Verbatim Text and Code Listings in LaTeX

Raphael Frey

https://github.com/alpenwasser/TeX/

March 25, 2017

Abstract

There are many ways to integrate verbatim text (text which is presented in the resulting document as it is in the source code) and listings (which may include pretty printing, syntax highlighting and other fancy things) into a LaTeX document, as can easily be seen when looking at the corresponding topic pages on CTAN [11, 12]. This document presents examples for a few of them. The idea is not to provide in-depth documentation, but to give a brief overview of a few possibilities to enable the reader to get an idea of what's out there and what might, potentially, be useful to them. Consulting the respective user manual is always recommended for more detailled information and achieving the best results.

For the *very* impatient: For most code listings, I use the `minted` package these days. While it is slow (at least on first compilation), it has served me well most of the time. But there are alternatives (hence this document), and personal preferences matter, too. One's mileage may vary.

Contents

1 The `verbatim` Environment and `\verb` Command

LaTeX provides a `verbatim` Environment which, although it has quite a few limitations, is simple to use and often gets the job done well enough. For fancier things, there exists a re-implementation with a few added niceties. For inline code, there is the `\verb` command.

1.1 Stock LaTeX

The stock LaTeX `verbatim` environment is used like this:

Code

```
\begin{verbatim}
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
\end{verbatim}
```

LaTeX Output

```
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
```

Writing inline code can be done with the `\verb` command:

Code

```
Inline verbatim content is
\verb|typeset inside regular text|
instead of being put in a special box or
paragraph or being similarly separated.
```

LaTeX Output

Inline verbatim content is `typeset inside regular text` instead of being put in a special box or paragraph or being similarly separated.

1.2 Re-Implementation of the `verbatim` and `verbatim*` Environments [1]

The re-implementation alleviates some limitations of the stock `verbatim` environment. For more details and to evaluate whether these limitations may be of relevance to your use case, consult the documentation at [1]. Basic usage is identical to the stock `verbatim` environment.

Code

```
\begin{verbatim}
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
\end{verbatim}
```

LaTeX Output

```
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
```

2 The `verbatimbox` Package [2]

A box is defined via `\begin{verbbox}[options] content \end{verbbox}`. This, however, does not yet display the box. Instead, it is stored for later use via the `\theverbbox` command. This can be easily wrapped inside an `\fbox`, optionally adding some vertical space with the `\addvbuffer` command.

Code

```
\begin{verbbox}[\arabic{VerbboxLineNo}:\hspace{1ex}]
#include <stdio.h>

main() {
    print("Hello, world!\n");
}\end{verbbox}
\fbox{\addvbuffer[10pt 5pt]\theverbbox} % different spacing above and below
\fbox{\addvbuffer[10pt]\theverbbox}     % same spacing above and below
```

LaTeX Output

```
1:     #include <stdio.h>
2:
3:     main() {
4:         print("Hello, world!\n");
5:     }
```

```
1:     #include <stdio.h>
2:
3:     main() {
4:         print("Hello, world!\n");
5:     }
```

As usual, check the documentation [2] for more options and possibilities. Also, the optional argument `[\arabic{VerbboxLineNo}:\hspace{1ex}]` to `\begin{verbbox}` must be on the same line and only on that line. The `\end{verbbox}` is on the same line as the last line of code so that no additional empty line is added to the output.

For cases where the limitation of having only a single named box to which one can refer, `\theverbbox`, must be overcome, there exists the `\myverbbox` command, by which named boxes can be created for later use. One use case for this is the use of verbatim environments in tabular environments, where the use of `verbatim` is not allowed (example from the `verbatimbox` manual [2]).

Code

```
\begin{myverbbox}{\vtheta}\theta\end{myverbbox}
\begin{myverbbox}{\valpha}\alpha\end{myverbbox}
\begin{tabular}{|c|c|} \hline
\valpha & $\alpha$ \\ \hline
\vtheta & $\theta$ \\ \hline
\end{tabular}
```

LaTeX Output

`\alpha`	α
`\theta`	θ

`verbatimbox` also allows including the contents of an entire file:

Code

```
\verbfilebox{code/hw.c}
\theverbbox
```

LaTeX Output

```
#include <stdio.h>

main() {
print("Hello, world!\n");
}
```

Because `verbatimbox` uses boxes to save its contents, they cannot be broken across pages. For breaking verbatims across pages, there exist the `verbnobox` environment and the `\verbfilenobox` command. These do not allow recalling the verbatim contents at a later point though; their output is generated directly where they are placed in the source code.

Code

```
\begin{verbnobox}[\arabic{VerbboxLineNo}:\hspace{1ex}]
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
\end{verbnobox}
```

LaTeX Output

```
1:     #include <stdio.h>
2:
3:     main() {
4:         print("Hello, world!\n");
5:     }
6:
```

Code

```
\verbfilenobox{code/hw.c}
```

LaTeX Output

```
#include <stdio.h>
main() {
print("Hello, world!\n");
}
```

3 The `fancyvrb` Package [3]

`fancyvrb` offers several advancements over the standard LaTeX verbatim environment. Some of them are:

- Footnotes can contain verbatim content.
- Several different verbatim environments are provided, and it is possible to create new environments.
- Verbatim content can be stored and recalled.
- Files can be read and written in verbatim mode.

Putting verbatim content in footnotes is accomplished by invoking the `\VerbatimFootnotes` command after the preamble at some point, after which verbatim content can be put inside footnotes:

3.1 Footnotes

After invoking VerbatimFootnotes, verbatim can be put into footnotes[1]. Note that the verbatim content in the footnote must be on a single line (though the footnote text itself can break across multiple lines of source code.)

Code

```
\VerbatimFootnotes
And then we have some text\footnotemark with a footnote, and that footnote
shall contain verbatim content.
\footnotetext{\verb+_And here  we are,  down in the  footnotes!!_+ This is
outside the verbatim content in the footnote.}
```

LaTeX Output

And then we have some text[1] with a footnote, and that footnote shall contain verbatim content.

3.2 Inline Verbatim Content

`fancyvrb` allows convenient inlining of verbatim content via the `\DefineShortVerb`, the `\UnDefineShortVerb` and the `\Verb` commands[2]:

Code

```
We can write \Verb+_inline verbatim_+
by defining delimiting characters ad
hoc, or via:
\DefineShortVerb{\|}
And now we  can use
|_this_ _delimiter_|  around  inline
verbatim content,  until its special
meaning is revoked via
\UndefineShortVerb{\|}.
And  now the  |delimiter| no  longer
has any effect. A new one could also
be defined.
```

LaTeX Output

We can write `_inline verbatim_` by defining delimiting characters ad hoc, or via: And now we can use `_this_ _delimiter_` around inline verbatim content, until its special meaning is revoked via . And now the |delimiter| no longer has any effect. A new one could also be defined.

[1] `_And here we are, down in the footnotes!!_` This is outside the verbatim content in the footnote.

[2] Note that the underscores in this example are not necessary, but merely serve to emphasize that we are in verbatim mode, because in regular text mode it is a special character which would need to be escaped like so: `\_`.

3.3 Verbatim Environments

`fancyvrb` provides several verbatim environments. The most straightforward replacement for the stock LaTeX `verbatim` environment is the `Verbatim` environment[3]:

Code

```
\begin{Verbatim}
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
\end{Verbatim}
```

LaTeX Output

```
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
```

An example with a lot of options (not all of which might be sensible):

Code

```
\begin{Verbatim}[%
    gobble=8,
    numbers=left,
    numbersep=4pt,
    showspaces=true,
    showtabs=true,
    baselinestretch=1.5,
    tabsize=4,
    frame=single,
    framerule=1mm,
    framesep=1em,
    rulecolor=\color{teal},
    label={%
        [\fbox{\Large{Hello World}}]%
        end of code}%
    ]
    #include <stdio.h>

    main() {
            print("Hello, world!\n");
    }
    \end{Verbatim}
```

LaTeX Output

Hello World

```
1  #include <stdio.h>
2
3  main() {
4      print("Hello, world!\n");
5  }
```

end of code

We can also define custom environments:

[3] For a complete list of options, consult the user manual at `fancyvrb`.

Code

```
\DefineVerbatimEnvironment%
    {CustomVerbatim}{Verbatim}{%
    gobble=8,
    numbers=left,
    numbersep=4pt,
    baselinestretch=1.1,
    frame=single,
    framerule=1mm,
    framesep=1em,
    rulecolor=\color{teal}}

\begin{CustomVerbatim}[%
    label={%
    [\fbox{\Large{Hello World In C}}]
    end of code}]
    #include <stdio.h>

    main() {
        print("Hello, world!\n");
    }
\end{CustomVerbatim}

\begin{CustomVerbatim}[%
  label={%
  [\fbox{\Large{Hello World in Perl}}]
  end of code}]
    #!/usr/bin/env perl
    use strict;
    use warnings;
    use v5.10;

    say("Hello, world!");
\end{CustomVerbatim}
```

LaTeX Output

Hello World In C

```
1  #include <stdio.h>
2
3  main() {
4      print("Hello, world!\n");
5  }
```

end of code

Hello World in Perl

```
1  #!/usr/bin/env perl
2  use strict;
3  use warnings;
4  use v5.10;
5
6  say("Hello, world!");
```

end of code

There is also the possibility to store and recall verbatim content, which, as in the case of `verbatimbox` package, allows to use verbatim content where it is otherwise not allowed, and to re-use it repeatedly in a convenient way. Additionally, content can be written to and read from external files. Lastly, it shall be mentioned that the `fancyvrb` package can interface with the `listings` package (see Chapter 5) for code formatting.

4 The `listing` Package [4]

The **listing** package is not itself made for formatting code, but provides a new environment **listing** similar to **figure** and **table**, although not floating, which can be captioned and indexed.

Note that other packages provide this functionality as well, and there may be clashes (for example with the **minted** package, which provides an identically named environment and index list)[4].

A piece of example code, using the **fancyvrb** package for formatting the code, might look like this[5]:

Code

```
\begin{listing}
    \begin{GlobalCustomVerbatim}[label={\fbox{\Large{Hello World in Python}}}]
        #!/usr/bin/env python3

        print("Hello, world!")
    \end{GlobalCustomVerbatim}
\caption{This is a listing with a caption.}
\end{listing}
```

Which will yield as its output[6]:

Hello World in Python

```
1 #!/usr/bin/env python3
2
3 print("Hello, world!")
```

Listing 4.1: This is a listing with a caption.

The List of Listings can be generated at any place in the document by[7]:

Code

```
\listoflistings
```

LaTeX Output

List of Listings

[4] If you have a look at the code of this document, you will see that some workaround magic has been applied to the source code of the **listing** package to produce a customized version, renaming some crucial commands so that they do not clash with the other packages which have been loaded.

[5] The **GlobalCustomVerbatim** environment is similar to the **CustomVerbatim** environment defined above. The curious reader may feel inclined to look at this document's source code.

[6] The astute reader will notice that, for once, the output of this code is not put inside one of those fancy boxes with round corners. This is because the **listing** environment leaves what is called *outer par mode*, and thus it will refuse to work inside one of those boxes.

[7] The LoL also picks up a listing from the **listings** package from later in this document, despite the **listings** package using **lstlistoflistings** for its LoL, because life is weird sometimes.

5 The `listings` Package [5]

The `listings` package is a very powerful package for formatting code, and probably one of the more popular choices[8]. It supports inline code, code segments in paragraphs and floats and code input from external files. Covering the entirety of its capabilities would be far beyond the scope of this document; we shall focus on a few concise examples for common use cases. For more information, as you may have guessed, consult the package documentation at [5].

Code

```
set{language=C}
e writing normal text, code
be inlined with the listings
age: \lstinline!int i = 5;!.
```

LaTeX Output

While writing normal text, code can be inlined with the listings package: `int i = 5;`.

Without doing much, the output will look like this:

Code

```
\begin{lstlisting}[%
    language=C,
    gobble=8]
    #include <stdio.h>

    /* Comment required. */

    main() {
        print("Hello, world!\n");
        int a;
        a = 3;
        if (a = 2) {
            magic(a);
        } else {
            darkmagic(--a);
        }
    }
\end{lstlisting}
```

LaTeX Output

```
#include <stdio.h>

/* Comment required. */

main() {
    print("Hello, world!\n");
    int a;
    a = 3;
    if (a = 2) {
        magic(a);
    } else {
        darkmagic(--a);
    }
}
```

[8] Going by my very scientifically reliable gut feeling.

But `listings` allows a lot of customization:

Code

```
\lstdefinestyle{myC}{
    language          = C,
    showstringspaces  = false,
    basicstyle        = \ttfamily,
    numbers           = left,
    numberstyle       = \tiny,
    numbersep         = 4pt,
    commentstyle      = \itshape,
    keywordstyle      = \color{blue},
    stringstyle    = \color{magenta},
    identifierstyle = \color{violet},
    breaklines        = true,
    breakatwhitespace = true,
    tabsize           = 4
}
\begin{lstlisting}[%
    language=C,
    style=myC,
    gobble=8]
    #include <stdio.h>

    /* Comment required. */

    main() {
        print("Hello, world!\n");
        int a;
        a = 3;
        if (a = 2) {
            magic(a);
        } else {
            darkmagic(--a);
        }
    }
\end{lstlisting}
```

LaTeX Output

```
1  #include <stdio.h>
2
3  /* Comment required. */
4
5  main() {
6      print("Hello, world!\n");
7      int a;
8      a = 3;
9      if (a = 2) {
10         magic(a);
11     } else {
12         darkmagic(--a);
13     }
14 }
```

Note that we only had to define `keywordstyle`, `commentstyle` and `stringstyle` here, and `listings` automatically chose which words to highlight in which manner. A list of supported languages is in the manual at [5]; additional languages and new styles for existing languages (as done here) can be defined if they do not exist or if the existing ones do not suit your tastes. Adding new languages will obviously require you to define the appropriate key words and the way in which they are to be highlighted.

Including external files is done with the `\lstinputlisting` command. We can also give a caption (regardless of whether or not the listing is in a float) or, if we don't want the **Listing** text before the description, a `title` argument. Putting a caption also puts the listing in the list of listings (invoked by the `\lstlistoflistings` command).

Code — LaTeX Output

```
\lstinputlisting[
    caption = The \emph{Hello World} program in C,
    frame=lines,
    language=C,
    style=myC]{code/hw.c}
```

Listing 5.1: The *Hello World* program in C

```
1 #include <stdio.h>
2
3 main() {
4     print("Hello, world!\n");
5 }
```

A particularly nice feature for didactic purposes is the `escapeinside` argument, which allows to escape to LaTeX code inside an `lstlistings` environment.

Code — LaTeX Output

```
\begin{lstlisting}[%
    escapeinside=||,
    basicstyle = \ttfamily,
    gobble=8,
    frame=single,
    frameround=tttt,
    title = Example of \texttt{escapeinside}]
    By using the escapeinside argument, we can |\textcolor{red}{highlight}|
    particular sections of a code fragment with |\LaTeX| and
    use arbitrary |\LaTeX| commands in a listing.
\end{lstlisting}
```

Example of `escapeinside`

```
By using the escapeinside argument, we can highlight
particular sections of a code fragment with LaTeX and
use arbitrary LaTeX commands in a listing.
```

Listings can also be made floating by adding the `float` key to its optional arguments. Having a look at its manual is highly recommended, as it is a very configurable package.

6 The `listingsutf8` Package [6]

If you need to do a code listing for content with multi-byte characters, the **listingsutf8** package might be of use to you. It is based on the **listings** package, but pre-processes a multibyte-char file before passing it to **listings**. It can be loaded in the preamble either *after* or *instead of* the **listings** package.

Note that **listingsutf8** is limited to the **\lstinputlisting** command; no replacement is supported for the **lstlisting** environment, the **\lstinline** command, or custom environments defined via **\lstnewenvironment**.

7 The `matlab-prettifier` Package [7]

The **matlab-prettifier** package builds on top of the **listings** package and, as the name suggests, specializes in replicating as closely as possible the syntax-highlighting of the Matlab editor.

Code

```
\lstinputlisting[style=Matlab-editor]{code/fourier.m}
```

LaTeX Output

```
% ------------------------------------------------------------ %
% This  is  a  manual  implementation  of  the  DFT  for %
% educational purposes.  No sane  person should  ever use %
% this code for production purposes (much too slow).       %
%                                                          %
% (c) 2017 Raphael Frey, webmaster@alpenwasser.net         %
% ------------------------------------------------------------ %
clear all;close all;clc;
set(0,'DefaultFigureWindowStyle','docked');

L = 1e3;           % number of samples in time domain
tmax = 2*pi;       % maximum value of time axis
t = 0:2*pi/L:tmax; % time vector
K = 10;            % number of harmonics
f = 1;             % base frequency of signal in Hertz

% Generate square wave signal with K harmonics
x = 0;
for k = 1:K
    component = 4/pi*1/(2*k-1) * sin(2*pi*(2*k-1)*f*t);
    x = x + component;
end

% Discrete Fourier Transform of said signal
X = zeros(1,L);
for m = 0:L-1
    for n = 0:L-1
        X(m+1) = X(m+1) + x(n+1) * exp(-j*2*pi*m*n/L);
    end
end
```

Code

```
\begin{lstlisting}[gobble=8,style=Matlab-bw]
    % Discrete Fourier Transform of said signal
    title = 'DFT';
    X = zeros(1,L);
    for m = 0:L-1
        for n = 0:L-1
            X(m+1) = X(m+1) + x(n+1) * exp(-j*2*pi*m*n/L);
        end
    end
\end{lstlisting}
```

LaTeX Output

```
% Discrete Fourier Transform of said signal
title = 'DFT';
X = zeros(1,L);
for m = 0:L-1
    for n = 0:L-1
        X(m+1) = X(m+1) + x(n+1) * exp(-j*2*pi*m*n/L);
    end
end
```

Code

```
Inline code can be highlighted as well like this:
\lstinline[style=Matlab-pyglike]!break!. Since this is rather cumbersome,
particularly if used frequently, one may choose to define a shorthand via
the \verb|\lstMakeShortInline| mechanism from the \verb|listings| package:

\lstMakeShortInline[style=Matlab-pyglike]`

And then we can typeset `break` like this. We can undefine the shorthand.

\lstDeleteShortInline`

And now `this` does notterhing particularly special anymore. Obviously, it
is recommended to pick a character as the delimiter which does not clash
with the rest of your document or \LaTeX's inner workings.
```

LaTeX Output

Inline code can be highlighted as well like this: `break`. Since this is rather cumbersome, particularly if used frequently, one may choose to define a shorthand via the `\lstMakeShortInline` mechanism from the `listings` package:
And then we can typeset `break` like this. We can undefine the shorthand.
And now ‘this‘ does notterhing particularly special anymore. Obviously, it is recommended to pick a character as the delimiter which does not clash with the rest of your document or LaTeX’s inner workings.

8 The `minted` Package [8]

The **minted** package outsources the actual syntax highlighting to the powerful Python library Pygments [9], which supports over 300 languages and a lot of styles. Obviously, this means you must have a reasonably modern Python installation which includes Pygments.

Assuming the prerequisites are fulfilled, there is one more small but important thing: In order for Pygments to work its magic, LaTeX must be allowed to call it. In most setups, this will be disabled by default (that is: LaTeX is not allowed to execute any external commands on your computer). When compiling your LaTeX document, enable this via the **-shell-escape** switch[9]:

```
$ pdflatex -shell-escape your-document-with-minted-stuff-in-it
```

The primary downside of **minted** is that because it makes an external program call for every piece of content it should typeset, it is comparatively slow on first execution. Once it has parsed the content, and as long as said content does not change, it stores it in a cache directory (**_minted-<your-filename>** by default) and compile times should fall to reasonable levels again[10].

The standard environment is called **minted**. It accepts optional parameters inside the usual brackets, and requires an argument for the language as which it is supposed to parse its input. If you do not wish to have colored highlighting, you can pass the **text** option as the language.

Code

```
\begin{minted}[gobble=4]{C}
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
\end{minted}
```

LaTeX Output

```
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
```

[9] Similarly:
`$ xelatex -shell-escape your-document-with-minted-stuff-in-it`
and
`$ lualatex -shell-escape your-document-with-minted-stuff-in-it`,
respectively.

[10] As an example: Compiling this document for the first time takes about 30 seconds on my laptop; compiling it after **minited** has created its cache directory takes about 1.3 seconds.

Including files is done via the `\inputminted` command:

Code — LaTeX Output

```
    \inputminted[%
        linenos=true,
        numbersep=4pt,
        obeytabs,
        showtabs,
        style=borland,
        ]{C}{code/fib.c}
```

```c
#include <stdio.h>
#include <stdlib.h>

/* Calculates Fibonacci Sequence up to seq_length */

int fib(int);

int main ( int argc, char *argv[] )
{
        if ( argc != 2 )
        {
                printf( "usage: %s <positive integer>\n", argv[0] );
                return 0;
        } else {
                int seq_length;
                int current_elem;
                int current_fib;
                current_elem = -1;
                seq_length = atoi(argv[1]);
                while (seq_length > ++current_elem)
                {
                        printf(
                                "Fibonacci Number %d is %d\n",
                                current_elem + 1,
                                fib(current_elem));
                }
                return 0;
        }
}


int fib(int n)
{
        if ( n == 0 )
                return 0;
        else if ( n == 1 )
                return 1;
        else
                return ( fib(n-1) + fib(n-2) );
}
```

Code

```
Setting inline content is done with the mintinline command, so you can
typeset \mintinline{Python}{print("Hello, world!")} code like this.
```

LaTeX Output

Setting inline content is done with the mintinline command, so you can typeset `print("Hello, world!")` code like this.

It is also possible to escape to LaTeX inside a `minted` environment. First, the escape characters have to be defined as an optional argument. Note that this command can be a bit fragile at times and sometimes breaks. So if you get weird behavior while using this option, you can try to redefine the delimiting characters or find some other workaround.

Code

```
\begin{minted}[autogobble,escapeinside=||,style=bw]{C}
    #include <stdio.h>

    main() {
        |\textcolor{red}{print}|("Hello, world!\n");
    }
\end{minted}
```

LaTeX Output

```
#include <stdio.h>

main() {
    print("Hello, world!\n");
}
```

There are many more options for `minted` and the documentation is accordingly expansive.

9 The `tcolorbox` Package [10]

In case you've been wondering how those nice boxes which display both code and output have been made throughout this document, but are scared to look at the source code, that would be the `tcolorbox` package.

It is not a package strictly made for typesetting code, but, as the name suggests, is concerned with making boxes. It can, however, interface directly with the `listings`, `listingsutf8` and the `minted` package for formatting code. For more information, one may choose to peruse its *very* thorough manual[11].

10 References

[1] Rainer Schöpf and The LaTeX Team. "*verbatim -– Reimplementation of and extensions to LaTeX verbatim*", Version 1.5q, 2001-MAR-12. [Online], http://ctan.org/pkg/verbatim, [Accessed: 2017-MAR-22].

[2] Steven B. Segletes. "*verbatimbox – Deposit verbatim text in a box*", Version 3.13, 2014-MAR-12. [Online], http://ctan.org/pkg/verbatimbox, [Accessed: 2017-MAR-22].

[3] Timothy Van Zandt and Herbert Voß and Denis Girou. "*fancyrvb – Sophisticated verbatim text*", Version 2.8, 2010-MAY-15. [Online], http://ctan.org/pkg/fancyvrb, [Accessed: 2017-MAR-22].

[4] Volker Kuhlmann and Matthew Hebley. "*listing – Produce formatted program listings*", Version 1.2, 1999-MAY-25. [Online], http://ctan.org/pkg/listing, [Accessed: 2017-MAR-22].

[5] Brooks Moses and Carsten Heinz and Jobst Hoffman. "*listings – Typeset source code listings using LaTeX*", Version 1.6, 2015-JUN-04. [Online], http://ctan.org/pkg/listings, [Accessed: 2017-MAR-22].

[6] Heiko Oberdiek. "*listingsutf8 – Allow UTF-8 in listings input*", Version 1.3, 2016-MAY-16. [Online], http://ctan.org/pkg/listingsutf8, [Accessed: 2017-MAR-22].

[7] Julien Cretel. "*matlab-prettifier – Pretty-print Matlab source code*", Version 0.3, 2014-JUN-19. [Online], http://ctan.org/pkg/matlab-prettifier, [Accessed: 2017-MAR-22].

[8] Konrad Rudolph and Geoffrey Poore. "*minted – Highlighted source code for LaTeX*", Version 2.4.1, 2016-OCT-31. [Online], http://ctan.org/pkg/minted, [Accessed: 2017-MAR-22].

[9] Georg Brandl and Pygments Contributors. "*Pygments – Python Syntax Highlighter*". [Online], http://pygments.org, [Accessed: 2017-MAR-24].

[10] Thomas F. Sturm. "*tcolorbox – Coloured boxes, for LaTeX examples and theorems, etc*", Version 4.02, 2017-FEB-24. [Online], http://www.ctan.org/pkg/tcolorbox, [Accessed: 2017-MAR-24].

[11] Comprehensive TeX Archive Network. "*Topic listing*". [Online], http://ctan.org/topic/listing, [Accessed: 2017-MAR-22].

[12] Comprehensive TeX Archive Network. "*Topic verbatim*". [Online], http://ctan.org/topic/verbatim, [Accessed: 2017-MAR-22].

[11] All 478 pages of it, as of March 25, 2017.