

This document is part of **openLilyLib**<sup>1</sup>,  
a collection of resources for the LilyPond notation software<sup>2</sup>  
and the  $\text{\LaTeX}$  typesetting system.

Excerpt from:  
**The openLilyLib Tutorials**

# Plain Text Files in Music

Urs Liska

June 28, 2013

---

<sup>1</sup><http://www.openlilylib.org>

<sup>2</sup><http://www.lilypond.org>

Copyright © 2012–13 Urs Liska and others

---

openLilyLib is a collaborative and free software and documentation project. All contributions are copyright by their attributed authors.

If not stated otherwise all creative content distributed by the openLilyLib project is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit

<http://creativecommons.org/licenses/by-sa/3.0/>.

---

All software that is part of openLilyLib 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. To view a copy of this license, visit

<http://www.gnu.org/licenses/gpl.html>

# Contents

<b>1. Plain Text Format</b>	<b>6</b>
1.1. Transparency and Control . . . . .	6
1.2. Content, Meaning and Appearance . . . . .	7
1.3. Readability and Stability of Text and Binary Files . . . . .	9
1.4. Editor Independence . . . . .	10
1.5. Programmability . . . . .	10
1.6. Compiling Files vs. Instant Feedback . . . . .	11
<b>2. Version Control</b>	<b>14</b>
2.1. Basics of Version Control . . . . .	14
2.2. Collaborative Editing . . . . .	16
<b>3. LilyPond</b>	<b>18</b>
3.1. Frescobaldi . . . . .	18
3.2. Textual Representation of Music . . . . .	21
3.3. Variables and Includes . . . . .	23
3.4. Comments . . . . .	24
3.5. Conclusion . . . . .	24
<b>4. <math>\LaTeX</math></b>	<b>27</b>
4.1. Music Examples . . . . .	28
4.1.1. lilypond-book . . . . .	28
4.1.2. musicexamples . . . . .	28
4.1.3. OOOlilyPond . . . . .	29
4.2. Notational Elements . . . . .	29
4.3. Conclusion . . . . .	30
<b>5. Applications</b>	<b>31</b>
5.1. Preparing a Musical Edition . . . . .	31
5.2. Book preparation . . . . .	32
5.3. “Crowd Editing” . . . . .	32
5.4. Single-source Publishing . . . . .	33
<b>6. Where To Go From Here?</b>	<b>34</b>
<b>A. Appendix</b>	<b>36</b>
A.1. Comparison of File Formats . . . . .	36

## *Contents*

A.2. musicexamples Document . . . . .	45
---------------------------------------	----

## Abstract

Most musicians and musicologists are used to editing text documents and musical scores with graphical WYSIWYG tools<sup>3</sup>, and wouldn't even consider a different option at all. This paper discusses an approach to authoring musical documents that is based on editing *plain text files* instead. The described concepts, tools, and workflows have significantly changed my life as a document author, and I wholeheartedly endorse them because I strongly believe in their unique and substantial advantages.

The plain text approach is practically non-existent in the humanist disciplines or in the music business, while being de facto standard in many natural and computer sciences. Working with plain text based tools indeed requires a certain shift in mind-set for people who aren't already familiar with the corresponding working paradigms. And it can't be denied that the learning curve is considerable. But this investment is absolutely justified because on the long run it greatly benefits productivity—starting from superior result quality, extending through robust and reliable storage forms, up to workflow options unimaginable otherwise. Reading and writing music, playing an instrument, investigating a manuscript source—all this involves a very long and intense learning curve, and we mastered them as a matter of course in order to become the professionals we are. So don't be afraid!

This document aims to give you an overview of the power that text based approaches can give you. It is not meant as a detailed description or a guide to using the software mentioned—for that, please refer to the suggested reading at the end.

The software packages I will introduce you to on the following pages are:

- *Version Control Systems* – that keep your work under control
- *LilyPond* – the program that lets you engrave beautiful scores
- *LaTeX* – the professional typesetting engine for text documents

---

<sup>3</sup>What You See Is What You Get—such programs provide an editing environment where the document is displayed identically to the way it will be printed.

# 1. Plain Text Format

Who on earth would voluntarily enter music as a text file? Aren't these people just nerds who think that only what hurts can be good? Isn't it *natural* to edit musical scores within a graphical user interface?

Well, using graphical user interfaces to edit a document with a mouse or other pointing device is obviously quick, effective and easy. However, there are lots of reasons why it *is* a good idea to edit and store documents in text files. Text based work avoids several fundamental problems that other approaches share, and it opens up a whole range of advanced options that are only available in this context. And it is good to know that nowadays there are editors available that significantly streamline your work with text files.

Firstly, let us consider a number of features working with plain text files offers.

## 1.1. Transparency and Control

Have you ever wondered how your notation program internally manages the contents you entered, especially when you had the impression it maliciously made fun of you? One of the reasons I turned my back on graphical notation programs was my frustration with being completely at the mercy of the software with regard to my personal layout decisions. I can't recall (and actually don't want to) how often I ran through the loop of 1.) entering music, 2.) moving items around, 3.) applying some manual tweaks like flipping stems, breaking beams, suppressing or parenthesizing cautionary accidentals, 4.) hitting "Update Layout" and finally 5.) tearing my hair out because at least one third of my manual settings mysteriously vanished.

It's been a long while since I had used graphical software, so maybe things have improved a lot, but the fundamental fault is still valid: I am at the mercy of the software and can't see or control how it represents the content internally. A friend finally stopped using (a later version of) the same graphical program when he fixed one missing accidental in a score he had earlier tweaked to perfection—and this accidental caused the whole layout to break irreversibly. With a graphical program there is no notion of the manual changes against the default result, so the only resort is to "undo" and hope this will fix it ...

Besides this risk of losing work I also can't usually tell how the program interprets changes I've made. For example, if I add a description text somewhere near the violin staff, will it follow the music when the page layout changes or will it stay in the same position relative to the corner of the page?

If I edit a plain text file instead I'm completely in control over all these issues. If I tell the program to break a beam or to draw a slur above the note it is explicitly and reproducibly defined and readable by anybody. There is nothing hidden in "settings" dialogs or even unreadably buried somewhere in the file as the result of dragging something with the mouse. If I

manage to break something (which of course happens) I can review what I did, compare to the last working version<sup>1</sup> and understand and fix the issue.

There is a price for this, namely having to enter code by hand and learning how to do that, but it is very rewarding on the long run. And being in control just *feels* better ...

## 1.2. Content, Meaning and Appearance

Through the overwhelming presence of WYSIWYG applications we have got used to the impression that the *visual appearance* of a document is identical to its *content*. But this is actually a quite shortsighted assumption.

If you look at that text document:

### My new chapter

#### With a section

This is the **continuous** text with some **emphasized** words.

you will notice that there are several parts formatted in sans-serif bold face of different sizes. You may guess that the author has applied several style sheets to get this appearance, but to verify it you'd actually have to select the texts and look into the corresponding dialogs or toolbox elements. Only then you can see whether the author manually formatted the text, applied paragraph styles or character styles. He may even have applied character instead of paragraph styles to format the headings.

The corresponding  $\text{\LaTeX}$  source file would look like:

```
\documentclass{article}
\newcommand{\terminus}{\textbf{\textsf{\#1}}}
\begin{document}
  \chapter{My new chapter}
  \section{with a section}
  This is the \terminus{continuous} text with
  some \textbf{\textsf{emphasized}} words.
\end{document}
```

While this may look complicated on first sight it actually is much clearer because it distinctly shows you the content *and* the author's intentions. The `\documentclass` is something like a document template, but much more powerful. `\terminus` is defined as a character style applying bold ("`...bf`" for BoldFace) and sans-serif ("`...sf`" for Sans-Serif) to its content ("`\#1`"). Inside the document body (enclosed by the `\begin` and `\end` statements) you can see the *content* along with *semantic markup*. The two headings use explicit sectioning commands which are (for now) equivalent to paragraph styles. The two bold words in the text line are formatted differently: the first one is enclosed by our `\terminus{}` character style while the second one is manually formatted as bold and sans-serif. This mixture of formatting techniques is highly

<sup>1</sup>See the "version control" chapter 2 on page 14

## 1. Plain Text Format

disadvantaged and only used for the sake of the example. But you may notice that in the plain text file this “problem” is immediately obvious while the graphical program actually hides away such inconsistencies.

If we think about musical scores we have to come to a very similar conclusion. I began this chapter with the rhetorical pretention that it is “natural” to edit a score in a graphical way. But in fact that’s not entirely true because a score isn’t a graphical object like a painting but rather a graphical *representation* of the musical *content*.

If you consider the following minimal score and its (complete) LilyPond input file:



```
{  
  \time 3/4  
  \key es \major  
  \clef bass  
  es8 f8 g8 as8 f8 es8  
}
```



you can see that the source on the right side completely defines the *musical content* of the score: a time signature, a key definition, a clef, and a few notes with durations. Of course the page layout is also an inseparable part of the score document, but on a conceptual level it is completely separate from the musical content. Without going into more detail here you should note that in LilyPond's plain text file you have complete control over the separation of content, meaning and appearance. While this at first may seem somewhat abstract it is actually a very important point when it comes to preparing different output from one source.

### 1.3. Readability and Stability of Text and Binary Files

In the previous section you have seen examples of plain text files for a text and a score document. OK, I admit that it takes some time getting used to filtering out the markup commands in a text file mentally. But I can't emphasize enough that you actually can *read* that file *at all*.

Does this bold statement surprise you? Don't you think that you can immediately conceive the content of the graphical score, much faster and easier than the source file that you first have to read and interpret? Well, then you are again exchanging the real document file and the graphical representation the graphical programs expose to you.

Please take the time and have a look at section [A.1](#) on page [36](#). That's what the file contents from the previous examples actually look like. After skimming through the Word™, OpenOffice™, Finale™ and Sibelius™ files you should reconsider the “overly complex” *source code listings* of the previous section. Now you'll probably see why I emphasize that you can *read* `TEX` and LilyPond files, isn't it? You may have noticed that both Word™ and OpenOffice™ also have an alternative file format that is stored in “plain XML”. While a human actually *can* read these files the actual content is very deeply buried inside them and practically inaccessible to the human eye.

The fact that plain text files are human readable has two major implications:

**Recovery after file corruption** There are several ways that can make a file unusable, the most common being accidental deletion, a general disk failure or the crash of a program leading to an inconsistent state of the file (the latter fortunately becoming increasingly rare nowadays). Usually such files can be partially retrieved with special tools, but the picky thing is the “partially”. If you have a binary file that can only be read by its original program the whole file is now unusable, even if the corrupted portion is very small. From a corrupted plain text file on the other hand you will be able to retrieve everything *as content* that is still there at all. And with some luck you may even guess the missing pieces. Sometimes it will still be more efficient to start again from scratch, but depending on the nature of your project this characteristic of plain text files may actually be a life-saver.

**Restoring “ancient” files** As programs mature they modify their file formats. Usually programs can still open the files from older program versions, but that isn't an endless option. With the majority of programs you would need to have some older version around to be able to open “ancient” files. And with new operating systems chances become increasingly worse

to find one. Generally speaking, when software becomes unsupported its documents tend to become unusable.

This may also happen with programs that store their data in plain text files. But as the file format is usually much better documented it is more likely that you will find a conversion option still available. And if everything fails and you have a file but no dedicated editor you still have the plain text and free access to it.

### 1.4. Editor Independence

Plain text workflows separate the tasks of editing, processing and displaying documents. This way you are not restricted to *the one* application the (commercial) vendor imposes on you, but you are free to choose any editing environment you like, consider appropriate or just feel comfortable with. There are programs that try to integrate all parts of a workflow, just like the WYSIWYG tools do—but it is your free choice to use them or not.

You can even edit a project with different tools at a time. For example it might be handy to be able to edit a document with any text editor on your smartphone, or just going to a public internet access point and send yourself an email with a sketch—both of which you would then process to a pdf document when you are home again.

Another interesting application is the use of plain text as input to web forms. Many web forms, e. g. in forums, allow you to enter your posts in *Markdown*<sup>2</sup>, a minimalist plain text language, and convert them to formatted HTML. And recently *Wikipedia* added support for LilyPond code that will be rendered as music examples<sup>3</sup>.

### 1.5. Programmability

Plain text files can be edited not only by any text editor but by *anything* that can edit text files—which makes it accessible to about any programming language you may think of. You are not restricted to any scripting interface the vendor graciously provides you, but you can apply any imaginable operations to your source files: analyze them, process them, or even computationally generate them from scratch. There are many possible applications, from applying contrapuntal operations or algorithmic composition up to managing music examples in a database.

If you have never programmed and don't intend to learn it you may think that's a rather academic feature. But this isn't true at all. For example I'm currently working on the infrastructure for a large orchestral score where those who enter the music are only exposed to small segments of individual instrumental parts. The full score (which is initially filled with rests) will automatically build itself and use whatever segments have already been entered. This is an example of an approach where the administrators of a project *explicitly* use programming to allow sophisticated logic in the project while at the same time *hiding the complexity* from the regular contributors.

---

<sup>2</sup><http://daringfireball.net/projects/markdown/>

<sup>3</sup><http://en.wikipedia.org/wiki/Help:Score>

## 1.6. Compiling Files vs. Instant Feedback

One thing the average WYSIWYG user needs to get used to is *not* to get instant feedback. Graphical programs immediately display any updates in their graphical user interface, while the text files first have to be *compiled* before you can see the results of your modification.

While this may seem cumbersome, it actually isn't a problem but an inherent advantage, and a big one! Other than a graphical program the compiling program can first read the input file, get a solid internal representation of the content and structure, and finally take its time to produce the best possible layout for the given content. It can judge the implication of a modification for the whole document and rethink the layout completely. The problems graphical programs are encountering are immediately visible when you (for example) try to move a figure in a 250 pages text document containing 80 figures, or when you insert some music in the middle of a longer score—tasks that don't upset text based programs at all.

As a logical result from this different approach text based programs (I'm speaking of LilyPond and  $\LaTeX$  in particular) produce *default output* of significantly superior quality than their graphical counterparts. While it is obviously possible to produce publication quality output with Finale™ or Sibelius™ (but not with Word™ or OpenOffice™) the default output of LilyPond is practically always *readable* and *usable* without any further intervention. As a rule of thumb I'd say that you can use LilyPond scores (and  $\LaTeX$  documents) without bothering about layout details as long as you don't want to publish them. This is a significant improvement for the development process of documents and editions as you can concentrate on the real *content* up to the final stage of print preparation.

Look at the following example of a complex score<sup>4</sup> and note that there isn't a single manual intervention. All I did to help LilyPond do its job was assigning the musical lines to the right voice numbers (1–4) so it knew whether to place stems up- or downwards and how to move items to avoid clashes.

The image shows a musical score excerpt for Arnold Schönberg's "Gethsemane" (fragment). It consists of two staves: a vocal staff labeled "Bar." (Baritone) and an orchestral staff labeled "Orch." (Orchestra). The vocal staff is in bass clef with a key signature of three flats (B-flat, E-flat, A-flat) and a common time signature (C). It contains three measures of music, with lyrics "Geist der Wahr - heit, mei-nen Zwie - spalt," written below the notes. The measures are numbered 66, 67, and 68. The orchestral staff is in treble and bass clefs with the same key signature and time signature. It contains three measures of music, with measure numbers 66, 67, and 68. The notation is complex, featuring many accidentals and ties.

<sup>4</sup>Arnold Schönberg: Excerpt from "Gethsemane" (fragment). © Belmont Music Publishers, permission granted for use in this tutorial. You may use and redistribute the example unchanged provided you keep this copyright notice in place.

## 1. Plain Text Format

A musical score excerpt from Arnold Schönberg's 'Gethsemane'. It features a vocal line in the upper staff and a piano accompaniment in the lower staves. The vocal line includes the lyrics 'mei - ne dunk - le Schuld.' across measures 69, 70, 71, and 72. The piano accompaniment is complex, with many accidentals and a dense texture.

Music Example 1.1: Arnold Schönberg: Excerpt from “Gethsemane”

It is really amazing how LilyPond manages to do most things immediately right. While the score isn't perfect to publication quality there is *nothing* that hampers reading the music when placed on the music stand.

Now have a look at the following example of another complex score<sup>5</sup> that LilyPond did *not* manage to engrave perfectly without assistance:

A musical score excerpt from Arnold Schönberg's 'Nie ward ich, Herrin, müd' op. 8, 4. It shows a piano accompaniment with complex textures, including triplets and slurs. The notation is dense and includes many accidentals. The engraving shows some issues, such as overlapping notes and slurs that are not perfectly aligned.

Music Example 1.2: Arnold Schönberg: Excerpt from “Nie ward ich, Herrin, müd” op. 8, 4

Please note that there isn't any single manual tweak to that engraving either. I just entered the music as in the previous example. Of course this is far from perfect, and the LilyPond developers would surely take this example as a source of inspiration for improvements of the engraving engine. But if you excuse the long phrasing slur from m. 1 throughout 3 (which I judge no other automated engraving system would be able to get right by itself either) you will notice that the engraving is completely readable—you could put it on the music stand and play from it right away (which I can confirm from practical experience ...).

If you compare that to the attempt one of the two “big players” took at the same measures (with the same task description: entering music, assign voices correctly, no further manual tweaks), you will understand what I mean ...

<sup>5</sup>Arnold Schönberg: Excerpt from: “Nie ward ich, Herrin, müd” | Nr. 4 aus “6 Orchesterlieder” | für Gesang und Orchester | op. 8/1, © 1911 by Universal Edition A. G., Wien/UE 3041, [www.universaledition.com](http://www.universaledition.com) [permission granted for use in this tutorial. You may redistribute the example provided you keep this copyright notice and the hyperlink in place]

## 1. Plain Text Format



There are a few more features about working with text files that I'll write about in the LilyPond chapter 3 on page 18, namely the potentials of using *variables*, of being able to *include* files and design sophisticated cascading set-ups, and finally of *commenting* and *documenting* source files. But first let's take the time to learn about what is maybe the single most important feature of plain text files: their accessibility to *version control*.

## 2. Version Control

*Version control* is a concept I had heard about long ago, and I assumed it was one of the tools belonging to software development in general. But realizing that it can very well be used by musicians and musicologists was a revelation that made me completely revise my working methods last year. It actually *changed my life* as a document author, and I can't imagine how I could ever have existed without ...

As a first approximation you may understand version control as an infinitely flexible implementation of *undo/redo* mechanisms. Versioning stores the complete development history of your document or project and lets you investigate *any* state it has ever been in over time. Furthermore it lets you roll back (or reapply) *any individual* change you have applied at any time, not necessarily in (reverse) chronological but in arbitrary order.

Suppose two weeks ago you reworked a chapter of your book and now you realize that someone else has just now published exactly the same insights—you need to remove these changes and try to get the chapter into the state it had been before. With a traditional toolchain you would be lucky to have a version of that state at hand, either as a backup file or printed on paper. With the backup file you may roll back your work to that state—but of course you lose all work done afterwards. Or you could put the (digital or printed) backup and the current version side by side and manually emend the changes you applied. Both solutions are cumbersome, error prone, and they rely on the—random—existence of a backup of exactly the desired state.

With a document under version control on the other hand you may identify the exact change-set that you want to undo and just revert these changes—and leave everything done before or after the way it is. It may not be obvious to you, but this can easily span decades, programs', and even operating systems' lifecycles—because it works on plain text files as seen in the previous chapter. You think that's cool? I can tell you it *is*! And this is just the surface – we'll later deal with other, maybe even more important aspects of versioning.

### 2.1. Basics of Version Control

The fundament all this is built upon is the line-by-line comparison of the complete project directory that the versioning program performs when you “save” your project<sup>1</sup>. Of course these “lines” that are compared refer to the lines in your plain text files. In a text document a line will usually contain one coherent entity like a sentence, a list item or the like. In a score this might be (depending on the complexity of the material) one voice in one measure (but you can lay out your text files as you want.) So this means that whenever you save your project the versioning software records (e. g.) which sentences or which measures you have modified, stores this as a *changeset*, and adds this to the project history. And you can completely decide

---

<sup>1</sup>The actual terminology is different but I'll try to use familiar terms as long as possible.

## 2. Version Control

for yourself how fine-grained you tailor such changesets: They may envelope the work of a day, the complete revision of a chapter or a single corrected note. But note that the first example is actually a bad one because the idea of versioning is to organize the history of a project as a series of *coherent* sets of changes. Probably you'd rather split your day's work into snapshots like 1.) "proof-reading choir parts" 2.) "adding critical notes" 3.) "reorganizing directory structure". Note that this concept does work on the project directory and not on single files. So a changeset can span a whole set of files, thus linking related work together.

Once you have saved a changeset to the project it is part of its history, and it is practically impossible to lose that work again by accident. To "achieve" this you would have to apply some really advanced tools you normally never touch. You can think of a project under version control like a bag where you put all your stuff in, while the versioning software keeps track of all changes. Whenever you want to investigate any earlier state of your project the versioning system uses the content of the bag and its own records to bring your project directory into the form it had at that specific state. From there you can revert, modify or reapply any changeset that doesn't create structural conflicts<sup>2</sup>.

But you could also create a new *branch* from that state. *Branching* is one more important concept of source code versioning. You can see a branch as a "session" that is independent from other lines of work. To use the given example (of inspecting an earlier point in project history) you might go back to a certain point, create a new branch and work in a different direction. If you are satisfied with this new approach you can simply replace your earlier work with the new one (while not completely losing it because it's kept in the project history)—or you can decide to throw away the new attempt and switch back to the original version of your document. A more common application of branches is to start any new work that might get your document/project into an inconsistent state in the context of a branch. This is merged with the main line of work only later when it is finished. An example for this would be integrating the results of a critical revision into a score—this way the main line would keep its consistent original state up to the moment when you merge in the revision in one single step.

I have omitted mentioning any concrete software that does this ominous versioning. This is because there are numerous tools in this area, but they more or less share the described concepts. In their generic form they all are command line programs that you can tell to perform actions on your project like saving a new changeset, managing branches or inspecting dedicated states of history. For all versioning systems there exist graphical front-ends for all major operating systems that aim to ease working with the systems. But one has to admit that versioning systems are *very* powerful tools that consequently deserve some attention getting used to and some care in working with them. You *may* have heard the names *CVS*, *Subversion*, *Git* or *Mercurial*, but if you don't it's no problem. They share some concepts and differ in others—but that is beyond the scope of this article and actually completely unimportant for its purpose.

---

<sup>2</sup>Such a conflict would for example arise if you would try to revert changes to some lines in a file that has in the meantime been deleted.



### 2.2. Collaborative Editing

While all this is already very exciting we still haven't touched the most important aspect of all: the impact versioning can have on the concepts of *collaborative editing*. The ambiguity of the expression is intended—I'm talking about editing (or more generally working on/with) musical works in a collaborative and social way, and I'm talking of simultaneously editing source files. To emphasize the potentials I will take a step back and first describe traditional ways of collaboration ("traditional" being quite relative, of course).

The most basic way to collaborate on a project is to exchange ideas personally, by phone or by email, and have one person edit the document(s). This way the document is guaranteed to be in a consistent state, but the workflow isn't really practical if you don't happen to share an apartment or at least an office.

The next (and quite common) way is to exchange documents by email or on data storage media like disks or USB devices. You will surely have already worked that way or do it on a regular basis. While this *may* work in simple cases (for example two collaborators and only a very small number of files) it adds a level of complexity that can easily lead to problems:

1. What do I do with a document returned by a partner? Do I replace my original version or do I make a copy (presumably named something like "essay-revision-Michael-2009-12-04" or "essay-v003")? What if there are more than two people and/or numerous iterations involved?
2. How do I prevent the situation that a document is simultaneously modified by different people? This can easily happen if my partner sends a document a second time with further modifications while I have already made changes to it. If we're lucky and notice the conflict at all the only resort is probably to compare the files manually!
3. Working with such a system is only possible if you are dealing with a quite small number of files. Currently I'm working on an edition project with a project directory containing more than 500 files. It would definitely be impossible to even try to keep track of modifications of this number of files via email.

The first and the last problem can be dealt with by storing the files at some shared location—in a local network or at one of the numerous providers of *cloud based* storage. This way you don't have to manage different copies of your documents, which improves the situation a lot.

The second problem however—preventing simultaneous modifications—can't be solved reliably with this approach. To avoid conflicts a team has to obey to strict policies that aren't always easy to enforce. People have to "lock" files they want to edit, which is quite difficult if they aren't permanently connected to the network. While not theoretically impossible such an approach bears an enormous inherent risk, especially with larger numbers of collaborators—and sooner or later there *will* be accidents ...

This is where *version control* comes into play, and this is where we *really* enter new grounds in our scholarly or artistic workflows. Using plain text files and version control enables us to benefit from techniques and strategies that have been invented for and improved for decades in software development<sup>3</sup>. And the key concept in this regard is *collaboration*. Software is mostly

<sup>3</sup>In fact Git has been originally written to manage the development of the Linux kernel.



## 2. Version Control

developed by teams or even large communities, and version control systems have been originally designed to seamlessly manage contributions by a large number of people. For example the Linux kernel project accounts for several hundreds of thousand “commits” by more than 10.000 contributors.

The heart of a versioned workflow is a shared data inventory, or *repository*, located on a server that is accessible by all project members<sup>4</sup>. With *distributed* version control (which is what I’m talking about exclusively) each contributor has a local copy of the repository on which he can work without having contact to the shared server. There he adds his snapshots to the project history (as described earlier), and whenever it is appropriate or practical he can synchronize his work with the data on the server, downloading the new work from others, integrating this with his own work and finally uploading the local changes to the shared repository.

Earlier I wrote that version control is based on line-by-line comparison of files. This is of even greater importance in distributed collaborative workflows, as it effectively renders file locking unnecessary. I would like to stress that it is for example even possible that one contributor adds articulations to one hand of a piano part while another one at the same time fixes engraving shortcomings in the other hand of the same measure without any problem. As long as the two people won’t modify *the same line* in the same source file, all of this synchronization will work silently without anybody really noticing it. This will even work when person A modifies some content of a file `foo.txt` while person B *at the same time* renames that file!

Of course the point of this isn’t to make conflicts *unlikely*—this would actually be quite negligent. The advantage is that potential conflicts can be reduced to the conflicting line(s) in the source file. If simultaneous changes can’t be reconciled automatically there will be markers in the text file showing both conflicting versions. From there one can decide individually which version to keep (or to mix them) and submit the result to the versioning system. So versioning can’t completely prevent conflicts to happen but it makes sure they aren’t fatal.

Another feature that has become crucial to my work relies on the way versioning software records changesets. As described earlier they keep track of file differences, and they can be inspected later. This way I can see exactly what others have done to a file without bothering to compare the files manually or visually<sup>5</sup>. Office programs have a feature to record changes (whatever it may be called in a given program and language). But once I have accepted or rejected the changes (and of course I can only use the document when I have done so) this (visual) indicator vanishes. Changesets in a versioned project on the other hand are part of the project history and therefore “eternally” present.

To sum up: in a project with version control each contributor has access to and can work simultaneously on the whole data inventory, with very little risk of conflicts and a straightforward way to resolve conflicts that may eventually arise nevertheless. This opens up quite a lot of perspectives on collaborative work that I consider very promising. In chapter 5 on page 31 I will describe some of them.

---

<sup>4</sup>This could be a server on the local network or—much more common—a dedicated service provider such as GitHub (<https://github.com>) or Bitbucket (<https://www.bitbucket.org>)

<sup>5</sup>In fact I can even inspect others’ changes on the go, as there exist smartphone apps for several hosting providers

## 3. LilyPond

When it comes to engraving musical scores my tool of choice is *GNU LilyPond*<sup>1</sup>. Not only do I love its beautiful score output, but I really value how seamlessly and professionally I can integrate scores with (L<sup>A</sup>T<sub>E</sub>X) text documents, and I have come to absolutely rely on the workflows and concepts based on the plain text approach that it allows.

Of course I would love to give you a thorough introduction to this amazing software but that would be quite pretentious and especially out of scope of this article. Therefore I will keep my focus on the potentials of the text based approach, giving you just enough information to get an impression on how it would “feel” to work with it and to get the necessary foundation for the following ideas.

As you know quite well by now LilyPond works by reading in text files and processing them to engraved scores. So one effectively *has* to learn how music is represented in LilyPond’s input language. But it may be good to know that while it is actually possible to edit the text files with any plain text editor there actually *are* graphical programs assisting you heavily in dealing with the text files. Although there are a few I will only show you one.

### 3.1. Frescobaldi

*Frescobaldi*<sup>2</sup> is a complete editing environment that allows you to do everything you need within one interface. In that it is quite similar to the graphical programs I have earlier dismissed as conceptually flawed. But there is a big difference, as Frescobaldi presumably offers the best of both worlds: it gives you the greatest possible graphical access to the editing process while in its core still being a—sophisticated—text editor.

Its main window presents itself like many programs that allow to edit textual source files of graphical documents: it is split in a *text editor* and a *music view* (see figure 3.1 on the following page). In the music view on the right hand you see (surprise) the music. Please note that a) Frescobaldi isn’t responsible for generating the engraving in this view and b) this view doesn’t have a “live” connection to the source but is only updated when LilyPond is run. So you could actually call Frescobaldi a WYSIWYG editor because what you see actually *is* what you finally get—but you can’t (yet) edit the score in a graphical way in this music view. On the left side of the screen shot you can see the *Quick Insert* widget, one of the tools that simplify your life by “remembering” how to insert all sorts of graphical symbols for you—but again: they won’t insert them “into the score” but as text into your input file.

Another example of the useful tools is the *Score Setup Wizard* that allows you to graphically configure a score of arbitrary complexity (just like in other programs) and that generates the

---

<sup>1</sup><http://www.lilypond.org>

<sup>2</sup><http://www.frescobaldi.org>

### 3. LilyPond

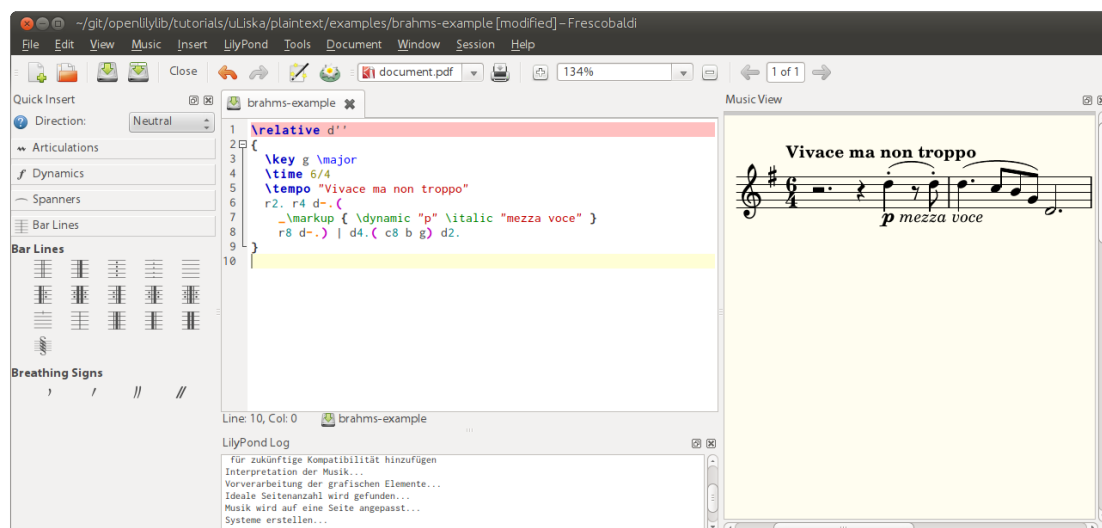


Figure 3.1.: Frescobaldi’s Main Window

text used for the corresponding score for you (see figure 3.2 on the next page).

If you look at the text editor again (in figure 3.1) you will see that the text is colored a lot (I will talk about the actual text soon). This is a feature that Frescobaldi shares with most programming editors: it “knows” LilyPond’s language and highlights the source text according to its structure. Once one has got used to that practice it greatly eases reading and writing the necessary text files.

When you start typing a command Frescobaldi can also make suggestions on how to complete them (figure 3.3 on the next page)—also a valuable assistance, not so much because of its time-saving but because it assists your memory and efficiently reduces errors.

But maybe the most important feature for daily work is that you can click on an element in the music view and will be taken to the corresponding place in the input file. In figure 3.4 on page 21 you can see the slur highlighted in purple in the music view and the highlighted text in line 8 in the text editor. This also works in the other direction: If you select text in the editor the corresponding notation elements are highlighted in the score. This lets you easily navigate your text file(s) and removes a handicap one could have seen in editing plain text files in the past.

One focus of current Frescobaldi development is the enhancement of the music view’s functionality. So it will soon be possible to partially “edit” the score with the mouse—of course in fact you will be able to tweak something visually and Frescobaldi will insert the appropriate text in the source file, possibly giving you the option to select from different approaches. Currently there are plans to implement (e. g.) correcting pitches, drawing and shaping curves, and annotating objects (see also section 3.4 on page 24). This will make “working with text files” even more comfortable.

Of course these remarks were only a very cursory overview of Frescobaldi. A thorough introduction can’t be the intention of this paper, but I wanted to give you a feeling to what extent current and smart editors can smoothen your editing experience. Now you are ready to

### 3. LilyPond

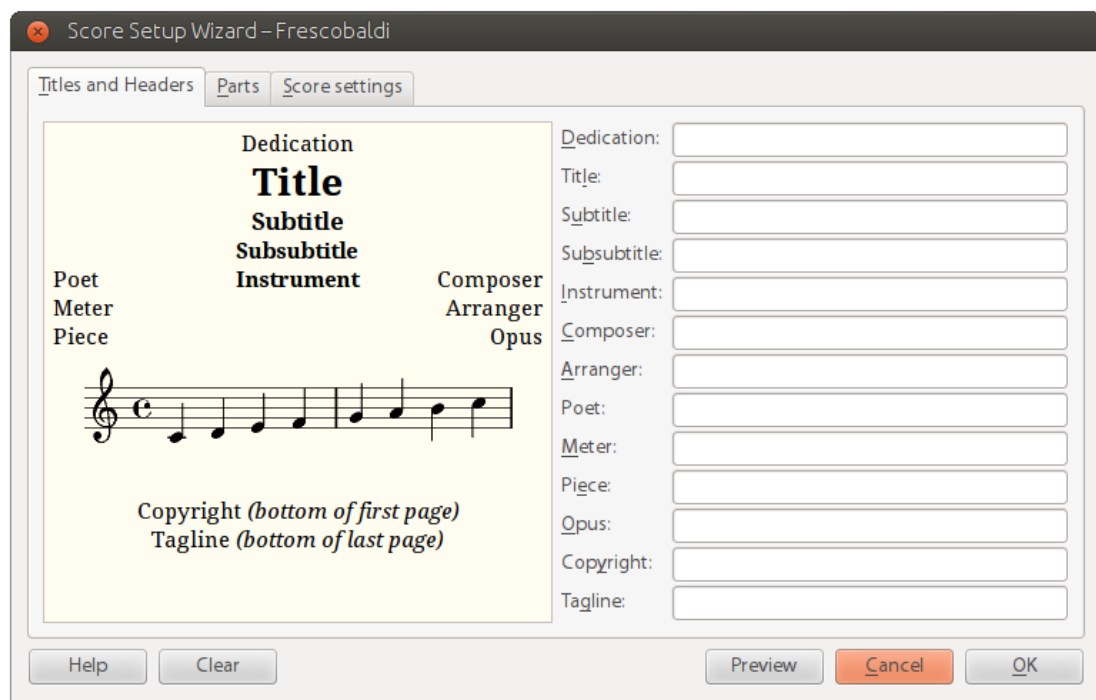


Figure 3.2.: Frescobaldi: Score Setup Wizard

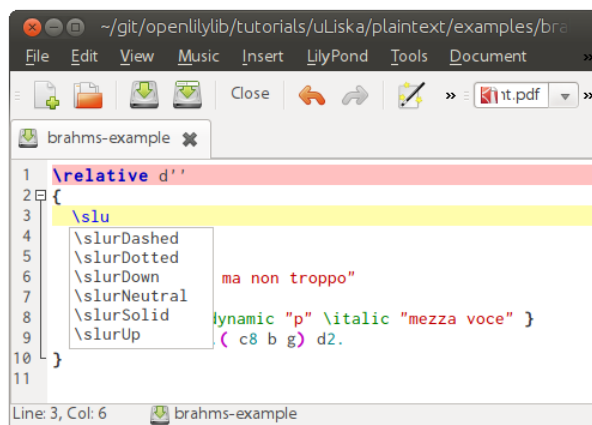


Figure 3.3.: Frescobaldi: Suggestions for available commands

### 3. LilyPond

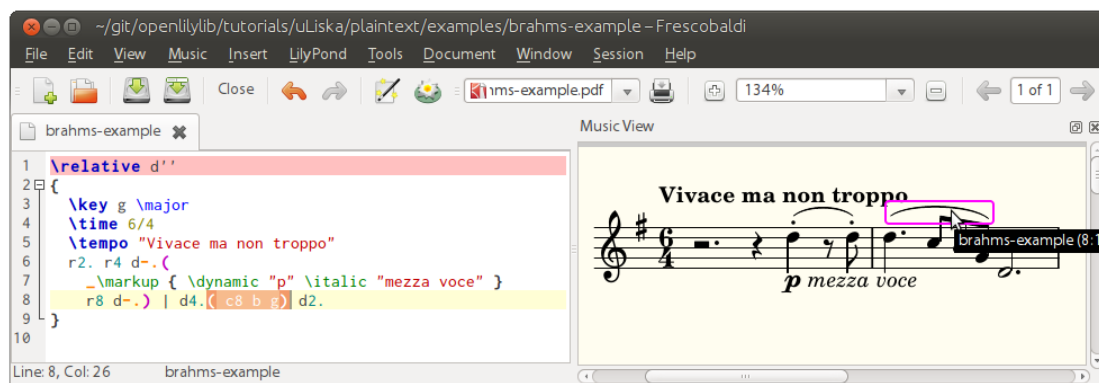


Figure 3.4.: Frescobaldi: Two-way link between input file and score

learn something about how LilyPond input files actually work.

## 3.2. Textual Representation of Music

In figure 3.1 on page 19 (and figure 3.4) you have seen a short score excerpt from Brahms’ first violin sonata, along with the LilyPond code that produced it. Now I’ll use that example to show you a little bit about the basic elements in a LilyPond score.

At the most basic level notes are represented by their pitch and duration, as you can see in this “bare” example:

```
{
  r2. r4 d r8 d
  d4. c8 b g d2.
}
```



The text writes rests (r) and pitches (d etc.) as well as durations (8, 4, 4., 2.), and everything is enclosed in a pair of curly braces. This is LilyPond’s notion of a *music expression*. A score is represented by one big music expression which can be constructed of (or split into) arbitrarily nested or chained smaller music expressions. There are a few things “missing” in the source that LilyPond adds by itself, namely a staff, a clef, and beams.

Two more elements aren’t actually figured out automatically, but hidden from the example. In fact you have to write down `\key g \major` and `\time 6/4` to tell LilyPond about the key and time signatures. But the beaming in the second measure is done automatically according to the time signature. Many things can be defined by such *commands*.

Elements like articulations or dynamics are attached to notes or rests by printing them after the reference item, like in this fake example:

```
{
  r2. r4 d-. r8 d-.
  d4. \sf c8\> b g\! d2.->
}
```



### 3. LilyPond

We have a few articulations that are attached with a hyphen, the staccato dot written as a dot, the accent as a >, all characters have been chosen to be as intuitive as possible. Dynamics are attached by a backslash, in the example we have the *sf* with \sf, the > with \>, and (as the only unusual item) the \! which represents the *end* of the hairpin.

We can attach *spanners* to pairs of notes, e.g. the slurs in the following example. Slurs are written as opening and closing round brackets (*after* the note they are referring to) There you can see that you can attach multiple items to a note by just writing them one after another

```
{
  r2. r4 d-.( r8 d-.)
  d4.( c8[ b] g) d2.
}
```



In addition to the slurs you can also see an example of manually setting the beams against the rules for the time signature, which is indicated by the use of square brackets (*after* the notes they are referring to).

In the last example of this *very* short introduction to LilyPond files we'll add the necessary text elements to the score. The *tempo* indication is entered by the \tempo command (the non-verbal part is here as an example only), while the *p mezza voce* is a \markup—a text that is attached to a note like any articulation or dynamic sign.

```
{
  \tempo "Vivace ma non troppo" 2. = 60
  r2. r4 d-.( -\markup
    { \dynamic "p" \italic "mezza voce" }
    r8 d-.)
  d4.( c8 b g) d2.
}
```



One thing you may have noticed in this last example is that I have broken the first line of music into three. LilyPond doesn't care about this *white space* and allows you to lay out your text file as you find appropriate—which usually will mean to keep coherent entities on a single line so that versioning will work optimally. In this case the entity is the *markup*, also an expression enclosed in curly braces. You could just throw some text in there, but in this score we format the elements, one using the dynamics font, the other as italic text.

Now you've seen your first real-world example of a LilyPond score. Of course the complexity of these text files increases when the music gets longer or more complex. But when looking at an input file you should always keep in mind that it is built from very small and rather simple units. And you should feel comfortable with knowing that there are editors at hand that really help you navigating and editing them.

### 3.3. Variables and Includes

In the previous section I wrote that LilyPond adds necessary elements to what you have entered. In the case of a file just consisting of a music expression as in our example it will actually add the necessary context for a staff, the score, and a “book” which represents the output file. This works only for simple scores such as music examples or maybe plain chants, but for most real scores you will have to define the score structure explicitly. While one would perhaps start with writing the music expressions directly inside the score set-up, there is the concept of *variables* that proves infinitely useful for any project of some dimensions.

The basic structure of a score looks like this:

```
\score {
  \new Staff { ... music expression ... }
  \layout{}
  \midi{}
}
```

Inside the score block there is one *Staff* that contains a music expression (I could have pasted the code from the previous examples here). `\layout{}` tells LilyPond to create a score, the optional `\midi{}` block additionally creates a MIDI file. Instead of just `\new Staff` there would be the complete structure of all (potentially nested) staves of the score, so the file would soon become very complex and confusing. To avoid this one wraps up the music definition in a variable (e.g. `violin = { ... music ... }`) and simply writes `\new Staff \violin` in the score block.

Reading this you might consider it just a detail of the input file structure and wonder how this relates to you. But packaging musical expressions in variables isn’t only useful to reduce the complexity of the score block but serves many more goals. It is a way for transparently and robustly store, use *and reuse* blocks of music. Reusing music is very useful, not only for the—admittedly quite specific—case of musical snippets or patterns, but also for the fundamental handling of the needs of musical engraving. Earlier I wrote about the separation of content and appearance, here we are dealing with separation of the *definition* and *use* of music. Music defined in a variable is used by a score—and it can equally be used by another score. Different scores accessing shared music may be a full score and the instrumental parts or different transpositions of the same song. In both cases the music is defined only once, and any update to that definition is propagated to all scores automatically. This makes the handling of scores and parts very robust in LilyPond.

Being able to reuse variables would only be a partial advantage if one couldn’t separate the definitions even one step further, into separate files, and *include* them. This way one can create a collection of files, a *library*, and keep them available for further use. This is not only useful for reusing work in future projects but also within one project. You can organize style and layout templates (i. e. definitions that influence the page layout and the appearance of the music) in a hierarchical and sophisticated way and have all files share some common styles (like font selections) but have different additional styles for different output formats (full score, part, music example ...). Any change in one of the style files is then automatically propagated to

### 3. LilyPond

all scores. Or you can make use of a technique called *commenting out*—place two alternative includes but comment one of them out, like this:

```
\include "draft-layout.ily"  
%\include "publish-layout.ily"
```

The second line of this is a *comment*, so only the draft-layout file will be included (and may color editorial additions red, print additional information about the date or whatever you like to define) as long as you are working on the score. When you're finished and prepare the final printout you can just move the percent sign one line up and get your final publication layout. And if you have this construction in a shared style file this switch will have global affect to all scores that are part of your project.

#### 3.4. Comments

The last example of the previous section led us smoothly to the next aspect: *comments*. LilyPond—as virtually all plain text formats—allows you to mark parts of a file as comments, i. e. text that will be ignored by the typesetting. This can for example be used to comment on technical issues you are facing. You could for example explain why you have decided to implement a certain voice-leading to solve a typographic problem, or leave a mark that some issues will have to be resolved. Combine this with the fact that all your manual interventions are written explicitly in the file and compare that with the situation of a graphical score editing program where you can hardly even tell what your manual interventions were ...

Currently there are plans to greatly enhance this functionality, especially with preparing scholarly editions in mind. It will be possible to enter specific comments in the input file—*i. e. directly beside the music they are referring to*—and a tool that will be called `lilypond-doc` will output them to a variety of formats. This will provide nicely formatted lists of 1. TODO items, 2. musicological questions, 3. typographical questions, and finally 4. critical remarks. The items in the list will be smartly linked, so that clicking on them will point you directly to the corresponding place in the source file. *Frescobaldi* will probably be able to allow you to insert such comments directly from within the music view, so you can click on a note and enter the comment. This way you will be able to do all your work *within the actual score* (and collectively with your collaborators), and finally you will get the entries for the critical report nicely sorted and formatted as source text to be used in the corresponding  $\text{\LaTeX}$  document!

#### 3.5. Conclusion

By now it surely won't make you wonder that I wholeheartedly endorse the use of LilyPond as a notation software. I can't recommend highly enough to give it at least a try—but you should give it a fair try and expect some time and effort needed initially. From my experience (having once been a beginner myself and from seeing other beginners) I may give you some advice on *dos* and *don'ts*:

Do *not* jump right in and try to accomplish a complex project that has to be finished in time, or you will surely run into frustrations. Ideally you should first read and follow the „Learning



### 3. LilyPond

Manual<sup>3</sup> from LilyPond’s excellent documentation which will introduce you smoothly to the basic concepts. But you may also tackle a task that is „real“ and on your desktop already. This will make it more interesting, but be sure that it isn’t too complex and that you won’t have to struggle with deadlines.

Be sure *not* to try it completely on your own, alone with you and the documentation. The ideal way would be to have someone mentoring you, giving you the right ideas and giving the right answers to the questions that really matter at a time. Of course this isn’t available for most of the beginners, so be sure to sign up to the `lilypond-user` mailing list<sup>4</sup>. Reading the discussion there (and skipping what seems too complicated for now) will be helpful to get a feeling for it, and the community is usually very responsive and helpful, as long as you show interest and own efforts.

Be sure to get used to using the documentation. As mentioned it is excellent, but apart from the Learning Manual it is a *reference* that mostly won’t guide you through the concepts. You will have to read things more than once, and for the starting time you will need to lookup things quite often. Even later you will need to have good access to that reference. So you should make yourself acquainted with the layout of the documentation as soon as possible.

Admittedly it takes some time and effort to get productive with LilyPond. But the benefits clearly outweigh that in my opinion. First of all you instantly get beautiful output that is pleasing the eye. LilyPond’s very existence is due to the frustration two musicians had with the anemic sheet music they were confronted day to day on their orchestral music stands. Therefore LilyPond’s first ideal and models are traditional, hand-engraved scores, and you (and your readers) will benefit from that saturated page impression out-of-the-box and in each score. Practically this means that you will have *significantly less work* to tune a score to be usable as performance material than when creating it with the known graphical programs (or the other way round: If you don’t do that work at all (as obviously many composers do) the resulting material will be vastly superior). If you want to prepare scores for publication you will probably have to spend time with LilyPond too, applying a smaller number of corrections which will each take more time than with the graphical programs. But as explained these corrections are much more robust, can be tracked and modified with a clean interface.

The other aspect of output quality is at least as important: Since you get *usable* results by default you can concentrate on the content much longer and only care about engraving details when actually preparing the final printout for publication. But *if* you want to apply manual modifications while still working on the content you have them in a clean and traceable way, and they won’t step in the way when the final beautification takes place.

But what really sets working with LilyPond apart is the potential of new workflows powered by the plain text approach, as I have described in the previous chapters. Seamlessly integrating work with others through versioning, documenting your work *in situ*, setting up sophisticated and cascading style sheets on „house“, project or file level are the keywords to be mentioned here. Interfacing scores or examples with  $\LaTeX$  documents will be one more topic, to be discussed in the next chapter.

It wouldn’t be honest to conceal one issue that I haven’t touched so far: LilyPond’s behaviour

---

<sup>3</sup><http://www.lilypond.org/doc/v2.16/Documentation/learning/index.html>

<sup>4</sup><https://lists.gnu.org/mailman/listinfo/lilypond-user>

### 3. LilyPond

towards exchanging documents with other software. Currently using LilyPond is kind of a one-way street with the only exit being the score in one of the various graphical output formats. There are several ways to export/convert documents from other programs to be engraved by LilyPond (although you should expect some manual work to be done), but you can't simply export LilyPond input files to formats that can be used by other programs. This usually isn't an issue when you are just creating scores to be printed, such as performance or teaching material, or music examples for books. But it can be a show-stopper if you are preparing editions for commercial publishing houses because the majority of them has their established workflows and insists on getting the scores delivered as Finale™ or Sibelius™ files. As you surely can imagine by now I wouldn't ever want to miss the plain text approach in preparing editions, so I would prefer doing it that way even if I knew (and felt sorry about it) that the score would end up being printed by one of the two programs. So at least for this application it is a big desideratum to get LilyPond to export to MusicXML, a common interchange format. As it stands, it doesn't even seem to be a major rewrite of the program but rather a few months of programmer's work. Maybe *one* rather large project with a sufficiently large organisation capable of bringing up some money would be enough to fix the situation. But unfortunately there can't be any promises when (and if at all) this hole will be closed.

## 4. L<sup>A</sup>T<sub>E</sub>X

L<sup>A</sup>T<sub>E</sub>X is for typesetting text documents what LilyPond is for typesetting scores. It provides typesetting as opposed to mere text processing, providing the usual document author—who doesn't know too much about the rules of typography—with the necessary expertise to produce professionally typeset documents. And it works by compiling plain text input files, so everything I have written about the advantages of using plain text files applies to L<sup>A</sup>T<sub>E</sub>X equally. Therefore I won't get into details introducing you to the concrete syntax of its input files, but try to introduce you concisely to some of the specifically musical features.

Your interface to L<sup>A</sup>T<sub>E</sub>X is the same as to LilyPond: The input files can be edited with *any* editor, and the compilers are command line programs. But there are (quite numerous) dedicated editors or IDEs<sup>1</sup> that assist you in editing the documents and organizing the compilation process for you. Earlier in section 1.2 on page 7 I showed you a basic L<sup>A</sup>T<sub>E</sub>X document in which you saw the three fundamental elements: the *document class*, *commands* and *environments* (although I didn't explicitly name the latter). As mentioned the document class is somewhat comparable to a document template in an office application. But it is much more powerful in that it can contain many layout elements, commands and more functionality. The L<sup>A</sup>T<sub>E</sub>X distribution contains many document classes for many purposes (from standard articles over beamer presentations to classes specific to fields of research), but you can also download classes from other sources, and you can (and will) write your own. One important aspect of document classes is the selection of *packages* to be included.

When run on a basic document as in our example L<sup>A</sup>T<sub>E</sub>X only loads a small subset of its possibilities in order to save time and memory. If you need additional functionality you can include it through the use of *packages*. Such a package might contain anything from a small set of related semantic markup commands up to sophisticated bibliographic functionality<sup>2</sup>. When you want a different style for tables or lists, to include images, or add music examples—chances are good that there is a package enabling just what you need.

As I have also mentioned earlier the *commands* and *environments* are roughly similar to character and paragraph styles as used in office documents. But they too are much more powerful because they are more like programmer's commands, especially as they can use arguments. A command can do anything with its arguments which makes it a very versatile tool. As a single example I have a command for entries in a revision report that takes arguments for measure number, position in the measure, affected voice/system, and the actual comment. This information it then processed to a nice layout, skipping the separator characters when an argument is empty. If I should decide upon a completely different layout I just have to update the command to propagate it to the whole document—or a complete set of documents if I have defined it in a

---

<sup>1</sup>Integrated Development Environments

<sup>2</sup>See [http://en.wikibooks.org/wiki/LaTeX/Package\\_Reference](http://en.wikibooks.org/wiki/LaTeX/Package_Reference) for a list of some packages with short comments on their functionality

package. And of course I will soon be able to incorporate the report entries that `lilypond-doc` will give me from within the LilyPond scores ...

*Environments* are somewhat similar, with the difference that the commands are a one-time event while environments have a begin and an end (like the *document environment* seen in the initial example). Environments like `figure` or `table` allow you to place graphical items as *floating* objects. This allows  $\LaTeX$  to place them at the best position concerning the page breaking. While this is something one has to get used to it is actually a big enhancement towards a *professional* typesetting, especially of books and similar documents. As producing books is the basic motivation for  $\LaTeX$  you also can expect professional tools with regard to bibliography, indices or figure lists etc.

Now that we have seen a *very* short introduction to  $\LaTeX$  it is time to consider some specifically *musical* aspects to authoring text documents.

## 4.1. Music Examples

Of course you can embed images in text documents if you want to add music examples to a text.  $\LaTeX$  does a very good job at determining the best possible layout for them. But there are a few dedicated tools that aid you with this specific task, in particular I will talk about `lilypond-book` and `musicexamples`.

### 4.1.1. `lilypond-book`

The program `lilypond-book` is part of the LilyPond distribution and allows to embed LilyPond source code directly in  $\LaTeX$  documents. First you run `lilypond-book` on the document, which will call LilyPond to generate all (necessary) examples and create a copy of the document including these generated files. Then you run  $\LaTeX$  to compile this processed copy. This procedure has some overhead due to the large number of intermediate files, and many people using `lilypond-book` with complex documents or on a regular basis actually approach this using self-made scripts. But if you have documents with lots of (small) examples you won't find any better solution to manage your music sources *in situ*.

### 4.1.2. `musicexamples`

Another way to embed music examples in  $\LaTeX$  documents is `musicexamples`, a package that is part of `openLilyLib`<sup>3</sup>. Originally started as an alternative approach to `lilypond-book` not requiring intermediate documents it can actually work together with it quite well. `musicexamples` provides environments and commands to include and manage music examples in  $\LaTeX$  documents. There is support for floating and non-floating environments, single-line or multi-line examples (with the multi-line ones being able to be split on several pages). You can also print full-page examples, with special support for examples starting on odd or even pages (you can force an example to start at the next odd or even page). All examples in a document share the same counter and can be exported to a contiguous list of music examples.

---

<sup>3</sup><http://www.openlilylib.org/musicexamples>

It is possible to use *any* images for your music examples, but there is special support for using LilyPond (of course). There are tools compiling your LilyPond files to files that can directly be used in the  $\LaTeX$  document, especially taking care of the odd/even issue. This may be the way for you if you want to manage your scores separately from the text document. Finally there will be<sup>4</sup> helper scripts that keep the music examples up to date (and conditionally recompile missing music example files).

Instead of existing image files (LilyPond generated or not) you can equally embed LilyPond code with `lilypond-book` and benefit from both approaches: `lilypond-books` in-place example code and `musicexamples`' example management.


You can see a comprehensive example document with all sorts of music examples in example A.2 on page 46.


### 4.1.3. OOoLilyPond

Finally I have to mention another option to integrate music examples in text documents, although it isn't quite as professional. *OOoLilyPond*<sup>5</sup> is an OpenOffice extension which allows to store LilyPond code as embedded objects in Writer documents. It can compile them in place and display the corresponding image files while keeping the source code ready for further editing. Unfortunately this can only produce bitmaps and no vector graphics, resulting in a rather medium output quality. But for daily use it may be a good compromise.

## 4.2. Notational Elements

Another member of our family of resources is *lilyglyphs*. This package allows to include LilyPond's notational elements in the continuous text of  $\LaTeX$  documents. This is significantly different from entering music examples as it will embed them as characters, like *mf* for example. The main advantage of this package over any other solution I know of is that these elements are readily available, scale well with the surrounding text size, and that you can realize virtually anything you can do with LilyPond.

Elements that are part of LilyPond's OpenType font can be accessed through predefined commands (the above example was written `\lilyDynamics{mf}`) or—if there isn't a predefined command yet—through their glyph names from the font, like this  mensural clef `\lilyGlyph{clefs.mensural.c}`.

For elements that aren't part of the OpenType font and that are normally drawn by LilyPond *lilyglyphs* uses small images that have been generated with LilyPond. The use is identical if there are predefined commands like this  `\crotchet` or other non-text  $\leq$  elements. If there isn't a predefined command or if you need a very non-standard symbol, *lilyglyphs* assists you in creating your own predefined commands with a relatively easy to use interface: You will provide a small "template file", and a tool will manage the process for you up to generating the necessary  $\LaTeX$  command. This way you can include arbitrary notational elements as characters in your text documents, with perfect control over the scaling and positioning.

<sup>4</sup>This part of the package hasn't been implemented yet.

<sup>5</sup><http://oolilypond.sourceforge.net/>

### 4.3. Conclusion

If you care at all about typography and write text documents you should love  $\LaTeX$ . If you don't care about typography you should know that typography *is* effective even if you didn't intend it. Typography can't be absent, usually it's just *bad* typography then. Therefore documents should be created with good tools.  $\LaTeX$  produces good typography, word processors don't, period. DTP programs like InDesign™ or QuarkXPress™ can be used to create good-looking documents, but they are very expensive, and especially for long documents they are significantly less efficient. If you add to that the described possibilities to integrate text and music and especially the potentials of versioning and collaboration you should see that  $\LaTeX$  is definitely worth giving a try. If you are already acquainted with LilyPond the learning curve won't be as steep anymore, although it surely needs some change of mind when coming from office applications.

If you have to deliver documents in other forms than PDF files (usually office files that hopefully are typeset with professional tools afterwards) you still can benefit from the versioning part which improves your efficiency— I wouldn't want to work any other way by now. Other than LilyPond scores  $\LaTeX$  source files can be exported to a variety of formats like RTF or HTML. This process isn't always guaranteed, though—the more sophisticated your tools and commands are the more problems will arise with export filters. Which should be evident: If you design complex custom commands with multiple arguments, how should a converter know to translate them to RTF for example? So if you write program notes that have to be delivered as a Word file you can still use  $\LaTeX$  to author the document but you should keep it simple in terms of layout or structure. But this shouldn't really be considered a restriction because if you're working in such a context you are usually expected to deliver “raw” material anyway.

## 5. Applications

### 5.1. Preparing a Musical Edition

This is what I actually experienced recently—and what I definitely don’t want to miss anymore. During the preparation of an edition I’m currently working on we switched our workflow from sharing files in a Dropbox<sup>1</sup> to versioning it with Git. It was a real revelation to me, and I don’t know how we managed before. This becomes especially obvious when we are dealing with issues that date from before that time and there isn’t any Git history present to inspect them.

There are several steps in preparing a scholarly edition, which are partly interdependent and are partly distributed between different team members:

- Entering the music
- Setting up the score layout and the style sheets
- Proof-reading the music
- Scholarly revision
- Fine-tuning the engraving
- Authoring critical reports
- Layout of the main volume and prepress

First of all the complete process can be done within one code base. Any collaborator can edit anything at any time, usually without stepping on others’ feet, as described in the “Versioning” chapter 2 on page 14. Traditionally one would work with one file (or one for each piece/movement) and would have to take great care about potential version conflicts. Basically these files would have to be completely processed by one person before being passed on to the next in the chain.

An important side-effect of this situation is that tasks can be shared in a much more flexible way. As anybody can edit everything he can also handle any task he feels able or has time to. Most of us aren’t completely single-minded and specialised on a single skill. My main part in this project is being one of the scholarly editors, but I have also entered parts of the music and can do a lot for pinpointing typographical issues. The guy who is responsible for the fine-engraving has a good eye on his own and can also spot questionable issues in the musical text.

We have developed the concept of a “draft mode” that allows us to visualize (i. e. mostly colorize) issues and have some kind of “in-source” communication through messages that are written to the command line. When a piece is finished we just switch to “publication mode” and have our print-ready scores. Comments in the source files increase the impact of this concept, but the next project will take this a major step further. Currently I’m developing a system that

---

<sup>1</sup>Dropbox is a cloud based file hosting provider. See <http://www.dropbox.com>

## 5. Applications

will allow comprehensive annotations inside the source files. This way we'll have automatic lists of TODO-items, issues and questions etc. that are generated while compiling the scores. And ultimately we'll have the entries for the critical report right in the source files!

As all general layout and styling is done in central style sheets it isn't an issue at all to keep the different scores consistent. We don't even have to decide upon it before starting work. All changes to the layout are automatically propagated to all scores, and it would for example be possible that someone is assigned the task to develop the general appearance of the publication *while the others are working on the edition*, always having the latest version of the musical text as example material. That way the edition can also automatically benefit from improvements of LilyPond that are released along the way. Only when starting the final beautification would the development of the layout settings have to stop—but that's a restriction that is inherent in the matter and not imposed by the technical infrastructure.

While this isn't that important for small individual scores it is a major “selling point” for ambitious and large-scale projects. And I would love to discuss and experience the impact this could have on working on a scholarly edition with numerous volumes and multiple editors. Remember what I wrote about plain text files outliving operating systems' lifecycles? Such a system could very well be the perfect match for a long-running critical edition.

### 5.2. Book preparation

The same is true for the preparation of books, and of compilation books from more than one author in particular. All collaborators can seamlessly work together in a shared code base: the author(s), editors, typesetters, and maybe additional specialists preparing figures or music examples. A collaborative set-up provides an efficient workflow between an author and his editor, and it can be particularly useful when proof-reading or documents or polishing them linguistically with many assistants. It has to be said that such a work-flow is already standard in many scientific journals in other disciplines.

Of course the musical edition described in the previous section is also a “book” in this sense, as the scores are created with LilyPond and included in a  $\text{\LaTeX}$  document. The complete volume can be compiled at any time and reflects the current state of the development of the scores.

### 5.3. “Crowd Editing”

It isn't really common yet because it can exclusively be inspired by the use of workflows that are driven by version control. But with the capability to manage contributions by a (practically) unlimited number of participants it is possible to extend the concepts from the the two previous sections and create musical scores with a large community. This way the workload can be “clustered” among a potentially great number of people, each one according to his capabilities and special knowledge (about specific instruments for example), and large quantities of music entered in astonishingly little time. While this doesn't save actual working hours it can be used to process them in parallel. Even in a large-scale symphonic score each single part is quite manageable, and if there is one contributor for each part (or better: two contributors for two parts who do a peer-review before submitting) the score should grow rapidly. I know of



## 5. Applications

at least one semi-professional choir that has by now switched to preparing its scores in such a social manner.

In such a context the aspect of programmability may once more become important. While the individual contributors don't have to be programmers at all the "project" can be extensively programmed. It can gain advanced and specialized functionality, benefit from automatic project management or just from giving the "ordinary engraver" an easy-to-use interface.

### 5.4. Single-source Publishing

Plain text files are a good starting point for creating documents for different output formats. One example already discussed is the possibility to create several different scores (or parts) with LilyPond from a single set of input files. I don't have concrete experiences to present but it's inherent in the nature of plain text files that they are good for use in single-source publishing. This can easily apply to generating documents for printing, beamer presentation, mobile devices and web pages from single sources, to name just a few. I think—and hope—that this can also be made very fruitful in the light of current discussions to present scholarly editions in modern ways beyond printed scores.

## 6. Where To Go From Here?

What I've laid out for you over the previous few dozen pages of course only slightly scratches the surface. You have seen a few examples that were necessary to get an impression of the material, but in general I talked on a rather conceptual level. This document isn't intended for and can't be used to actually start working with the tools in question. It could be the opening chapter of a valuable book on document authoring but I don't see anybody having spare time to write the rest of it ...

Therefore I'm going to provide you with references to further reading on the different subjects. If you know of or find interesting introductions on any level feel free to tell me so I can extend the list.

### Plain Text

<http://www.terminally-incoherent.com/blog/2012/05/21/why-plain-text/> is a blog post that I have read only after having written this document. I think it tells you everything that I have written on the characteristics of plain text files, but from a different perspective and maybe even more urgent and convincing.

### Version Control

<http://pages.cs.wisc.edu/~driscoll/software/vcs/index.html> gives a visually rich introduction to the concepts of version control.

### Git

- <http://git-scm.com/book>  
The official documentation
- <http://sixrevisions.com/resources/git-tutorials-beginners/>  
List with presumably easy introductions
- <http://www.vogella.com/articles/Git/article.html>  
Long introduction with comprehensive usage examples

You get Git at:

- <http://git-scm.com>  
The official source.  
On Linux you will probably have it in your software repositories  
On Windows and Mac you can also get it bundled with one of the GUI tools, for example from <https://github.com> or <https://www.bitbucket.org>.

There are numerous GUI tools around, a number of them collected on <http://git-scm.com/downloads/guis>. But I recommend starting to use it on the command line because this way you will get a better understanding of what's happening exactly.

### LilyPond

The main resource for LilyPond is its website <http://www.lilypond.org>. There you'll find the software as well as a truly huge manual. Of particular interest for now are the Introduction<sup>1</sup> and the Learning Manual<sup>2</sup>.

A most welcome resource is the `lilypond-user` mailing list<sup>3</sup> where you can find tons of information and get help with your current problems. Two new resources are the openLilyLib project<sup>4</sup> (where you'll also find information on `musicexamples`, `lilypond` and `lilypond-doc`), and the *Scores of Beauty* blog<sup>5</sup>.

### L<sup>A</sup>T<sub>E</sub>X

Given the age of the L<sup>A</sup>T<sub>E</sub>X project it is not surprising that there exists a wealth of documentation and tutorials about using it. Therefore I will only direct you to the project's homepage <http://latex-project.org> and especially the page <http://latex-project.org/guides/> there which is a collection of links to resources that have been selected by the project members themselves.

For German readers I can additionally recommend one book: Joachim Schlosser, *Wissenschaftliche Arbeiten schreiben mit L<sup>A</sup>T<sub>E</sub>X* (<http://www.latexbuch.de>). It has helped me getting into the spirit of L<sup>A</sup>T<sub>E</sub>X by taking the perspective of an academic writer and providing information (with the necessary amount and complexity) to start authoring scholarly documents.

---

<sup>1</sup><http://www.lilypond.org/introduction.html>

<sup>2</sup><http://www.lilypond.org/doc/v2.16/Documentation/learning/index.html>

<sup>3</sup><https://lists.gnu.org/mailman/listinfo/lilypond-user>

<sup>4</sup><http://www.openlilylib.org>

<sup>5</sup><http://lilypondblog.org>

# A. Appendix

## A.1. Comparison of File Formats

In this appendix I'd like you to contemplate a little bit over a few raw listings of file contents of different programs. We start with a minimal score created with LilyPond



Music Example A.1: A minimal LilyPond score, with its *full* listing: `{ c' }`

On the following two pages you see the beginning of the printouts of the files that form a comparable mini-score in Finale™ and Sibelius™. I restricted it to the first of 15 and 11 pages.

After that we'll do the same with a text document whose L<sup>A</sup>T<sub>E</sub>X source looks like this:

```
\documentclass{article}
\begin{document}
  \section*{Überschrift}
  Das ist ein Text mit \emph{hervorgehobenem} Element.
\end{document}
```

So after the score files you'll see the text document and then its corresponding document files (first pages) from:

- Word 2007 (.docx)
- Word 1997–2003 (.doc)
- Word 2003 XML (.xml)
- OpenOffice (.odt)
- OpenOffice „Flat XML“ (.fodt)

You may notice that both Word and OpenOffice provide XML files that somehow qualify as plain text files in that they *can* be read by a human. But you also see that the content information is really buried inside all that markup, so it isn't really usable as a plain text format. Besides these formats aren't practical for versioning as they contain lots of “volatile” information like timestamps or window positionings. In my experience such a file may be modified considerably just by opening it on another computer.

[illegible]

[illegible]

Überschrift

Das ist ein Text mit **hervorgehobenem** Element.

[illegible]



[illegible]

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?mso-application progid="Word.Document"?>
<wordDocument xmlns:aml="http://schemas.microsoft.com/aml/2001/core" xmlns:wpc="http://
schemas.microsoft.com/office/word/2010/wordprocessingCanvas" xmlns:dt="uuid:C2F41010-65B3-11d1-
A29F-000AA00C14882" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:o="urn:schemas-microsoft-com:office:office" xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:w10="urn:schemas-microsoft-com:office:word" xmlns:w="http://schemas.microsoft.com/office/
word/2003/wordml" xmlns:wx="http://schemas.microsoft.com/office/word/2003/auxHint" xmlns:wne="http://
schemas.microsoft.com/office/word/2006/wordml" xmlns:wsp="http://schemas.microsoft.com/office/
word/2003/wordml/sp2" xmlns:sl="http://schemas.microsoft.com/schemaLibrary/2003/core"
w:macrosPresent="no" w:embeddedObjPresent="no" w:ocxPresent="no" xml:space="preserve"><w:ignoreSubtree
w:val="http://schemas.microsoft.com/office/word/2003/wordml/sp2"/><o:DocumentProperties><o:Author>Katja
Liska</o:Author><o:LastAuthor>Katja Liska</o:LastAuthor><o:Revision>2</o:Revision><o:TotalTime>0</
o:TotalTime><o:Created>2013-04-24T09:59:00Z</o:Created><o:LastSaved>2013-04-24T09:59:00Z</
o:LastSaved><o:Pages>1</o:Pages><o:Words>7</o:Words><o:Characters>51</o:Characters><o:Lines>1</
o:Lines><o:Paragraphs>1</o:Paragraphs><o:CharactersWithSpaces>57</
o:CharactersWithSpaces><o:Version>14</o:Version></o:DocumentProperties><w:fonts><w:defaultFonts
w:ascii="Calibri" w:fareast="Calibri" w:h-ansi="Calibri" w:cs="Times New Roman"/><w:font w:name="Times
New Roman"><w:panose-1 w:val="02020603050405020304"/><w:charset w:val="00"/><w:family w:val="Roman"/
><w:pitch w:val="variable"/><w:sig w:usb-0="E0002AFF" w:usb-1="C0007841" w:usb-2="00000009"
w:usb-3="00000000" w:csb-0="000001FF" w:csb-1="00000000"/></w:font><w:font w:name="Cambria
Math"><w:panose-1 w:val="02040503050406030204"/><w:charset w:val="01"/><w:family w:val="Roman"/
><w:notTrueType/><w:pitch w:val="variable"/></w:font><w:font w:name="Calibri"><w:panose-1
w:val="020F0502020204030204"/><w:charset w:val="00"/><w:family w:val="Swiss"/><w:pitch
w:val="variable"/><w:sig w:usb-0="E00002FF" w:usb-1="4000ACFF" w:usb-2="00000001" w:usb-3="00000000"
w:csb-0="0000019F" w:csb-1="00000000"/></w:font></w:fonts><w:styles><w:versionOfBuiltInStyleNames
w:val="7"/><w:latentStyles w:defLockedState="off" w:latentStyleCount="267"><w:lsdException
w:name="Normal"/><w:lsdException w:name="heading 1"/><w:lsdException w:name="heading 2"/
><w:lsdException w:name="heading 3"/><w:lsdException w:name="heading 4"/><w:lsdException
w:name="heading 5"/><w:lsdException w:name="heading 6"/><w:lsdException w:name="heading 7"/
><w:lsdException w:name="heading 8"/><w:lsdException w:name="heading 9"/><w:lsdException w:name="toc
1"/><w:lsdException w:name="toc 2"/><w:lsdException w:name="toc 3"/><w:lsdException w:name="toc 4"/
><w:lsdException w:name="toc 5"/><w:lsdException w:name="toc 6"/><w:lsdException w:name="toc 7"/
><w:lsdException w:name="toc 8"/><w:lsdException w:name="toc 9"/><w:lsdException w:name="caption"/
><w:lsdException w:name="Title"/><w:lsdException w:name="Default Paragraph Font"/><w:lsdException
w:name="Subtitle"/><w:lsdException w:name="Strong"/><w:lsdException w:name="Emphasis"/><w:lsdException
w:name="Table Grid"/><w:lsdException w:name="Placeholder Text"/><w:lsdException w:name="No Spacing"/
><w:lsdException w:name="Light Shading"/><w:lsdException w:name="Light List"/><w:lsdException
w:name="Light Grid"/><w:lsdException w:name="Medium Shading 1"/><w:lsdException w:name="Medium Shading
2"/><w:lsdException w:name="Medium List 1"/><w:lsdException w:name="Medium List 2"/><w:lsdException
w:name="Medium Grid 1"/><w:lsdException w:name="Medium Grid 2"/><w:lsdException w:name="Medium Grid 3"/
><w:lsdException w:name="Dark List"/><w:lsdException w:name="Colorful Shading"/><w:lsdException
w:name="Colorful List"/><w:lsdException w:name="Colorful Grid"/><w:lsdException w:name="Light Shading
Accent 1"/><w:lsdException w:name="Light List Accent 1"/><w:lsdException w:name="Light Grid Accent 1"/
><w:lsdException w:name="Medium Shading 1 Accent 1"/><w:lsdException w:name="Medium Shading 2 Accent
1"/><w:lsdException w:name="Medium List 1 Accent 1"/><w:lsdException w:name="Revision"/
><w:lsdException w:name="List Paragraph"/><w:lsdException w:name="Quote"/><w:lsdException
w:name="Intense Quote"/><w:lsdException w:name="Medium List 2 Accent 1"/><w:lsdException w:name="Medium
Grid 1 Accent 1"/><w:lsdException w:name="Medium Grid 2 Accent 1"/><w:lsdException w:name="Medium Grid
3 Accent 1"/><w:lsdException w:name="Dark List Accent 1"/><w:lsdException w:name="Colorful Shading
Accent 1"/><w:lsdException w:name="Colorful List Accent 1"/><w:lsdException w:name="Colorful Grid
Accent 1"/><w:lsdException w:name="Light Shading Accent 2"/><w:lsdException w:name="Light List Accent
2"/><w:lsdException w:name="Light Grid Accent 2"/><w:lsdException w:name="Medium Shading 1 Accent 2"/
><w:lsdException w:name="Medium Shading 2 Accent 2"/><w:lsdException w:name="Medium List 1 Accent 2"/
><w:lsdException w:name="Medium List 2 Accent 2"/><w:lsdException w:name="Medium Grid 1 Accent 2"/
><w:lsdException w:name="Medium Grid 2 Accent 2"/><w:lsdException w:name="Medium Grid 3 Accent 2"/
><w:lsdException w:name="Dark List Accent 2"/><w:lsdException w:name="Colorful Shading Accent 2"/
><w:lsdException w:name="Colorful List Accent 2"/><w:lsdException w:name="Colorful Grid Accent 2"/
><w:lsdException w:name="Light Shading Accent 3"/><w:lsdException w:name="Light List Accent 3"/
><w:lsdException w:name="Light Grid Accent 3"/><w:lsdException w:name="Medium Shading 1 Accent 3"/
><w:lsdException w:name="Medium Shading 2 Accent 3"/><w:lsdException w:name="Medium List 1 Accent 3"/
><w:lsdException w:name="Medium List 2 Accent 3"/><w:lsdException w:name="Medium Grid 1 Accent 3"/
><w:lsdException w:name="Medium Grid 2 Accent 3"/><w:lsdException w:name="Medium Grid 3 Accent 3"/
><w:lsdException w:name="Dark List Accent 3"/><w:lsdException w:name="Colorful Shading Accent 3"/
><w:lsdException w:name="Colorful List Accent 3"/><w:lsdException w:name="Colorful Grid Accent 3"/
><w:lsdException w:name="Light Shading Accent 4"/><w:lsdException w:name="Light List Accent 4"/
><w:lsdException w:name="Light Grid Accent 4"/><w:lsdException w:name="Medium Shading 1 Accent 4"/
><w:lsdException w:name="Medium Shading 2 Accent 4"/><w:lsdException w:name="Medium List 1 Accent 4"/
><w:lsdException w:name="Medium List 2 Accent 4"/><w:lsdException w:name="Medium Grid 1 Accent 4"/
><w:lsdException w:name="Medium Grid 2 Accent 4"/><w:lsdException w:name="Medium Grid 3 Accent 4"/
><w:lsdException w:name="Dark List Accent 4"/><w:lsdException w:name="Colorful Shading Accent 4"/
><w:lsdException w:name="Colorful List Accent 4"/><w:lsdException w:name="Colorful Grid Accent 4"/
><w:lsdException w:name="Light Shading Accent 5"/><w:lsdException w:name="Light List Accent 5"/
><w:lsdException w:name="Light Grid Accent 5"/><w:lsdException w:name="Medium Shading 1 Accent 5"/
><w:lsdException w:name="Medium Shading 2 Accent 5"/><w:lsdException w:name="Medium List 1 Accent 5"/
```



```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<office:document xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
  xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0"
  xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
  xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
  xmlns:draw="urn:oasis:names:tc:opendocument:xmlns:drawing:1.0"
  xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" xmlns:xlink="http://
www.w3.org/1999/xlink" xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
  xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"
  xmlns:svg="urn:oasis:names:tc:opendocument:xmlns:svg-compatible:1.0"
  xmlns:chart="urn:oasis:names:tc:opendocument:xmlns:chart:1.0"
  xmlns:dr3d="urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0" xmlns:math="http://www.w3.org/1998/Math/
MathML" xmlns:form="urn:oasis:names:tc:opendocument:xmlns:form:1.0"
  xmlns:script="urn:oasis:names:tc:opendocument:xmlns:script:1.0"
  xmlns:config="urn:oasis:names:tc:opendocument:xmlns:config:1.0" xmlns:ooo="http://openoffice.org/2004/
office" xmlns:ooow="http://openoffice.org/2004/writer" xmlns:oooc="http://openoffice.org/2004/calc"
  xmlns:dom="http://www.w3.org/2001/xml-events" xmlns:xforms="http://www.w3.org/2002/xforms"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:rpt="http://openoffice.org/2005/report" xmlns:of="urn:oasis:names:tc:opendocument:xmlns:of:1.2"
  xmlns:xhtml="http://www.w3.org/1999/xhtml" xmlns:grddl="http://www.w3.org/2003/g/data-view#"
  xmlns:tableooo="http://openoffice.org/2009/table" xmlns:field="urn:openoffice:names:experimental:ooo-
ms-interop:xmlns:field:1.0" xmlns:formx="urn:openoffice:names:experimental:ooxml-odf-
interop:xmlns:form:1.0" xmlns:css3t="http://www.w3.org/TR/css3-text/" office:version="1.2"
  office:mimetype="application/vnd.oasis.opendocument.text">
  <office:meta><meta:initial-creator>Katja Liska</meta:initial-creator><meta:creation-
date>2013-04-24T11:59:00</meta:creation-date><dc:creator>Katja Liska</
dc:creator><dc:date>2013-04-24T11:59:00</dc:date><meta:editing-cycles>2</meta:editing-
cycles><meta:document-statistic meta:table-count="0" meta:image-count="0" meta:object-count="0"
meta:page-count="1" meta:paragraph-count="2" meta:word-count="8" meta:character-count="56" meta:non-
whitespace-character-count="50"/><meta:generator>LibreOffice/3.5$Linux_x86 LibreOffice_project/350m1
$Build-2</meta:generator><meta:user-defined meta:name="Info 1"/><meta:user-defined meta:name="Info 2"/
><meta:user-defined meta:name="Info 3"/><meta:user-defined meta:name="Info 4"/></office:meta>
  <office:settings>
    <config:config-item-set config:name="ooo:view-settings">
      <config:config-item config:name="ViewAreaTop" config:type="int">0</config:config-item>
      <config:config-item config:name="ViewAreaLeft" config:type="int">0</config:config-item>
      <config:config-item config:name="ViewAreaWidth" config:type="int">37816</config:config-item>
      <config:config-item config:name="ViewAreaHeight" config:type="int">15714</config:config-item>
      <config:config-item config:name="ShowRedlineChanges" config:type="boolean">true</config:config-item>
      <config:config-item config:name="InBrowseMode" config:type="boolean">>false</config:config-item>
      <config:config-item-map-indexed config:name="Views">
        <config:config-item-map-entry>
          <config:config-item config:name="ViewId" config:type="string">view2</config:config-item>
          <config:config-item config:name="ViewLeft" config:type="int">10906</config:config-item>
          <config:config-item config:name="ViewTop" config:type="int">3501</config:config-item>
          <config:config-item config:name="VisibleLeft" config:type="int">0</config:config-item>
          <config:config-item config:name="VisibleTop" config:type="int">0</config:config-item>
          <config:config-item config:name="VisibleRight" config:type="int">37814</config:config-item>
          <config:config-item config:name="VisibleBottom" config:type="int">15713</config:config-item>
          <config:config-item config:name="ZoomType" config:type="short">0</config:config-item>
          <config:config-item config:name="ViewLayoutColumns" config:type="short">0</config:config-item>
          <config:config-item config:name="ViewLayoutBookMode" config:type="boolean">>false</config:config-
item>
          <config:config-item config:name="ZoomFactor" config:type="short">130</config:config-item>
          <config:config-item config:name="IsSelectedFrame" config:type="boolean">>false</config:config-item>
        </config:config-item-map-entry>
      </config:config-item-map-indexed>
    </config:config-item-set>
    <config:config-item-set config:name="ooo:configuration-settings">
      <config:config-item config:name="PrintAnnotationMode" config:type="short">0</config:config-item>
      <config:config-item config:name="PrintHiddenText" config:type="boolean">>false</config:config-item>
      <config:config-item config:name="PrintDrawings" config:type="boolean">true</config:config-item>
      <config:config-item config:name="PrintProspectRTL" config:type="boolean">>false</config:config-item>
      <config:config-item config:name="PrintBlackFonts" config:type="boolean">>false</config:config-item>
      <config:config-item config:name="PrintTextPlaceholder" config:type="boolean">>false</config:config-
item>
      <config:config-item config:name="PrintProspect" config:type="boolean">>false</config:config-item>
      <config:config-item config:name="PrintSingleJobs" config:type="boolean">>false</config:config-item>
      <config:config-item config:name="PrintRightPages" config:type="boolean">true</config:config-item>
      <config:config-item config:name="PrintGraphics" config:type="boolean">true</config:config-item>
```

## **A.2. musicexamples Document**

On the following pages you'll see a complete document with all sorts of music examples in it.



## musicexamples Example Document

This document is a blindtext with music examples of all kinds interspersed. It is an example of the use of the `musicexamples` L<sup>A</sup>T<sub>E</sub>X package. There are hints highlighted in red where the examples are included in the score. At the end of the document you'll find a list of music examples.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue dui dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consetetur adipiscing elit, sed diam nonumy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

We start with a regular full-page example on the following page.

```
\fullPageLilypondExample{default-score}  
  {Default fullpage example}  
  {\label{xmp:default-fullpage-example}}
```

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue dui dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consetetur adipiscing elit, sed diam nonumy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis.

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem

Music Example 1: Default fullpage example

Ignaz Edlen von Mosen gewidmet  
**Heidenröslein**  
Für eine Singstimme mit Begleitung des Pianoforte  
componirt von

Gedicht von J. W. v. Goethe

Franz Schubert  
Op. 3, No. 3

**Lieblich.** (♩ = 69)

Singstimme

Pianoforte

Sah ein Knab' ein Rös-lein\_stehn, Rös-lein auf der

Hei - den, war so jung und mor - gen-schön, lief er schnell es

nachgebend

nah'\_zu\_sehn, sah's\_mit\_ vie - len\_ Freu - den. Rös-lein, Rös-lein,

wie oben

Rös-lein.rot, Röslein auf der Hei - den.

ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, At accusam aliquyam diam diam dolore dolores duo eirmod eos erat, et nonumy sed tempor et et invidunt justo labore Stet clita ea et gubergren, kasd magna no rebum. sanctus sea sed takimata ut vero voluptua. est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat.

Now we enter a fullpage example that will start on the next odd page. This means that in this case the example will skip a complete page. Note that this works because the actual file on disk is named `odd-page-score-odd.pdf`.

```
\fullPageLilypondExample{odd-page-score}
{Fullpage example starting on odd page}
{\label{xmp:odd-page-fullpage-example}}
```

Consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue dui dolore



te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Please note that you can't (so far) enter any further fullpage music examples between the last command and the actual printing of the fullpage example, because the command would then try to print the wrong file. See <https://sourceforge.net/p/openlilylib/tickets/9/>.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis.

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, At accusam aliquyam diam diam dolore dolores duo eirmod eos erat, et nonumy sed tempor et et invidunt justo labore Stet clita ea et gubergren, kasd magna no rebum. sanctus sea sed takimata ut vero voluptua. est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat.

Consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy

Music Example 2: Fullpage example starting on odd page

Ignaz Edlen von Mosen gewidmet  
**Heidenröslein**  
Für eine Singstimme mit Begleitung des Pianoforte  
componirt von  
Gedicht von J. W. v. Goethe Franz Schubert  
Op. 3, No. 3

**Lieblich.** (♩ = 69)

Singstimme

Pianoforte

Sah ein Knab' ein Rös-lein\_stehn, Rös-lein auf der

Hei - den, war so jung und mor - gen-schön, lief er schnell es

nah'\_zu\_sehn, sah's\_mit\_ vie - len\_ Freu - den. Rös-lein, Rös-lein,

*nachgebend*

12 *wie oben*

Rös-lein.rot, Röslein auf der Hei - den.

eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Now we enter a fullpage example that doesn't have a corresponding pdf file. This can be used to enter placeholders for music examples that haven't been finished yet. Note that the entry in the list of music examples is printed red.

```
\fullPageLilypondExample{missing-score}
{Fullpage example whose pdf file isn't present}
{\label{xmp:missing-fullpage-example}}
```

**Missing full-page example:**  
missing-score

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis.

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Now we enter a single system example inside a non-floating environment. Note the three pairs of curly braces that are necessary to deal with the dot in the filename.

```
\begin{musicExampleNonFloat}
  \lilypondSFE{{{single-system-example.preview}}}}
  \caption{Single system music example (floating)}
  \label{xmp:single-system-floating}
\end{musicExampleNonFloat}
```



Music Example 4: Single system music example (non-floating)

Now we enter a single system example inside a floating environment. In this case  $\LaTeX$  places it at the top of the following page, but it could also be placed somewhere else because it is floating. Note the three pairs of curly braces that are necessary to deal with the dot in the filename.

```
\begin{musicExample}
  \lilypondSFE{{{single-system-example.preview}}}}
  \caption{Single system music example (floating)}
  \label{xmp:single-system-floating}
\end{musicExample}
```

Consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam



Music Example 5: Single system music example (floating)

et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus.

Here we can enter a reference to example 1 on page 2 with the command `\fref{xmp:default-fullpage-example}`.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Having included the `rotating` package we can now print example 6 on the next page that is rotated by 90 degree:

```
\begin{sidewaysmusicexample}
  \lilypondSFE{{{single-system-example.preview}}}
  \caption{Sideways music example}
  \label{xmp:sidewaysexample}
\end{sidewaysmusicexample}
```

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue dui dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consetetur adipiscing elit, sed diam nonumy nibh euismod tincidunt

5 **Lieblich.** (♩ = 69)

The musical score is for a piece titled "Lieblich." in 2/4 time, with a tempo marking of a quarter note equal to 69 beats per minute. The score is written for piano on a grand staff. The key signature has one sharp (F#). The melody is in the treble clef, and the accompaniment is in the bass clef. The piece consists of 69 measures.

Music Example 6: Sideways music example

ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis.

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Having included the `sidecap` package we can now print example 7 where the caption is printed beside instead of below.

```
\begin{SCmusicexample}
  \lilypondSFE{{{really-short-example.preview}}}}
  \caption{Example with caption beside}
  \label{xmp:sidecapexample}
\end{SCmusicexample}
```



Music Example 7: Example with caption beside

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis.

Finally we can – having included the `wrapfig` package – print example 8 on the next page where the text is wrapped around the example.

```

\begin{wrapmusicexample}{4}{6.2cm}
  \lilypondSFE{{{really-short-example.preview}}}}
  \caption{Example with text warp}
  \label{xmp:wrapfigxample}
\end{wrapmusicexample}

```

Ut wisi enim ad minim veniam,  
quis nostrud exerci tation ullamcor-  
per suscipit lobortis nisl ut aliquip  
ex ea commodo consequat. Duis  
autem vel eum iriure dolor in hend-  
rerit in vulputate velit esse molestie  
consequat, vel illum dolore eu feugiat  
nulla facilisis at vero eros et accum-  
san et iusto odio dignissim qui blan-  
dit praesent luptatum zzril delenit  
augue duis dolore te feugait nulla fa-  
cilisi.



Music Example 8: Example with text warp

Nam liber tempor cum soluta no-  
bis eleifend option congue nihil im-  
perdiet doming id quod mazim placerat facer possim assum. Lorem ipsum  
dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod  
tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim  
veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip  
ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie  
consequat, vel illum dolore eu feugiat nulla facilisis.

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd  
gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem  
ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod  
tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At  
vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren,  
no sea takimata sanctus est Lorem ipsum dolor sit amet.

## List of Music Examples

1	Default fullpage example . . . . .	2
2	Fullpage example starting on odd page . . . . .	5
3	Fullpage example whose pdf file isn't present . . . . .	6
4	Single system music example (non-floating) . . . . .	7
5	Single system music example (floating) . . . . .	8
6	Sideways music example . . . . .	9
7	Example with caption beside . . . . .	10
8	Example with text warp . . . . .	11