ragged-right`
: Produce ragged-right lines with natural spacing, i.e., `ragged-right = ##t` is added to the LilyPond snippet. This is the default for the `\lilypond{}` command if no `line-width` option is present. It is also the default for the `lilypond` environment if the `fragment` option is set, and no line width is explicitly specified.

`noragged-right`
: For single-line snippets, allow the staff length to be stretched to equal that of the line width, i.e., `ragged-right = ##f` is added to the LilyPond snippet.

`line-width`
`line-width=`*size*`\`*unit*
: Set line width to *size*, using *unit* as units. *unit* is one of the following strings: `cm`, `mm`, `in`, or `pt`. This option affects LilyPond output (this is, the staff length of the music snippet), not the text layout.

  If used without an argument, set line width to a default value (as computed with a heuristic algorithm).

  If no `line-width` option is given, `lilypond-book` tries to guess a default for `lilypond` environments which don't use the `ragged-right` option.

`notime`
: Do not print the time signature, and turns off the timing (time signature, bar lines) in the score.

`fragment`
: Make `lilypond-book` add some boilerplate code so that you can simply enter, say,

  ```
  c'4
  ```

  without `\layout`, `\score`, etc.

`nofragment`
: Do not add additional code to complete LilyPond code in music snippets. Since this is the default, `nofragment` is redundant normally.

`indent=`*size*`\`*unit*
: Set indentation of the first music system to *size*, using *unit* as units. *unit* is one of the following strings: `cm`, `mm`, `in`, or `pt`. This option affects LilyPond, not the text layout.

`noindent`
: Set indentation of the first music system to zero. This option affects LilyPond, not the text layout. Since no indentation is the default, `noindent` is redundant normally.

`quote`
: Reduce line length of a music snippet by $2*0.4$ in and put the output into a quotation block. The value '0.4 in' can be controlled with the `exampleindent` option.

`exampleindent`
: Set the amount by which the `quote` option indents a music snippet.

`relative`
`relative=`*n*
: Use relative octave mode. By default, notes are specified relative to middle C. The optional integer argument specifies the octave of the starting note, where the default `1` is middle C. `relative` option only works when `fragment` option is set, so `fragment` is automatically implied by `relative`, regardless of the presence of any `(no)fragment` option in the source.

LilyPond also uses `lilypond-book` to produce its own documentation. To do that, some more obscure music fragment options are available.

`verbatim` The argument of a LilyPond command is copied to the output file and enclosed in a verbatim block, followed by any text given with the `intertext` option (not implemented yet); then the actual music is displayed. This option does not work well with `\lilypond{}` if it is part of a paragraph.

If `verbatim` is used in a `lilypondfile` command, it is possible to enclose verbatim only a part of the source file. If the source file contain a comment containing '`begin verbatim`' (without quotes), quoting the source in the verbatim block will start after the last occurrence of such a comment; similarly, quoting the source verbatim will stop just before the first occurrence of a comment containing '`end verbatim`', if there is any. In the following source file example, the music will be interpreted in relative mode, but the verbatim quote will not show the `relative` block, i.e.

```
\relative c' { % begin verbatim
  c4 e2 g4
  f2 e % end verbatim
}
```

will be printed with a verbatim block like

```
  c4 e2 g4
  f2 e
```

If you would like to translate comments and variable names in verbatim output but not in the sources, you may set the environment variable `LYDOC_LOCALEDIR` to a directory path; the directory should contain a tree of '`.mo`' message catalogs with `lilypond-doc` as a domain.

`addversion`
(Only for Texinfo output.) Prepend line `\version @w{"@version{}"}` to `verbatim` output.

`texidoc` (Only for Texinfo output.) If `lilypond` is called with the '`--header=texidoc`' option, and the file to be processed is called '`foo.ly`', it creates a file '`foo.texidoc`' if there is a `texidoc` field in the `\header`. The `texidoc` option makes `lilypond-book` include such files, adding its contents as a documentation block right before the music snippet.

Assuming the file '`foo.ly`' contains

```
\header {
  texidoc = "This file demonstrates a single note."
}
{ c'4 }
```

and we have this in our Texinfo document '`test.texinfo`'

```
@lilypondfile[texidoc]{foo.ly}
```

the following command line gives the expected result

```
lilypond-book --pdf --process="lilypond \
  -dbackend=eps --header=texidoc" test.texinfo
```

Most LilyPond test documents (in the '`input`' directory of the distribution) are small '`.ly`' files which look exactly like this.

For localization purpose, if the Texinfo document contains `@documentlanguage` *`LANG`* and '`foo.ly`' header contains a `texidoc`*`LANG`* field, and if `lilypond` is called with '`--header=texidoc`*`LANG`*', then '`foo.texidoc`*`LANG`*' will be included instead of '`foo.texidoc`'.

`lilyquote`
(Only for Texinfo output.) This option is similar to quote, but only the music snippet (and the optional verbatim block implied by `verbatim` option) is put into

a quotation block. This option is useful if you want to `quote` the music snippet but not the `texidoc` documentation block.

`doctitle` (Only for Texinfo output.) This option works similarly to `texidoc` option: if `lilypond` is called with the '`--header=doctitle`' option, and the file to be processed is called '`foo.ly`' and contains a `doctitle` field in the `\header`, it creates a file '`foo.doctitle`'. When `doctitle` option is used, the contents of '`foo.doctitle`', which should be a single line of *text*, is inserted in the Texinfo document as `@lydoctitle` *text*. `@lydoctitle` should be a macro defined in the Texinfo document. The same remark about `texidoc` processing with localized languages also applies to `doctitle`.

`nogettext`
(Only for Texinfo output.) Do not translate comments and variable names in the snippet quoted verbatim.

`printfilename`
If a LilyPond input file is included with `\lilypondfile`, print the file name right before the music snippet. For HTML output, this is a link. Only the base name of the file is printed, i.e. the directory part of the file path is stripped.

`fontload` This option includes fonts in all of the generated EPS-files for this snippet. This should be used if the snippet uses any font that LaTeX cannot find on its own.

4.4 Invoking `lilypond-book`

`lilypond-book` produces a file with one of the following extensions: '`.tex`', '`.texi`', '`.html`' or '`.xml`', depending on the output format. All of '`.tex`', '`.texi`' and '`.xml`' files need further processing.

Format-specific instructions

LaTeX

There are two ways of processing your LaTeX document for printing or publishing: getting a PDF file directly with PDFLaTeX, or getting a PostScript file with LaTeX via a DVI to PostScript translator like `dvips`. The first way is simpler and recommended[1], and whichever way you use, you can easily convert between PostScript and PDF with tools, like `ps2pdf` and `pdf2ps` included in Ghostscript package.

To produce a PDF file through PDFLaTeX, use

```
lilypond-book --pdf yourfile.pdftex
pdflatex yourfile.tex
```

To produce PDF output via LaTeX/`dvips`/`ps2pdf`, you should do

```
lilypond-book yourfile.lytex
latex yourfile.tex
dvips -Ppdf yourfile.dvi
ps2pdf yourfile.ps
```

The '`.dvi`' file created by this process will not contain note heads. This is normal; if you follow the instructions, they will be included in the '`.ps`' and '`.pdf`' files.

Running `dvips` may produce some warnings about fonts; these are harmless and may be ignored. If you are running `latex` in twocolumn mode, remember to add `-t landscape` to the `dvips` options.

[1] Note that PDFLaTeX and LaTeX may not be both usable to compile any LaTeX document, that is why we explain the two ways.

Texinfo

To produce a Texinfo document (in any output format), follow the normal procedures for Texinfo; this is, either call `texi2pdf` or `texi2dvi` or `makeinfo`, depending on the output format you want to create. See the documentation of Texinfo for further details.

Command line options

`lilypond-book` accepts the following command line options:

`-f` *format*
`--format=`*format*
: Specify the document type to process: `html`, `latex`, `texi` (the default) or `docbook`. If this option is missing, `lilypond-book` tries to detect the format automatically, see Section 4.5 [Filename extensions], page 29. Currently, `texi` is the same as `texi-html`.

`-F` *filter*
`--filter=`*filter*
: Pipe snippets through *filter*. `lilypond-book` will not –filter and –process at the same time. For example,

```
lilypond-book --filter='convert-ly --from=2.0.0 -' my-book.tely
```

`-h`
`--help`
: Print a short help message.

`-I` *dir*
`--include=`*dir*
: Add *dir* to the include path. `lilypond-book` also looks for already compiled snippets in the include path, and does not write them back to the output directory, so in some cases it is necessary to invoke further processing commands such as `makeinfo` or `latex` with the same `-I` *dir* options.

`-o` *dir*
`--output=`*dir*
: Place generated files in directory *dir*. Running `lilypond-book` generates lots of small files that LilyPond will process. To avoid all that garbage in the source directory, use the '`--output`' command line option, and change to that directory before running `latex` or `makeinfo`.

```
lilypond-book --output=out yourfile.lytex
cd out
```

...

`--skip-lily-check`
: Do not fail if no lilypond output is found. It is used for LilyPond Info documentation without images.

`--skip-png-check`
: Do not fail if no PNG images are found for EPS files. It is used for LilyPond Info documentation without images.

`--lily-output-dir=`*dir*
: Write lily-XXX files to directory *dir*, link into `--output` directory. Use this option to save building time for documents in different directories which share a lot of identical snippets.

`--info-images-dir=`*dir*
: Format Texinfo output so that Info will look for images of music in *dir*.

`--latex-program=`*prog*
: Run executable `prog` instead of `latex`. This is useful if your document is processed with `xelatex`, for example.

`--left-padding=`*amount*
: Pad EPS boxes by this much. *amount* is measured in millimeters, and is 3.0 by default. This option should be used if the lines of music stick out of the right margin.

  The width of a tightly clipped system can vary, due to notation elements that stick into the left margin, such as bar numbers and instrument names. This option will shorten each line and move each line to the right by the same amount.

`-P` *command*
`--process=`*command*
: Process LilyPond snippets using *command*. The default command is `lilypond`. `lilypond-book` will not `--filter` and `--process` at the same time.

`--pdf`
: Create PDF files for use with PDFLaTeX.

`-V`
`--verbose`
: Be verbose.

`-v`
`--version`
: Print version information.

Known issues and warnings

The Texinfo command `@pagesizes` is not interpreted. Similarly, LaTeX commands that change margins and line widths after the preamble are ignored.

Only the first `\score` of a LilyPond block is processed.

4.5 Filename extensions

You can use any filename extension for the input file, but if you do not use the recommended extension for a particular format you may need to manually specify the output format; for details, see Section 4.4 [Invoking lilypond-book], page 27. Otherwise, `lilypond-book` automatically selects the output format based on the input filename's extension.

extension	output format
'`.html`'	HTML

'.itely'	Texinfo
'.latex'	LaTeX
'.lytex'	LaTeX
'.lyxml'	DocBook
'.tely'	Texinfo
'.tex'	LaTeX
'.texi'	Texinfo
'.texinfo'	Texinfo
'.xml'	HTML

If you use the same filename extension for the input file than the extension lilypond-book uses for the output file, and if the input file is in the same directory as lilypond-book working directory, you must use --output option to make lilypond-book running, otherwise it will exit with an error message like "Output would overwrite input file".

4.6 Alternative methods of mixing text and music

This section shows methods to integrate text and music, different than the automated method with lilypond-book.

Many quotes from a large score

If you need to quote many fragments from a large score, you can also use the clip systems feature, see Section "Extracting fragments of music" in *Notation Reference*.

Inserting LilyPond output into OpenOffice.org

LilyPond notation can be added to OpenOffice.org with OOoLilyPond.

Inserting LilyPond output into other programs

To insert LilyPond output in other programs, use lilypond instead of lilypond-book. Each example must be created individually and added to the document; consult the documentation for that program. Most programs will be able to insert LilyPond output in 'PNG', 'EPS', or 'PDF' formats.

To reduce the white space around your LilyPond score, use the following options

```
\paper{
  indent=0\mm
  line-width=120\mm
  oddFooterMarkup=##f
  oddHeaderMarkup=##f
  bookTitleMarkup = ##f
  scoreTitleMarkup = ##f
}

{ c1 }
```

To produce a useful 'EPS' file, use

```
lilypond -dbackend=eps -dno-gs-load-fonts -dinclude-eps-fonts   myfile.ly
```

'PNG':

```
lilypond -dbackend=eps -dno-gs-load-fonts -dinclude-eps-fonts --png myfile.ly
```

5 Converting from other formats

Music can be entered also by importing it from other formats. This chapter documents the tools included in the distribution to do so. There are other tools that produce LilyPond input, for example GUI sequencers and XML converters. Refer to the website for more details.

These are separate programs from `lilypond` itself, and are run on the command line; see Section 3.2 [Command-line usage], page 10 for more information. If you have MacOS 10.3 or 10.4 and you have trouble running some of these scripts, e.g. `convert-ly`, see Section 2.1.1 [Setup for MacOS X], page 7.

Known issues and warnings

We unfortunately do not have the resources to maintain these programs; please consider them "as-is". Patches are appreciated, but bug reports will almost certainly not be resolved.

5.1 Invoking `midi2ly`

`midi2ly` translates a Type 1 MIDI 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 importing files from a program that has a converter for a direct format.

`midi2ly` converts tracks into Section "Staff" in *Internals Reference* and channels into Section "Voice" in *Internals Reference* contexts. Relative mode is used for pitches, durations are only written when necessary.

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. When invoked with quantizing (`-s` and `-d` options) `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.

It is invoked from the command-line as follows,

```
midi2ly [option]... midi-file
```

Note that by 'command-line', we mean the command line of the operating system. See Chapter 5 [Converting from other formats], page 31, for more information about this.

The following options are supported by `midi2ly`.

`-a, --absolute-pitches`
: Print absolute pitches.

`-d, --duration-quant=`*DUR*
: Quantize note durations on *DUR*.

`-e, --explicit-durations`
: Print explicit durations.

`-h,--help`
: Show summary of usage.

`-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`.

`-o, --output=`*file*
: Write output to *file*.

`-s, --start-quant=`*DUR*
: Quantize note starts on *DUR*.

`-t, --allow-tuplet=`*DUR*`*`*NUM*`/`*DEN*
: Allow tuplet durations *DUR***NUM*/*DEN*.

`-v, --verbose`
: Be verbose.

`-V, --version`
: Print version number.

`-w, --warranty`
: Show warranty and copyright.

`-x, --text-lyrics`
: Treat every text as a lyric.

Known issues and warnings

Overlapping notes in an arpeggio will not be correctly rendered. The first note will be read and the others will be ignored. Set them all to a single duration and add phrase markings or pedal indicators.

5.2 Invoking `musicxml2ly`

MusicXML is an XML dialect for representing music notation.

`musicxml2ly` extracts the notes, articulations, score structure, lyrics, etc. from part-wise MusicXML files, and writes them to a .ly file. It is invoked from the command-line.

It is invoked from the command-line as follows,

```
musicxml2ly [option]... xml-file
```

Note that by 'command-line', we mean the command line of the operating system. See Chapter 5 [Converting from other formats], page 31, for more information about this.

If the given filename is '`-`', `musicxml2ly` reads input from the command line.

The following options are supported by `musicxml2ly`:

`-a, --absolute`
: convert pitches in absolute mode.

`-h,--help`
: print usage and option summary.

`-l, --language=LANG`
: use a different language file 'LANG.ly' and corresponding pitch names, e.g. 'deutsch' for deutsch.ly and German note names.

`--lxml` use the lxml.etree Python package for XML-parsing; uses less memory and cpu time.

`--nd --no-articulation-directions`
: do not convert directions (`^`, `_` or `-`) for articulations, dynamics, etc.

`--no-beaming`
: do not convert beaming information, use LilyPond's automatic beaming instead.

`-o,--output=`*file*
: set output filename to *file*. If *file* is '`-`', the output will be printed on stdout. If not given, *xml-file*'`.ly`' will be used.

`-r,--relative`
: convert pitches in relative mode (default).

`-v,--verbose`
: be verbose.

`--version`
: print version information.

`-z,--compressed`
: input file is a zip-compressed MusicXML file.

5.3 Invoking `abc2ly`

ABC is a fairly simple ASCII based format. It is described at the ABC site:

http://www.walshaw.plus.com/abc/learn.html.

`abc2ly` translates from ABC to LilyPond. It is invoked as follows:

`abc2ly [`*`option`*`]... `*`abc-file`*

The following options are supported by `abc2ly`:

`-b,--beams=None`
: preserve ABC's notion of beams

`-h,--help`
: this help

`-o,--output=`*`file`*
: set output filename to *file*.

`-s,--strict`
: be strict about success

`--version`
: print version information.

There is a rudimentary facility for adding LilyPond code to the ABC source file. If you say:

```
%%LY voices \set 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.

Known issues and warnings

The ABC standard is not very 'standard'. For extended features (e.g., 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.

5.4 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 from the command-line as follows.

```
etf2ly [option]... etf-file
```

Note that by 'command-line', we mean the command line of the operating system. See Chapter 5 [Converting from other formats], page 31, for more information about this.

The following options are supported by `etf2ly`:

`-h,--help`
: this help

`-o,--output=`*FILE*
: set output filename to *FILE*

`--version`
: version information

Known issues and warnings

The list of articulation scripts is incomplete. Empty measures confuse `etf2ly`. Sequences of grace notes are ended improperly.

5.5 Generating LilyPond files

LilyPond itself does not come with support for any other formats, but there are some external tools that also generate LilyPond files.

These tools include

- Denemo, a graphical score editor.
- Rumor, a realtime monophonic MIDI to LilyPond converter.
- lyqi, an Emacs major mode.
- xml2ly, which imports MusicXML
- NoteEdit which imports MusicXML
- Rosegarden, which imports MIDI
- FOMUS, a LISP library to generate music notation
- `http://vsr.informatik.tu-chemnitz.de/staff/jan/nted/nted.xhtml`, has experimental export for LilyPond.
- `http://www.tuxguitar.com.ar/`, can export to LilyPond.
- `http://musescore.org` can also export to LilyPond.

Appendix A GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of 'copyleft', which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The 'Document', below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as 'you'.

A 'Modified Version' of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A 'Secondary Section' is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The 'Invariant Sections' are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The 'Cover Texts' are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A 'Transparent' copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file

format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not 'Transparent' is called 'Opaque'.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The 'Title Page' means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, 'Title Page' means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled 'History', and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled 'History' in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the 'History' section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled 'Acknowledgments' or 'Dedications', preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled 'Endorsements'. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as 'Endorsements' or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to

the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled 'Endorsements', provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections entitled 'History' in the various original documents, forming one section entitled 'History'; likewise combine any sections entitled 'Acknowledgments', and any sections entitled 'Dedications'. You must delete all sections entitled 'Endorsements.'

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an 'aggregate', and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License 'or any later version' applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

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.

Appendix B LilyPond index

\

\header in LaTeX documents 21

A

ABC 33

B

bugs 16

C

call trace 14
Coda Technology 34
coloring, syntax 8
command line options for `lilypond` 10
convert-ly 14

D

docbook 18
DocBook, music in 18
documents, adding music to 18
dvips 27

E

editors 8
emacs 8
enigma 34
error 14
error messages 14
errors, message format 14
ETF 34
External programs, generating LilyPond files 34

F

fatal error 14
FDL, GNU Free Documentation License 35
file searching 12
file size, output 9
Finale 34

H

html 18
HTML, music in 18

I

invoking dvips 27
Invoking `lilypond` 10

L

LANG 13
latex 18
LaTeX, music in 18
LILYPOND_DATADIR 13

M

MIDI 31
modes, editor 8
musicology 18
MusicXML 32

O

OpenOffice.org 30
options, command line 10
outline fonts 27
output format, setting 11

P

point and click 9
point and click, command line 11
PostScript output 11
preview image 23
Programming error 14

R

reporting bugs 16

S

Scheme dump 12
Scheme error 14
search path 12
`Staff` 31
SVG (Scalable Vector Graphics) 12
switches 10
syntax coloring 8

T

texi 18
texinfo 18
Texinfo, music in 18
thumbnail 23
titling and lilypond-book 21
titling in HTML 23
trace, Scheme 14
type1 fonts 27

U

Updating a LilyPond file 14

V

vim 8
`Voice` 31

W

warning 14