

The *lily* $\&$ *ly****ps*** Package

Version 0.2.0

Urs Liska

April 24, 2013

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in section A.2 on page 43, entitled “GNU Free Documentation License”.

The *lilyglyphs* L^AT_EX package is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This explicitly also applies for any code examples that may be part of this manual.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

A copy of the license is included in section A.1 on page 33, entitled “GNU General Public License”.

lilyglyphs is hosted at <https://github.com/lilyglyphs/lilyglyphs>
Contact information: lilyglyphs@ursliska.de

Contents

1	Introduction	5
1.1	Installation	6
2	Usage	7
2.1	Usage of predefined commands	7
2.2	Generic access commands	7
2.3	The optional argument: Scaling and placement	8
2.4	Examples	9
2.5	Dotted symbols	10
2.6	Optical size	10
3	Internals	11
3.1	Documentation of the generic access commands	11
3.1.1	Accessing <i>Emmentaler</i> Glyphs	11
3.1.2	Printing image files	13
3.2	How to write predefined commands	14
3.2.1	Commands that print single glyphs	14
3.2.2	Create Dotted Symbols	16
3.2.3	Create Multiply Dotted Symbols	18
4	Easily create commands with Python scripts	18
4.1	Generating Commands with Image Glyphs	19
4.1.1	Preparing the Input Files	19
4.1.2	Running the Script	20
4.1.3	Utilizing the results	21
4.1.4	Partial processing of the input file	23
4.1.5	Recreating Image Files	23
4.2	Generating Commands for <i>Emmentaler</i> Glyphs	23
4.2.1	Preparing the input file	24
4.2.2	Running the script and Utilizing the Results	25
5	Reference of Predefined Commands	27
5.1	Single Notes	27
5.2	Beamed notes	27
5.3	Clefs	27
5.4	Time Signatures	28
5.5	Numbers	29
5.6	Accidentals	29
5.7	Rests	30
5.8	Dynamic Text	31
5.9	Graphical Dynamic Symbols	32
5.10	Scripts	32

A	Appendix: Licenses	33
A.1	GNU General Public License	33
A.2	GNU Free Documentation License	43

1 Introduction

The package *lilyglyphs* is made for authors who want to include single musical elements in their \LaTeX texts. This is not meant to be used for full-fledged musical examples with staves, for which one would use packages like *musixtex* or the LilyPond software together with *lilypond-book*¹. Instead we are talking about inserting items like \flat or *sfz* within the paragraphs of continuous text.

To achieve this *lilyglyphs* aims at making the notation font and notational elements of the LilyPond notation software available to \LaTeX documents. LilyPond is a notation software package that is a competitor in producing the most beautiful musical engraving on the market, and one of the foundations of this beauty is its *Emmentaler* font. The *Emmentaler* fonts are accessed through the *fontspec* package, and *lilyglyphs* therefore relies on a \LaTeX distribution supporting *fontspec*. It was written using \XeLaTeX , but should also work with \LuaLaTeX – although this hasn’t been tested².

The *Emmentaler* font only provides a subset of LilyPond’s notational capabilities, as LilyPond creates many symbols by combining glyphs and drawing explicitly. To also access this kind of symbols we have chosen an approach that uses small pdf images to print glyphs not present in *Emmentaler*. There is an accompanying Python script to make the creation of new arbitrary symbols as easy as possible.

The *lilyglyphs* package was initiated and is maintained by Urs Liska, and is hosted as a git repository on GitHub³. If you would like to participate in its development, don’t hesitate to contact us.

For activating the capabilities of *lilyglyphs* just write `\usepackage{lilyglyphs}` in the preamble of your document. Then you can use the commands defined by the package or access glyphs directly through one of the generic access commands. Both are described in detail later, in this section you’ll only find a general overview of the possibilities.

Glyphs are scaled to fit normal text fonts, and the scaling automatically follows the scaling of the text. This also means they are influenced by \LaTeX ’s text size commands, so you can write something like `{\Large \flat}` to create

a larger \flat glyph than the normal \flat one or even something really `\Huge sfz` – which will probably spoil your line spacing if used within a paragraph. You can influence the size and vertical position of the glyphs – with the predefined commands as well as the generic ones – individually or globally through the use of optional arguments. This way you can also accomodate unusual text fonts that might otherwise not go well together with *lilyglyphs*.

¹<http://www.lilypond.org>

²Principally it is possible to extend the functionality so it would also work with plain \LaTeX , but as the package maintainer has neither experience nor active interest in this area this will only be implemented if there are volunteers who join us.

³<http://www.github.com/lilyglyphs/lilyglyphs> – Contact: <mailto:lilyglyphs@ursliska.de>

You can always combine commands (or glyphs created through generic access as shown in the next section) to create new commands for later reuse. But it is better to follow the documentation for package designers to create such commands in the way suggested also for package designers in section 3.2 on page 14. If you feel you have specific commands that would be a valuable addition to the package please don't hesitate to contact us.

1.1 Installation

As the procedures necessary to make this package work for you may vary depending on your operating system and L^AT_EX distribution, we can only give you some general hints on how to set up your system.

First of all you have to decide how you obtain the package. There are two main ways, and both have their pros and cons. You can either clone the GitHub repository or download one of the archived versions of the package.

The recommended way to get *lilyglyphs* is to clone the GitHub repository. If you consider contributing to the package it is preferable to first fork the repository, then clone into the fork and add the original repository as an additional remote. This way you can easily apply any changes, push them to your fork and send us a pull request.

There is one important issue to consider using the original repository: You will need working installations of both LilyPond and Python on your system to set the package up. This is because the pdf files that are used for the graphics driven commands aren't tracked in Git. When you have cloned the repository for the first time or have fetched updates that include new image files, you will have to run the `rebuild-pdfs.py` script in the `/py` scripts directory of the package. This will run LilyPond multiple times to create the necessary image files. Without these files the respective commands won't compile in your L^AT_EX documents. And if you think you might do without this subset of the available commands, consider that you won't even be able to compile the manual from its .tex source, which isn't included as a PDF file either – another issue to consider. But since you are reading this, you will probably have already managed to download the PDF version from GitHub ...

The big advantage of directly using the repository is that it is the easiest way to get updates that might occur quite often. Each time someone adds new items to the list of commands they will be instantly accessible through the repository, but regular releases will happen much less frequently. If updates should somehow break things in existing documents you can easily (and temporarily) check out the earlier version of the package that you used when originally writing your documents. Another advantage of the repository version is that you have all the tools at hand (especially the Python scripts) that make your life easier if you want to extend the package for yourself.

If you know that you will only use the *lilyglyphs* package, then you can download one of the archived versions from the download page on the package's GitHub site. These versions contain all necessary binary files to use the package

from your documents, but strips off any supplementary files that you might need to extend the package.

An “intermediate” solution would be to download an automatically archived version of the complete repository. This will contain the scripts and the supplementary files, but not the binary files. You should be able to download archives with the binary files for any released version from the download page.

The initial requirement is to make the package file (`lilyglyphs.sty`) available to your \LaTeX distribution. *The usual way to achieve this is to extract the archive or clone the repository to a place in the \LaTeX search path. This might for example be a subfolder of `texmf/tex/latex` in your home directory. On Linux you’d have to run `texhash /texmf` afterwards to update \LaTeX ’s cache.*

Of course the fonts have to be accessible to `fontspec`. Please refer to the `fontspec` documentation on how to make fonts visible to it.

*On a Linux system it may be enough to copy the **Emmentaler** font files to your home directory’s `.fonts` directory or in a new subdirectory of it, possibly running `fc-cache` afterwards to update the fontconfig cache. The font files are located in the `usr/share/lilypond/current/fonts/otf` subdirectory of your LilyPond installation. Eight `.otf` files (representing different optical sizes, see section 2.6 on page 10) are also included in the distribution archive of `lilyglyphs`.*

If you experience any issues during installation that would call for different instructions then please provide us some feedback.

2 Usage

2.1 Usage of predefined commands

In order to use predefined commands that have already been implemented you generally just have to enter the command to print the corresponding musical element, e.g. `\lilyTimeC` for a **C**.

The predefined commands are generally available in two forms, starred and unstarred (e.g. `\flat` and `\flat*`). The unstarred commands append a single space after the glyph while the starred ones provide the glyphs without trailing space. Use the unstarred versions usually in the continuous text, and the starred ones before punctuations, or if you want to combine multiple glyphs. There are some commands that don’t need this differentiation – notably Dynamics letters and numbers – because the underlying access technique works differently. For these commands you can just put whitespace after the closing curly brackets or not and \LaTeX will respect this. The documentation of the predefined commands always tells you whether there are starred versions or not.

2.2 Generic access commands

To print *Emmentaler* glyphs that aren’t covered by the predefined commands yet, `lilyglyphs` offers four generic access commands: `\lilyGlyph`, `\lilyText`, `\lilyGlyphByNumber` and `\lilyImage`.

Their mandatory argument is the content to be printed, which has to be given in a form specific to the respective command.

`\lilyGlyph` expects the OpenType glyph name. You can look up the glyph names in the Appendix of LilyPond’s *Notation Reference*⁴ or the somewhat reduced html page provided in the package download. The current pdf manual for *lilyglyphs* will eventually also contain a complete list of available glyphs, but this will only be updated every now and then (presumably when predefined commands of a new category are added).

`\lilyGlyphByNumber` expects the Unicode code of the glyph. You will generally not want to use this as the code positions aren’t guaranteed to stay the same with new versions of the fonts. There may be some uses for numerical access however.

`\lilyText` expects ordinary text as its argument. In fact it just switches the font to *Emmentaler* and then writes the string given as the argument. This only works for Dynamics letters, numbers and the glyphs \mp , \cdot , \cdot – as these glyphs are located at their ordinary character position in *Emmentaler*. But you can also enter any spacing commands (like `\hspace` or plain spaces). Keep in mind that this may result in line breaking inside your expression. If you need to prevent this you can surround your expression by a `\mbox`.

`\lilyImage` expects the basename of a usable image file. It then includes this file using the same optional argument mechanism as the other commands. What is somewhat special about this command is that it automatically scales the image relative to the current text font size, with being printed at its original size with `\normalsize`. You have to take care yourself that L^AT_EX finds and can handle the image file.

2.3 The optional argument: Scaling and placement

The generic access commands as well as the predefined commands allow an optional argument to be passed. This can contain a list of comma-separated options in `<key=value>` form that influence the appearance of the glyphs. Currently there are the `scale` and `raise` options.

`scale` changes the size of the glyph. As the *Emmentaler* glyphs are designed for a totally different purpose they often don’t fit very well in the context of continuous text. `scale` is given as a factor by which the default size is multiplied. With *Emmentaler* glyphs this has to be a positive number, otherwise you will get an error. But glyphs printed by `\lilyImage` (or predefined commands based on it) can also be scaled negatively. This results in an image that is rotated around the center of the bottom line of the original. You will therefore have to add an appropriate `raise` value (try e.g. 2 as a starting point). You will have to take some care about the horizontal spacing, as such a flipped image seems to use it with inverted direction. But you can safely put extra horizontal space

⁴<http://www.lilypond.org/doc/v2.16/Documentation/notation/the-feta-font>

after the image, and it is a valid and practical way to create two symbols from one image file.

raise changes the vertical placement of the glyph. The majority of glyphs is placed too low, so they need a positive **raise** value. **raise** is given as a decimal value without units, which is interpreted as *ex*, or x-height. As there is no *x* in a musical font, this is somewhat arbitrary, but it is a natural unit to scale with the font size. Usually you may start trying **raise** values between 0 and 0.5.

There are three levels where this influence may be effective: global setting, design time (with predefined commands) and command invocation.

At a **global level** the options are set to defaults of **scale=1** and **raise=0**, which basically means that the glyphs are unaltered. You can change these globally effective options at any time with `\lilyGlobalOptions{<options>}`. This may for example be necessary document-wide if you use a text font which doesn't harmonize well with *lilyglyphs*' default settings. But you can also use it if you for example need much bigger musical glyphs for presentation purposes.

At **design time** the package designers set the default options for their predefined commands, so they work right out of the box in most cases.

When **invoking** the commands within an actual document you can pass the options for the specific instance of the glyph.

The effective option values that are applied to a specific instance are calculated from all three levels. So if you pass an option during the command invocation, you don't set an absolute value, but modify the value already present. An option **scale=1.2** in a command doesn't mean that you apply 1.2 as the scaling factor to the original glyph, but that the glyph is printed 1.2 times the size it would have been printed without this option. So if you want to slightly increase the size of a single glyph you may pass 1.1 as the value to **scale**, even if you have earlier set the global scaling to 1.8 – the effective scaling will be $1.1 * 1.8$.

Technically speaking lilyglyphs applies a fourth layer of scaling with image files. It calculates a last scaling factor by multiplying the result of the above considerations with the ratio of the current font size versus the size of the `\normalsize` font.

2.4 Examples

Now it's time to see how you actually print *Emmentaler* glyphs using the generic access commands.

The *fermata* sign isn't implemented yet as a predefined command, so you can print it with `\lilyGlyph{scripts.ufermata}`: ♯ – we have looked up the name of the glyph in the documentation.

As this doesn't well fit in the line of text you will want to adjust its size and placement. This is done with the optional argument described in the previous subsection.

First we increase the size of the glyph with the **scale** argument. We find that a scaling factor of 1.4 seems suitable:

`\lilyGlyph[scale=1.4]{scripts.ufermata} - ♪`

As you can see the glyph is – as most *Emmentaler* glyphs are – placed too low, so you have to add the `raise` argument. A value of 0.3 seems fine – remember, the `raise` argument is interpreted as *ex*, but you don’t write down the unit.

`\lilyGlyph[scale=1.4,raise=0.3]{scripts.ufermata} - ♪`

You can now further see that the glyph is placed too far to the left – which is probably due to the fact that articulations are originally intended to be centered relative to their reference point (the note). So you have to add some more space before, which might be rational to enter in *ex*:

`\hspace{1ex}\lilyGlyph[scale=1.4,raise=0.3]{scripts.ufermata} - ♪`

If you want you can now simply enclose this definition in a `\newcommand` to be able to reuse it. However you are encouraged to follow our guidelines on how to create predefined commands as explained in section 3.2 on page 14. If you manage to write a command that you find useful for others also please submit it to us – or even better: if you figured out how to create commands in general, please join us.

2.5 Dotted symbols

With notes and rests you will find `\...Dotted` versions of commands. If you use significant scaling factors for the commands you have to take care about the gap between the glyph and the dot, because this sometimes doesn’t scale too well. Unfortunately one can influence neither the gap nor the scaling and raise factors of the dot independently. We have worked hard to enable the designer of a command to create rules how to scale the gap, but you still may run into problems here. In such a case you will have to either change the predefined command in the library or just create the dotted symbol from scratch.

`\...Dotted` commands have starred and unstarred versions.

You can write arbitrarily dotted symbols by adding `\lilyPrintMoreDots` after the dotted symbol. This command uses the existing dot settings (scale and raise) and prints another dot. By default it has a gap of 0.25ex, but you can override this by passing a number as an optional argument, which is interpreted as *ex*. For example if you take the command `\halfNoteRestDotted`, which prints a dotted half note rest: ♩. you can easily add more dots through `\halfNoteRestDotted*\lilyPrintMoreDots`: ♩.. (note the use of the starred and unstarred versions).

Please note that you should only call `\lilyPrintMoreDots` immediately after calling a `\...Dotted` command. Otherwise you may get surprising results or even errors because the underlying key-value variables are initialized wrongly or not at all.

2.6 Optical size

The *Emmentaler* fonts come in a set of eight “optical sizes”. These are variations of the font originally designed to be used at different point sizes. Generally you

can assume that fonts for larger sizes are somewhat lighter, while fonts for smaller point sizes give more weight on the paper.

`\lilyglyphs` gives you the option to access the available font versions, but it may make more sense to appreciate them as “weights” – although this is technically speaking or even conceptionally incorrect. *Emmentaler* has eight versions: 11, 13, 14, 16, 18, 20, 23, 26. If you conceive these as weights you would somehow order them from black (11) to light (26). You can switch the used optical size at any time in a document using the command `\lilyOpticalSize`, giving the number as an option. You have to make sure to supply a number corresponding to a font actually available on your system. Maybe this will someday also be available as an option to select for a single glyph. For now you would have to switch twice, before and after the glyph.

The optical size used by `\lilyglyphs` defaults to 16.

Known issues and warnings: Optical sizes don’t work with glyphs printed as images. If you *have* to use these glyphs in different weights, you will have to take care for it yourself. The general plan would be to create different versions of the glyph by creating different glyphs in LilyPond (presumably by using different staff sizes).

3 Internals

3.1 Documentation of the generic access commands

This section is essential for readers who want to understand how this package works internally, for example if they want to actively participate in its development. And it is recommended (but not exactly mandatory) for readers who want to write their own predefined commands or to contribute to the package by adding commands to it. The latter target group might be satisfied reading the next section about writing predefined commands, but understanding will be easier and deeper with this subsection.

In order to make the package `.sty` file easier to understand, its contents are split into multiple input files which are located in the `/commands` and `/core` subfolders. The most fundamental definitions are in the `keyval.inp` and `genericAccess.inp` files.

3.1.1 Accessing *Emmentaler* Glyphs

The command that actually prints glyphs from the *Emmentaler* fonts is `\lilyPrint`, defined in `genericAccess.inp`. It isn’t intended to be called directly within a document, but only from the predefined commands. It takes two arguments, the first – optional – being the comma-separated list of `<key=value>` pairs, the second the actual content to be printed.

```
\newcommand*{\lilyPrint}[2][]{%
```

```

\interpretLilyOptions{#1}%
\raisebox{{\lilyEffectiveRaise}ex}{%
  {\fontspec[Scale=\lilyEffectiveScale]{Emmentaler-\lilyOpticalSuffix}#2}%
}%
}

```

At first the command `\interpretLilyOptions` is called, where the options of the different levels are evaluated and calculated to their effective values. Then the content of `#2` is printed, within a `\raisebox` and with the currently selected opticals version of the *Emmentaler* font.

```

\newcommand*{\interpretLilyOptions}[1]{%
  \setkeys{lilyCmdOptions}{scale=1,raise=0}%
  \setkeys{lilyCmdOptions}{#1}%
  \pgfmathsetmacro{\lilyEffectiveScale}{%
    \lilyGlobalOptions@scale * \lilyCmdOptions@scale * \lilyDesignOptions@scale}%
  \pgfmathsetmacro{\lilyEffectiveRaise}{%
    \lilyGlobalOptions@raise + \lilyCmdOptions@raise + \lilyDesignOptions@raise}%
}

```

`\interpretLilyOptions` is defined in `keyval.inp`.

The `<key=value>` mechanism is achieved using the `keyval` package as the most basic solution available. *If this can be implemented more elegant, extensible or powerful using other packages, e.g. `pgfkeys`, we'd appreciate any input.* It uses three families of keys, corresponding to the three levels of options: `lilyGlobalOptions`, `lilyDesignOptions` and `lilyCmdOptions`.

In a first step the keys for the actual command options are initialized to a neutral state. This is necessary because otherwise options that aren't actually present in the command invocation were in an uninitialized state or in the state set by the last occurrence of the option. After this the command options are set to the actual state given in the command (i.e. the command the end user writes in the document). Options that are explicitly given override the default state while options that are not present don't affect it. Finally the effective values of the options are calculated from the global, the design and the command options. The scaling values are multiplied, the raise values added. While the command options have just been determined, the global options are valid globally (and can be changed globally) and the design options have been set by the command that actually called `\lilyPrint`. This is the reason why `\lilyPrint` should never be invoked directly – the design options would be in the unknown state of the previous invocation of `\lilyPrint`.

The next higher level are the three generic access functions `\lilyGlyph`, `\lilyGlyphByNumber` and `\lilyText`, defined in `genericAccess.inp`. They are very similar and differ only in the way they determine the actual content to be printed. As stated in the end user part of this documentation they expect two arguments, the optional `<key=value>` pair list and the contents. As a first step the commands initialize the design options to a neutral state,

because the “design” of the generic glyphs has to be neutral by design. In the second step they invoke `\lilyPrint`, passing the optional argument along and determine the printed content individually: `\lilyGlyph` calls the helper function `\lilyGetGlyph`, `\lilyGlyphByNumber` calls `\lilyGetGlyphByNumber`, while `\lilyText` just passes its contents argument unchanged to `\lilyPrint`.

These helper functions are important because most predefined commands call one of them to select glyphs from the *Emmentaler* fonts.

`\lilyGetGlyph` takes the glyph name as found in the LilyPond documentation. `\lilyGetGlyphByNumber` takes the Unicode character index of the intended glyph. But be aware that the Unicode index may change at any time with new versions of the *Emmentaler* font, so it usually isn’t a good idea to access glyphs through their index. There may be some uses for numerical access, however, e.g. to iterate through a range of glyphs.

3.1.2 Printing image files

`\lilyPrintImage`, defined in `genericAccess.inp`, is the command that prints glyphs from a supplied image file. It actually is quite similar to `\lilyPrint`, with only the extra consideration of scaling the image to the text font size.

```
\newcommand*{\lilyPrintImage}[2][]{%
  % interpret optional argument
  \interpretLilyOptions{#1}%
  % determine scaling factor to accomodate the current font size
  % (as images don't scale automatically with the font)
  \lilyScaleImage%
  % Print the image in a raisebox
  \raisebox{{\lilyEffectiveRaise}ex}{%
    \includegraphics[scale=\lilyImageEffectiveScale]{#2}%
  }%
}
```

First the command calls `\interpretLilyOptions`, which is the same as with `\lilyPrint`. But as an additional step `\lilyScaleImage` is called, and finally it uses `\lilyImageEffectiveScale` as the scaling factor instead of `\lilyEffectiveScale`. I won’t explain this in detail, but in effect it multiplies the `\lilyEffectiveScale` calculated before with the ratio of the current font size to the `\normalsize` size.

What is finally given as the content to be printed is the basename of an image file. With \LaTeX this can be a file in .pdf, .png or .jpg format, but we highly recommend using .pdf files for sake of printing quality.

As with printing *Emmentaler* glyphs there is no handling of design time options here, and for that reason you should never call this command directly from a document. Please use `\lilyImage` instead which does the same as `\lilyPrintImage` but additionally defaults the design time options to neutral values.

3.2 How to write predefined commands

Writing your own predefined commands is actually quite straightforward – and identical if you want to write a command for your document or for inclusion in the package. So if you find yourself creating predefined commands that you think are useful for general use, don't hesitate to submit them to us.

3.2.1 Commands that print single glyphs

Let's review an example of a predefined command, the `\doublesharp`.

```
% "accidentals.doublesharp"
\newcommand*{\doublesharpBase}[1] [] {%
    \setkeys{lilyDesignOptions}{scale=1.5,raise=0.35}%
    \lilyPrint[#1]{\lilyGetGlyph{accidentals.doublesharp}}%
}

\newcommand*{\doublesharp}[1] [] {\doublesharpBase[#1] }

\WithSuffix\newcommand\doublesharp*[1] [] {\doublesharpBase[#1]}
```

As you can see there are actually three commands, a base command and the starred and the unstarred versions. But let's start with the original command.

We use the starred version of `\newcommand`, because we always know that our commands are restricted to single paragraphs. We declare that our command accepts one optional argument, which defaults to empty. This argument can take the list of `<key=value>` options. When writing the commands, please take care not to omit the `%` characters at the line endings, as they prevent unwanted whitespace to be introduced in the output.

In the second line we define the design options for your command. In the example the designer has decided that a doublesharp glyph should be scaled to 1.5 and placed 0.35 ex above its default level.

The third line calls the internal `\lilyPrint` command. It passes the optional argument, with which the end user can override (i.e. modify) the designed values. As the `♯` is a glyph that has to be selected by its glyph name, we call `\lilyGetGlyph`, supplying the glyph name found in the documentation. The result of this command is passed as the `#2` to `\lilyPrint`.

To summarize: Writing a predefined command for printing a glyph from *Emmentaler* involves just two steps, setting the design time options and calling `\lilyPrint` with the appropriate `#2` argument.

If we have done it right, the new command prints the desired glyph without any trailing space. The side-effect is that \LaTeX will ignore any whitespace after the command. While it is always possible to write a pair of curly braces after or around the command to allow trailing whitespace, we decided to offer the starred and unstarred versions of commands. The unstarred versions provide a trailing space and are therefore intended for general use in continuous text.

The starred versions don't have this space and are intended for use before punctuation marks, in words or in combinations with other glyphs. For these we need the `\WithSuffix` construct that is shown in the example. We name the commands identical to the original command except for the trailing `*`. We give the same set of one optional argument, and in the command itself we just call the base one, passing the `#1` argument along and add a trailing space.

If you know the Unicode number of the desired glyph you can call `\lilyGetGlyphByNumber` instead of `\lilyGetGlyph`, but you can't be sure this number will stay the same forever.

Creating commands using image files is practically the same and even simpler. If you look at the definition of a `\crotchet`,

```
\newcommand*{\crotchetBase}[1] [] {%
    \setkeys{lilyDesignOptions}{scale=0.9,raise=-0.2}%
    \lilyPrintImage[#1]{crotchet}%
}
\newcommand*{\crotchet}[1] [] {\crotchetBase[#1] }
\WithSuffix\newcommand\crotchet*[1] [] {\crotchetBase[#1]}
```

you will notice that the only differences are that the actual printing is done with `\lilyPrintImage` instead of `\lilyPrint` and that therefore the basename of the image file can be passed directly. In section ?? on page ?? you will see a tool that allows to create numerous image commands quite easily.

As a last example we will look at the definition of ***rfz*** `\lilyRFZ`.

```
\newcommand{\lilyRFZ}[1] [] {%
    \mbox{%
        \lilyDynamics[#1]{r\hspace{0.035ex}fz}%
    }%
}
```

As mentioned in section 5.8 on page 31, `\lilyDynamics` is just a wrapper around `\lilyText`, setting the `scale` factor to 1.5. While the other generic commands only print single glyphs, `\lilyText` can print 'plain text', so usually there is no need to write predefined commands only to combine letters to a single command. In some cases this may however be necessary. In the given example of `\lilyRFZ` we need to apply a little bit of extra space between the ***r*** and the ***z***. We see that we can insert a `\hspace` command between the letters without any problems. But as it turns out \LaTeX may now decide to insert a line break at any time, so we have to additionally enclose this call to `\lilyDynamics` in a `\mbox`. The command just passes the optional argument to `\lilyDynamics`, so you can use these arguments as usual. This example is meant to encourage you to experiment with the definition of predefined commands. This is pretty much the same as usual when inventing new commands. You just have to deal with

the design time options, the optional argument and getting the correct input as the second argument.

If you had created this command you would have noticed that you can't get any whitespace after it – it is silently ignored. So you would want to create starred and unstarred versions of the command, which works as in the first example. In *lilypond* this is in fact implemented this way.

3.2.2 Create Dotted Symbols

It is not exactly trivial to create dotted symbols as predefined commands. Of course you can always use `\lilyDot` to print LilyPond's dot glyph, but if you want to create commands that combine a glyph and one or more dots you encounter two difficulties: You can't apply the optional arguments independently on the two items, and there may be issues with the scaling and the gap between the two glyphs.

There is some infrastructure in *lilypond* (defined in the file `dotted.inp`) to facilitate dealing with dotted symbols, but this implementation may not be completely satisfactory so far. Great care has been taken to hide as much complexity as possible in the mentioned file, in order to make the definition of actual commands as clean and concise as possible.

Let's analyze the implementation of a dotted half note rest:

```
% Dotted half note rest
\newcommand*{\halfNoteRestDottedBase}[1][]{%
  % define the optional arguments for the dot
  \setkeys{lilyDesignOptions}{scale=0.8,raise=0.2}%
  % Calculate effective scale/raise and the hspace for the dot
  \lilySetDotOptions[#1]{0.05}{0.5}{0}%
  % Print the rest and then the dot
  \halfNoteRest*[#1]\lilyDotSpace\lilyPrintDot
}
```

As you will note the command is a `...Base` command that will be have to be complemented by the starred and unstarred versions, but we will silently ignore this issue here (see section ?? on page ??).

The command takes the usual optional argument that can contain `<key=value>` pairs. This applies to the dotted symbol as a whole. Please note that in the last line the predefined command `\halfNoteRest[#1]` is called and passed the optional argument. You can only use this technique of creating dotted symbols on top of correctly implemented predefined commands.

The first thing you have to do is to define the `DesignTimeOptions` for the dot. They are relative to the original design of `\lilyDot`, and you have to adjust them so the dot suits the main glyph in its size and vertical position (you can ignore the horizontal spacing for now):

```
% define the optional arguments for the dot
\setkeys{lilyDesignOptions}{scale=0.8,raise=0.2}%
```


You should always set these options because otherwise you might get strange results or error messages.

The next step is to call a quite complex command `\lilySetDotOptions` that sets several options for the dot:

```
% Calculate effective scale/raise and the hspace for the dot
\lilySetDotOptions[#1]{0}{0.5}{0.4}%
```

This command takes one optional and three mandatory arguments. The optional argument is just the one that is written in the \LaTeX document and that is passed into the function. The remaining three arguments control the horizontal spacing between the main glyph and the dot. As we now have two individual elements we have to control the gap between them explicitly as it doesn't scale relative to the `scale` argument by itself. The relation between the `scale` factor and the horizontal gap can be understood as a curve (mathematically spoken: a 2nd order function).

The first (mandatory) argument sets the intensity of the curve. A value of 0 will result in a linear relation (no curve at all), that is when doubling `scale` the gap will be exactly twice as wide. Positive values will result in larger gaps for larger `scales`. As this is a quadratic function you will want to start with very small values or 0.

The second argument sets the general (linear) steepness of the curve. A value of 1 means that by increasing `scale` by 1 the gap will be wider by 1 ex . 0.5 seems a good starting point for this argument.

The last argument is an offset in ex for the whole curve which is independent from the scaling. You can use it to accomodate specifically wide or narrow glyphs.

`\lilySetDotOptions` takes all these informations, calculates some settings for the dot and stores them in internal variables that can be used by subsequent commands. Please understand that they may be partially or totally overwritten by the next use of *any* predefined command. So you have to call this command immediately before actually printing the dot, otherwise it may or may not provide satisfying results.

The final line actually calls three commands:

`\halfNoteRest*[#1]` prints the already defined main glyph. Note the use of the starred version, because the trailing space of the unstarred version would make it much harder to determine the parameters for the gap.

`\lilyDotSpace` prints a horizontal space that is determined by the previous call to `\lilySetDotOptions`.

`\lilyPrintDot` finally prints the dot with the settings just defined.

```
% Print the rest and then the dot
\halfNoteRest*[#1]\lilyDotSpace\lilyPrintDot
```

This was an explanation from the perspective of designing new predefined commands. If you want to know how this is implemented internally, please look at the generously commented file `core/dotted.inp`.

Known issues and warning: One issue that hasn't been addressed yet is the vertical placement of the dot when scaled. The dot is positioned relatively to the baseline of the text, and the main glyph may have a different center point. So when scaling the main glyph may seem to behave differently from the dot.

If you want to create a dotted version of a glyph that is printed from an image file it will generally be easier and more reliable to create a command using a new image file.

3.2.3 Create Multiply Dotted Symbols

There is no need to define additional versions of glyphs with more than one dot. For this purpose we have implemented the command `\lilyPrintMoreDots`. This prints a dot with the same characteristics as the preceding one from a dotted command (but remember that there shouldn't be any calls to other predefined commands in between). The horizontal gap between the dots scales linearly with a default of $0.25ex$ per unit of `scale`. But if you pass an number as an optional argument this is interpreted as a different gap in *ex*. This command is available in a starred and an unstarred version. Use the unstarred version when you need a space after the symbol, the starred one when you have a punctuation or another dot after it.

```
\halfNoteRestDotted*\lilyPrintMoreDots*! ─...!
\halfNoteRestDotted*[scale=1.5]\lilyPrintMoreDots and ─... and
```

4 Easily create commands with Python scripts

As we have seen in section 3.2.1 on page 14 it isn't really complicated to create predefined commands. But we have developed a set of tools and templates to streamline this process even further. With them it is mostly a matter of telling what you want, naming it and running a Python script. But keep in mind that you can always do it manually if the pre-set workflow imposes any restrictions that you don't want to accept. And it is nevertheless a good idea to have read section 3.2 on page 14 to have a good understanding of what is going on.

The process basically consists of writing a definitions file (something like a template), calling a script and then putting the results to a useful place. However it slightly differs if you create commands for using images or for using *Emmentaler* glyphs. Both ways require a working Python 2 installation to work, creating commands with images additionally involves running LilyPond. And the necessary scripts (and partially source files) are only available in the Git repository version of the package. If you don't have Git installed but want to profit from them you can download zipped versions of the complete repository from GitHub. You can't create image glyph commands at all without having LilyPond installed on your system, but you don't need Python in order to create new commands – it just provides a much more convenient workflow.

4.1 Generating Commands with Image Glyphs

The heart of the toolset is the Python script `buildglyphimages.py`, which is located in the `/py` directory. It reads in an input file with one or more command definitions in it, generates compilable LilyPond source files from them and uses LilyPond to create small pdf files containing the images to be included in the \LaTeX file. Then the script creates a new \LaTeX file with command definitions and example text, which can then be used to fine-tune the commands. Finally you will have to move the generated commands to some useful location.

But let us start with looking at the form of the input files.

4.1.1 Preparing the Input Files

`buildglyphimages.py` doesn't expect regular LilyPond source files as its input, but rather a file with one or multiple 'snippets' in it. We will later see that it makes sense to provide compilable Lilypond files, though.

Input definition files are to be stored in the subdirectory `definitions` of our `/glyphimages` directory.

The script parses the input file line by line until it finds a line that contains the special key `%%lilyglyphs`. Then it starts reading in an entry for a single glyph command.

Any subsequent line that starts with the percent sign as a LilyPond comment is interpreted as a comment. This comment will later be used for marking the command in the \LaTeX file. If a line contains the special key `%%protected`, the script skips this command. You should use this key whenever you consider a command finished. This won't only save the time of the LilyPond compilation, but will also prevent the existing command to be generated again in your output file.

If a line contains one of the keys `scale=` or `raise=` the value after the equals sign is interpreted as a new default value that the generated command gets as its design time options. These values are also kept for subsequent command entries until the file is finished or the script finds a new entry – which would replace the value. The idea behind this option is that it is likely that one defines a set of related commands within an input file. And it is likely that these related functions share common default values. For example all single notes were created with a default scaling of 0.75. Without this option you would have to adapt your result file for any single command, with it you can decide which default values you prefer for the commands at hand. The setting of option defaults also works if a command is marked as “protected”. Generally this is an advantage since you don't have to take care to correctly update the file when you mark something as protected. But there may be cases where you will want to remove the defaults setting from a `%%protected` clause.

Afterwards you provide the actual LilyPond command in the form of a variable holding a musical expression. The first line should contain exactly the name, the `'=` and the opening curly brace. The following lines are read as LilyPond code until the parser finds a line starting with the closing curly brace.

Here you can see a concise example of a lilyglyphs entry section:

```
%%lilyglyphs
% crotchet with upward stem
crotchet = {
  g'4
}
```

The name of the variable is very important, because it will also be used as the file name of the image file and the \LaTeX command name.

Warning: So far the parser isn't very smart. It just assumes that the input file it parses is correct. If you provide code with deviations from this explanation, the script will probably produce erroneous results or just stop working. Probably it won't do any harm, but we can't make any promises on that. Please consider the script as being in a **very experimental** state.

You can define as many entries in one file as you like – they will all be processed at once. Of course it is recommended to combine a coherent set of commands.

Everything that is outside of the lilyglyph entries is ignored by the script, so you can make use of these places to make a usable LilyPond file out of it. You can start off with the file `definitions/_template.ly`. After the definition of the lilyglyph entry it places a markup, then it copies the variable to a new variable named `symbol` and finally includes a score from a special include file `score.ily`. This will print your variable as the only element in a new score block that is prepared not to print staff, time signature and clef. You can repeat this several times and have LilyPond produce a sheet with all defined symbols. This way you can finish your design in “pure LilyPond” before actually going on to the \LaTeX part.

4.1.2 Running the Script

If you have prepared your input file you can run the `/py/buildglyphimages.py` script, passing the file name (without the path) along with the option `-i` (or `--input`). The script will look for the file in the `/glyphimages/definitions` directory.

In a first step the program parses this file and extracts information about the command definitions that haven't been marked as protected. Then it generates one compilable LilyPond source file for each command and saves it to a subdirectory of `/glyphimages/generated_src`. Each file is named according to the name of the command you specified in the input definition file, and the subdirectory is named according to the input file. So if you combined several glyphs in a file `singlenotes.ly` the results will be stored in a `singlenotes` subdirectory. And if you run the script again on a file with the same name but other definitions in it, it will add the new items to the directory.

Now these generated source files are compiled using LilyPond. The command line option `-dpreview` takes care of producing a pdf file with the smallest possible bounding box around its content. The directory is then cleaned up, and the resulting small PDF files are moved to the corresponding subdirectory below `glyphimages/pdfs`.

The final task of the script is to generate a \LaTeX file that is stored as `/stash_new_commands/imagages/newImageGlyphs.tex`. Any existing file (i.e. the result of the previous run) will be overwritten.

4.1.3 Utilizing the results

If you have successfully run the script you have the following results:

- LilyPond source files in the `/glyphimages/generated_src/INPUT_FILE_NAME` folder (one for each command),
- PDF files in the `/glyphimages/pdfs/INPUT_FILE_NAME` folder and
- the file `/stash_new_commands/images/newImageGlyphs.tex`.

You shouldn't touch the first two items but go on and open the `.tex` file. Keep in mind that this file will be overwritten by the next run of the script, so if you want to make any changes and/or keep the generated contents, you should immediately rename the file. It is a good idea to name it according to the definitions input file because these two files form a natural pair. This file is a working \LaTeX document that uses the `lilyglyphs` package. It contains `\newcommand` definitions with starred and unstarred versions, one for each definition of the input file that hasn't been skipped in the process. The visible part of the document contains a documentation table (as used throughout this manual) with all generated commands and example text for each generated command.

So what are we going to do with this file? Well, it depends on how/where you want to eventually use it, but the first step will be to fine-tune the commands.

The script generates default values for the design time values of the optional argument, and it would of course be purely random if it would be perfect right away. The first thing the generated example text does is showing that the command actually works. But its more important purpose is to print the new glyph in different contexts: in continuous text, before punctuations, at the beginning of a line etc. You should use these blocks of example text to adjust the arguments in the `\setkeys{lilyDesignOptions}` clause. You should specifically keep an eye on spacing issues: Does the glyph affect line spacing, is the "kerning" of the glyph correct? On the other hand you should probably try to keep corresponding glyphs at an equal scaling. Besides tweaking the optional argument values you can add space before or after the glyph, possibly for the starred and unstarred commands independently. Keep in mind that you may as well use negative `\hspace`.

Sometimes you will find that a rather tall glyph is difficult to accomodate because you have to scale it down so much in order not to affect the line spacing

that it looks unnaturally small. In such a case you might consider going back to the LilyPond side to review the design of your glyph (for example the glyphs for the single note commands have shortened stems). Please make sure that you understand the implications of the following section about partial processing.

If you are satisfied with the new commands you will have to move them to a useful place. In any case you should keep copies of the input files as well as the current .tex file as a reference. If you have put the command definitions in another file (e.g. in one of the .inp files of the *lilyglyphs* package) you have to remove them from the generated file, otherwise you will get compilation errors.

If you are just creating the commands for your personal use, you can either copy the command definitions to the preamble of your working L^AT_EX document or to a style file you might maintain. But of course we would be happy if you decided to contribute them to the package, for which there are several possible ways:

Probably the simplest way is to send the definitions file and your processed results file as an email to the package maintainer. He would then run the script to generate the intermediate LilyPond and the pdf files and copy your edited command definitions to appropriate places in the package.

If you are working on a fork of the GitHub repository (as is recommended for potential participators) you could do even better by incorporating the new commands directly and sending us a pull request. This works by completing the following steps:

- Move/copy the command definitions (with all comments) to appropriate .inp files in the **commands** subdirectory of the package root. If you find you should create a new .inp file because your commands belong to a new category please take the existing files as a model and add an appropriate `\input` statement to `lilyglyphs.sty`. Please leave your definitions file in the `glyphimages/definities` folder, with all commands marked as 'protected' (see next section for details). Please also leave the generated LilyPond files in their subdirectory (you would have a hard time finding them anyway). For any new files you might create please add the usual copyright comment at the beginning.
- *Please* add documentation for your added commands in the manual (`lilyglyphs.tex`). Find the appropriate subsection in the “Predefined commands” section and add information to it. At least we need the entries in the documentation table, but if there is anything special to note about the commands please explain this too. If you add to an existing subsection you may copy the rows from the table in the file generated by the script, otherwise you should copy the whole table (but update the caption and label fields appropriately).
- If you are happy with the result you can push to your fork of the repository and send a pull request through the GitHub web site. It would be nice if we would receive such a pull request as a single commit (use `rebase -i`), as this is much easier to check.

4.1.4 Partial processing of the input file

You are encouraged to put a coherent set of multiple command definitions in one definitions file and keep this file, as it is the source from that everything can be rebuilt at any time. But there will be occasions when you don't want the Python script to do all the work over and over again. If you add a new command to the input file (and you *should* add it to an existing file if it belongs to the same category) you only want the new command to be processed. Or if you are working on the fine-tuning of the commands (as described in the previous section) and decide that you have to change the LilyPond definition of a single glyph you also only want to reprocess this one. For this purpose you can mark entries in the input definitions file as “protected”, in order to prevent them to be newly processed by the script. To do this you enter a line containing `%%protected` in your entry definition (note the double percent sign and the missing space after them). You can see an example for this in the `_template.ly` file – because the example entry shouldn't be processed at all. From now on the marked command won't be processed anymore. This means that the LilyPond source file won't be generated again, LilyPond isn't run again for the script (which would be the most annoying thing), and the \LaTeX commands won't be generated again. The output file will of course be overwritten (you have renamed it before re-running the script, isn't it?) but it won't be cluttered with commands that you have already dealt with earlier.

4.1.5 Recreating Image Files

There are occasions when you have to (re)create the pdf files that serve as the glyph images without having to regenerate the \LaTeX commands or the LilyPond source files. If you clone the Git repository then the pdf files are not present because they aren't tracked by Git. Or if you pull updates from the remote repository that introduce new glyph images the pdf files won't be included for the same reason. In that case you can run the script `rebuild-pdfs.py`, which is also located in the `/py` directory. This script essentially compares the directories with the generated LilyPond sources and with the created pdf files, and if it finds a source without a corresponding pdf it will call LilyPond to recreate it. Of course this script will only work with a working LilyPond installation.

4.2 Generating Commands for *Emmentaler* Glyphs

Generating commands that print *Emmentaler* glyphs is very similar to the process with images described in the previous section, and it actually is even simpler, because there are no intermediate LilyPond files and compilations involved. All there is to it is: create an input definitions file, run the `/py/genGlyphCommands.py` on it and fine-tune the generated \LaTeX file. So in this chapter we mainly discuss the differences to the process as was already described.

4.2.1 Preparing the input file

The structure of the input file for the `genGlyphCommands` script is actually simpler than that for the `buildglyphimages` script. This is due to the fact that we wanted the latter to be valid LilyPond source file as well – a restriction we don’t need to consider here. As there is no creation of secondary files with LilyPond involved, the input file is much less important than with the image glyphs process. There you need to keep the input files in order to be able to re-create or change the image files at any given moment, but here you only need to keep them if you want to keep track of what you have done.

The script will look for the input file in the `/stash_new_commands/emmentaler` directory, so you have to save it to that directory. It can have any name and any or no file extension.

The input file should consist of one or more command definitions. Each command definition is composed of a set of lines with `key=value` pairs and finished by an empty line. Please note that you shouldn’t use whitespace around the equals sign. Python and \LaTeX style commands are ignored, so you can use them if you want. But keep in mind that empty lines have the special meaning of closing a command definition. Therefore it is important that your file ends with a complete empty line, otherwise the last entry will be discarded.

A command entry consists of several mandatory or optional lines. The order doesn’t matter, but it is surely advisable to stick to one style. The following items are possible/necessary:

- **cmd** (mandatory): Specifies the command name. You have to make sure that it is a valid \LaTeX name and that it isn’t in use already.
- **comment** (optional): You can pass a single line comment that will be used before the command definition. (If you want to have a multiline comment instead you can insert line breaks with `\n`.)
- **element** (mandatory): The actual element to be passed to the internal printing functions. The possible kind of values depends on the type of the command, as described with these types.
- **type** (mandatory, but defaulted): The type determines the internal printing command to be used with the command. Please refer to section 2.2 on page 7 for more information. The option is mandatory but defaults to “glyphname”, so you can skip it for the most common case that the glyph is called by name.
 - **“glyphname”**: The glyph is selected by its glyphname, which is what you have to specify as “element” (e.g. “accidentals.sharp”). You can look the glyph names up in LilyPond’s documentation or in the glyph list contained in the package.
 - **“number”**: The Unicode number is used to determine the glyph.

- **“text”**: The content is passed as plain text (works only for Dynamic letters, numbers and + - , .)
- **“dynamics”**: The content is also passed as plain text, but the scaling defaults to 1.5, which is quite suitable for dynamics.
- **“image”**: The command prints an image file. The usual way for creating such a command would seem to be using the other script, `buildglyphimages.py`, but this way you can also create commands using image files you obtained/created from any other source than LilyPond. You can even use this to create non-musical commands that print images and profit from *lily~~pond~~bs*’ infrastructure (like the automatic scaling with text size). You can use .pdf files (preferred) or .png or .jpg images for this purpose. You are encouraged to save these files to a dedicated folder different from the one where the LilyPond images are stored automatically. This folder has to be in view of L^AT_EX, but if you use one inside the *lily~~pond~~bs* directory this isn’t an issue. Of course such externally created image files aren’t automatically tracked by Git and therefore aren’t promoted to other users of the package.
The “element” for an image driven command is the plain file name without extension or path.

- **scale / raise** (optional): If one of these items is found they are set to be the new default value for the command’s optional argument. As described in section 4.1.1 on page 19 the value is kept until the end of the file is reached or the value is set newly.

Here you can see two examples of command entries:

```
cmd=fermataDown
element=scripts.dfermata
comment=downward fermata
# type glyphname is implicitly used

cmd=rinforzando
element=rfz
comment=Rinforzando, to be kerned
type=dynamics
```

4.2.2 Running the script and Utilizing the Results

You can run the `/py/genGlyphCommands.py` script and simply pass the prepared input file as the first (and only) parameter. The script will only look for the file in the `/stash_new_commands/emmentaler` directory.

The program parses the input file and creates a new file `/stash_new_commands/emmentaler/newGlyphCommands` that is exactly like the one created by `buildglyphimages.py` (see section 4.1.3 on page 21). It doesn’t create any intermediate files and doesn’t invoke LilyPond, however.

You finish off the process exactly like with the image driven commands, so we don't have to repeat that here.

5 Reference of Predefined Commands

The following sections document the predefined commands that already have been implemented. They generally contain a list of all glyphs from the *Emmentaler* documentation, explanations on the use of the commands (if necessary) and a table listing the implemented commands. Be aware that the lists and tables may be on different pages than the section header. Sections whose implementations haven't started yet may be present or not, so if you are missing glyphs please refer to LilyPond's documentation.

The documentation contains information if the commands are available in starred and unstarred commands, and if they are based on image files or *Emmentaler* glyphs – image commands are mentioned explicitly.

5.1 Single Notes

Single notes may be one of the most frequently used glyphs. They aren't present in *Emmentaler*, so they all are realized using included pdf image files. They have starred and unstarred versions. See table 1 on the next page for the available predefined commands.

5.2 Beamed notes

We will only provide a few complex symbols like beamed notes for default use. Of course one could have the wish for indefinite variations like notes with variable beam slope. But as long as it isn't possible to make this parametrical and draw them directly from within *lilyglyphs* it is probably a good idea to stick to a few basic commands. However it would be quite simple to create “plugin modules” that provide a series of related symbols, and maybe we will think about such an extension. For now see table 2 on page 29 for the implemented commands. They are available in starred and unstarred versions.

5.3 Clefs



























Some of the clef glyphs are among the few that are too large by default. You couldn't use a G clef within continuous text without severely  damaging line spacing. But if you scale them to a size that doesn't disturb line spacing, they look quite disproportionate, especially when combined with other elements: . Therefore we provide the clefs in two forms, a standard form which can be problematic in continuous text, and an -Inline version which looks somewhat funny but can be used within the line. The clef commands are available in the starred and in the unstarred form. See table 3 on page 29 for the available predefined commands.

Table 1: Single Notes

	<code>\crotchet</code>
	<code>\crotchetDown</code>
	<code>\crotchetDotted</code>
	<code>\crotchetDottedDown</code>
	<code>\crotchetDottedDouble</code>
	<code>\crotchetDottedDoubleDown</code>
	<code>\halfNote</code>
	<code>\halfNoteDown</code>
	<code>\halfNoteDotted</code>
	<code>\halfNoteDottedDown</code>
	<code>\halfNoteDottedDouble</code>
	<code>\halfNoteDottedDoubleDown</code>
	<code>\quaver</code>
	<code>\quaverDown</code>
	<code>\quaverDotted</code>
	<code>\quaverDottedDown</code>
	<code>\quaverDottedDouble</code>
	<code>\quaverDottedDoubleDown</code>
	<code>\semiquaver</code>
	<code>\semiquaverDown</code>
	<code>\semiquaverDotted</code>
	<code>\semiquaverDottedDown</code>
	<code>\semiquaverDottedDouble</code>
	<code>\semiquaverDottedDoubleDown</code>

5.4 Time Signatures

Emmentaler provides two “real” glyphs for time signatures, the **C** and the **⌘**. The commands `\lilyTimeC` and `\lilyTimeCHalf` have starred and unstarred versions.

Numerical (single and compound) time signatures can be printed using `\lilyTimeSignature: $\frac{4}{4}$` . This command expects two mandatory arguments: numerator and denominator. They are treated as `\lilyText`, so you can easily write compound time signatures like `\lilyTimeSignature{4 + 7}{8}: $\frac{4+7}{8}$` .

`\lilyTimeSignature` doesn’t need a starred version as it behaves as you would expect: It doesn’t print a trailing space but respects any whitespace after the closing bracket.

Table 2: Beamed Notes





	<code>\twoBeamedQuavers</code>
---	--------------------------------

Table 3: Clefs

	<code>\clefG, \clefGInline</code>	<code>clefs.G</code>
	<code>\clefF, \clefFInline</code>	<code>clefs.F</code>
	<code>\clefC, \clefCInline</code>	<code>clefs.C</code>

Known issues and warnings: `\lilyTimeSignature` also expects the optional argument as the other commands, but it doesn't understand the `raise` option correctly. The box with the time signature is vertically centered so it should generally be OK, but if you for some reason have to change its vertical position you should surround it manually by a `\raisebox`.

5.5 Numbers

Numbers can be entered with the already known `\lilyText` command. Access through the glyph names is possible but not necessary. Therefore we don't provide predefined commands for them. There are all ten numbers available. With the default scaling of 1.0 they generally fit as lowercase letters like **0123456789** `\lilyText{0 1 2 3 4 5 6 7 8 9}`. For Uppercase letters you can start trying a scaling of 1.3. A future version of the package will provide convenience functions with default scalings for upper/lowercase letters, fingerings, figured bass numbers, time signature numbers etc.

A special case are four glyphs that are grouped among the numbers: `+ - . ,` (plus, hyphen, fullstop and comma). These are also accessible through `\lilyText`, the example in the previous sentence being written as `\lilyText[scale=1.5]{+ - . ,}`.

5.6 Accidentals

The `\natural ♮`, the `\flat ♭` and the `\sharp ♯` replace the respective commands from standard L^AT_EX. Please note that all the accidentals are designed at the same scaling in order to allow a uniform appearance. You will however have to check if they don't affect an even line spacing.

Table 4: Time Signatures

C	<code>\lilyTimeC</code>	<code>timesig.C44</code>
♯	<code>\lilyTimeCHalf</code>	<code>timesig.C22</code>
$\frac{7}{8}$	<code>\lilyTimeSignature{7}{8}</code>	
$\frac{3+4}{4+8}$	<code>\lilyTimeSignature{3 + 4}{4 + 8}</code>	

Accidentals are available in starred and unstarred versions.
See table 5 for the list of implemented commands.

Table 5: Accidentals












♮	<code>\natural</code>	<code>accidentals.natural</code>
♯	<code>\sharp</code>	<code>accidentals.sharp</code>
♯↑	<code>\sharpArrowup</code>	<code>accidentals.sharp.arrowup</code>
♯↓	<code>\sharpArrowdown</code>	<code>accidentals.sharp.arrowdown</code>
♯↕	<code>\sharpArrowboth</code>	<code>accidentals.sharp.arrowboth</code>
♯/	<code>\sharpSlashslashStem</code>	<code>accidentals.sharp.slashslash.stem</code>
♯//	<code>\sharpSlashslashslashStemstem</code>	<code>accidentals.sharp.slashslashslash.stemstem</code>
♯/	<code>\sharpSlashslashslashStem</code>	<code>accidentals.sharp.slashslashslash.stem</code>
♯//	<code>\sharpSlashslashStemstemstem</code>	<code>accidentals.sharp.slashslash.stemstemstem</code>
×	<code>\doublesharp</code>	<code>accidentals.doublesharp</code>
♭	<code>\flat</code>	<code>accidentals.flat</code>
♭♭	<code>\flatflat</code>	<code>accidentals.flatflat</code>

5.7 Rests

Rests are usually available in starred and unstarred versions. See table 6 on the next page for the implemented commands.

For more information on how to use `\lilyPrintMoreDots` to produce multiply dotted rests please see section 2.5 on page 10.

Table 6: Rests

	<code>\wholeNoteRest</code>	Whole Note Rest
	<code>\wholeNoteRestDotted</code>	Dotted Whole Note Rest
	<code>\halfNoteRest</code>	Half Note Rest
	<code>\halfNoteRestDotted</code>	Dotted Half Note Rest
	<code>\halfNoteRestDotted*\lilyPrintMoreDots</code>	Example of Double Dotted Rest
	<code>\crotchetRest</code>	Crotchet Rest
	<code>\crotchetRestDotted</code>	Dotted Crotchet Rest
	<code>\quaverRest</code>	Quaver Rest
	<code>\quaverRestDotted</code>	Dotted Quaver Rest
	<code>\semiquaverRest</code>	Semiquaver Rest
	<code>\semiquaverRestDotted</code>	Dotted Semiquaver Rest

5.8 Dynamic Text

As explained earlier the Dynamic Letters can be accessed through `\lilyText` without providing glyph names or numbers as argument. For the available letters see 7. As a convenience there is a predefined command `\lilyDynamics`, which is just a wrapper around `\lilyText` that sets the `Scale` argument to a default value of 1.5.

Table 7: Single Dynamics Letters

<i>f</i>	<code>\lilyDynamics{f}</code>	forte
<i>p</i>	<code>\lilyDynamics{p}</code>	piano
<i>m</i>	<code>\lilyDynamics{m}</code>	mezzo-
<i>r</i>	<code>\lilyDynamics{r}</code>	rin-
<i>s</i>	<code>\lilyDynamics{s}</code>	s-
<i>z</i>	<code>\lilyDynamics{z}</code>	-z

These Letters can be combined to make complex Dynamics. *lily~~glyphs~~* doesn't provide a full set of predefined commands as they can easily be entered as single strings to `\lilyDynamics`, like `\lilyDynamics{sffzrmp}`, resulting in *sffzrmp*. In this specific situation you could enter a small horizontal space

between the \mathbf{z} and the \mathbf{r} – but as this combination wouldn’t occur in real life, we don’t need to demonstrate it here. There are a few predefined commands (see table 8) that are only used for the combination of letters that need some special “kerning” attention. These commands internally use `\lilyDynamics` with its default scaling. These predefined commands need starred and unstarred versions again, as \LaTeX would ignore spaces after the unstarred command.

Table 8: Combined Dynamics Expressions

\mathbf{rf}	<code>\lilyRF</code> , <code>\lilyRF*</code>	rinforzando
\mathbf{rfz}	<code>\lilyRFZ</code> , <code>\lilyRFZ*</code>	rinforzando (alternative)

5.9 Graphical Dynamic Symbols

Graphical dynamic symbols are realized by including image files and available as starred and unstarred versions. See table 9 for the implemented commands.

Table 9: Dynamics Signs


$\mathbf{<}$	<code>\crescHairpin</code>
$\mathbf{>}$	<code>\decrescHairpin</code>

5.10 Scripts

Script implementation has just begun. For the implemented glyphs table 10 on the next page.

You may find it strange that script glyphs sometimes seem to print too far to the left, clashing with the preceding text. This is due to the fact that in musical engraving scripts are centered relative to the note they belong to. Therefore you often have to add extra space before the glyph if you access them directly. The predefined commands should of course have this already built in.

Table 10: Scripts

	<code>\fermata</code>	Fermata
---	-----------------------	---------

A Appendix: Licenses

A.1 GNU General Public License

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

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

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as

changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License.

If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your

direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of

a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- (a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- (b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- (c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- (d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- (e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In

determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- (a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- (b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- (c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- (d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- (e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- (f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License

(for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public 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.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE

WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

A.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

`<http://fsf.org/>`

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

Preamble

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

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

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

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

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. (Thus, 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. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

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 Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document 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, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. 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, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include 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, PostScript or PDF 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.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states

that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

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 (or copies in media that commonly have printed covers) 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 computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. 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 fewer than five), unless they release you from this requirement.
- 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", Preserve 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. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements 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 to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

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, and that you preserve all their Warranty Disclaimers.

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 “Acknowledgements”, 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, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which 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 half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket 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, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is

the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

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. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.