

LATEX ENVIRONMENTS



TO IMAGE FORMAT

v1.8 — 2020-07-24^{*}

©2013–2020 by Pablo González L[†]

CTAN: <https://www.ctan.org/pkg/ltximg>

 <https://github.com/pablgonz/ltximg>

Abstract

`ltximg` is a `perl` script that automates the process of extracting and converting environments provided by `tikz`, `pstricks` and other packages from *input file* to image formats and standalone files using `ghostscript` and `poppler-utils`. Generates a file with only extracted environments and another with all extracted environments converted to `\includegraphics`.

Contents

1	License	1	8	Extract content	6
2	Motivation and Acknowledgments	1	8.1	Default environments	6
3	How it works	2	8.2	Extract whit docstrip tags	7
4	Requirements for operation	2	8.3	Prevent extraction and remove	7
5	The input file	2	9	Image Formats	7
6	Verbatim contents	3	10	How to use	8
6.1	Verbatim in line	3	10.1	Syntax	8
6.2	Verbatim standard	3	10.2	Command line interface	8
6.3	Verbatim write	4	10.3	Passing options from command line	10
7	Steps process	4	10.4	Options from input file	11
7.1	Validating Options	4	10.5	Passing options from input file	11
7.2	Comment and ignore	5	11	The way of arara	11
7.3	Creating files and extracting	5	12	Note for dvisvgm users	12
7.4	Generate image formats	5	13	Using arara and dvisvgm	13
7.5	Create output file	5	14	Final notes	14
7.6	Clean temporary files and dirs	6	15	Change history	14
			16	References	16
			17	Index of Documentation	17

1 License

This program 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 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.

2 Motivation and Acknowledgments

The original idea was to extend the functionality of the `pst2pdf`[9] script to work with `tikzpicture` and other environments. The `tikz`[2] package allows to *externalize* the environments, but, the idea was to be able to extend this to *any type* of environment covering three central points:

1. Generate a separate image files for environments.
2. Generate a standalone files with only the extracted environments.
3. Generate a file replacing the environments by `\includegraphics`.

^{*}This file describes a documentation for version 1.8, last revised 2020-07-24.

[†]E-mail: pablgonz@yahoo.com

From the side of T_EX there are some packages that cover several of these points such as the [preview](#)[1], [xcomment](#)[12], [extract](#)[13] and [cachepic](#)[14] packages among others, but none covered all points.

In the network there are some solutions in bash that were able to extract and convert environments, but in general they presented problems when the document contained “*verbatim style*” code or were only available for [Linux](#).

Analysed the situation the best thing was to create a new “*script*” that was able to cover the three points and was multi platform, the union of all these ideas is born [ltximg](#).

This script would not be possible without the great work of Herbert Voß author of [pst2pdf](#)¹ and Heiko Oberdiek author of [pdfcrop](#)². Several parts of the code have been taken and adapted from both scripts.

3 How it works

It is important to have a general idea of how the “*extraction and conversion*” process works and the requirements that must be fulfilled so that everything works correctly, for this we must be clear about some concepts related to how to work with the [input file](#), the [verbatim content](#) and the [steps process](#).

4 Requirements for operation

For the complete operation of [ltximg](#) you need to have a modern T_EX distribution such as T_EXLive or MiK_TE_X, have a version equal to or greater than 5.28 of [perl](#), a version equal to or greater than 9.24 of [ghostscript](#) and have a version equal to or greater than 0.52 of [poppler-utils](#).

The distribution of T_EXLive 2020 for [Windows](#) includes [ltximg](#) and all requirements, MiK_TE_X users must install the appropriate software for full operation.

The script has been tested on [Windows](#) (v10) and [Linux](#) (fedora 32) in x64 architecture using [ghostscript](#) v9.52, [poppler-utils](#) v0.84, [perl](#) v5.30 and the standard classes offers by L^AT_EX: [book](#), [report](#), [article](#) and [letter](#). If an [output file](#) is generated, the [graphicx](#)[10] and [grfext](#)[11] packages will be needed.

5 The input file

The [input file](#) must comply with certain characteristics in order to be processed, the content at the beginning and at the end of the [input file](#) is treated in a special way, before [\documentclass](#) and after [\end{document}](#) can go any type of content, internally the script will “*split*” the [input file](#) at this points.

If the [input file](#) contains files using [\input{file}](#) or [\include{file}](#) these will not be processed, from the side of the *script* they only represent lines within the file, if you want them to be processed it is better to use the [latexexpand](#)³ first and then process the file.

Like [\input{file}](#) or [\include{file}](#), blank lines, vertical spaces and tab characters are treated literally, for the *script* the [input file](#) is just a set of characters, as if it was a simple text file. It is advisable to format the source code [input file](#) using utilities such as [chktex](#)⁴ and [latexindent](#)⁵, especially if you want to extract the source code of the environments.

Both [\thispagestyle{style}](#) and [\pagestyle{style}](#) are treated in a special way by the script, if they do not appear in the preamble then [\pagestyle{empty}](#) will be added and if they are present and [{style}](#) is different from [{empty}](#) this will be replaced by [{empty}](#).

This is necessary for the image creation process, it does not affect the [output file](#), but it does affect the *standalone* files.

For the script the process of dividing the [input file](#) into four parts and then processing them:

```

1 % Part One: Everything before \documentclass
2 \documentclass{article}
3 % Part two: Everything between \documentclass and \begin{document}
4 \begin{document}
5 % Part three: : Everything between \begin{document} and \end{document}
6 \end{document}
7 % Part Four: Everything after \end{document}

```

If for some reason you have an environment [filecontents](#) before [\documentclass](#) or in the preamble of the [input file](#) that contains a *sub-document* or *environment* you want to extract, the script will ignore them.

¹<https://ctan.org/pkg/pst2pdf>

²<https://ctan.org/pkg/pdfcrop>

³<https://www.ctan.org/pkg/latexexpand>

⁴<https://www.ctan.org/pkg/chktex>

⁵<https://www.ctan.org/pkg/latexindent>

6 Verbatim contents

One of the greatest capabilities of this script is to “skip” the complications that *verbatim content* produces with the extraction of environments using tools outside the “T_EX world”⁶. In order to “skip” the complications, the *verbatim content* is classified into three types:

- Verbatim in line.
- Verbatim standard.
- Verbatim write.

6.1 Verbatim in line

The small pieces of code written using a “verbatim macro” are considered *verbatim in line*, such as `\verb|code|` or `\verb*|code|` or `\macro{code}` or `\macro[opts]{code}`.

Most “verbatim macro” provide by packages `minted`[18], `fancyvrb`[16] and `listings`[17] have been tested and are fully supported. They are automatically detected the *verbatim macro* (including *** argument) generates by `\newmint` and `\newmintinline` and the following list:

- | | | |
|------------------------|---------------------------|----------------------------|
| • <code>\mint</code> | • <code>\verb</code> | • <code>\pygment</code> |
| • <code>\spverb</code> | • <code>\Verb</code> | • <code>\Scontents</code> |
| • <code>\qverb</code> | • <code>\lstinline</code> | • <code>\tcboxverb</code> |
| • <code>\fverb</code> | • <code>\pyginline</code> | • <code>\mintinline</code> |

Some packages define abbreviated versions for “verbatim macro” as `\DefineShortVerb`, `\lstMakeShortInline` and `\MakeSpecialShortVerb`, will be detected automatically if are declared explicitly in *input file*.

The following consideration should be kept in mind for some packages that use abbreviations for verbatim macros, such as `shortvrb`[15] or `doc`[15] for example in which there is no explicit `\macro` in the document by means of which the abbreviated form can be detected, for automatic detection need to find `\DefineShortVerb` explicitly to process it correctly. The solution is quite simple, just add in *input file*:

```
\UndefinedShortVerb{\|}
\DefineShortVerb{\|}
```

depending on the package you are using. If your “verbatim macro” is not supported by default or can not detect, use the options described in 10.2 and 10.4.

6.2 Verbatim standard

These are the “classic” environments for “writing code” are considered *verbatim standard*, such as `verbatim` and `lstlisting` environments. The following list (including *** argument) is considered as *verbatim standard* environments:

- | | | | |
|-----------------------------------|-----------------------------|------------------------------|--------------------------|
| • <code>Example</code> | • <code>SaveVerbatim</code> | • <code>comment</code> | • <code>pyglist</code> |
| • <code>CenterExample</code> | • <code>PSTcode</code> | • <code>chklisting</code> | • <code>program</code> |
| • <code>SideBySideExample</code> | • <code>LTXexample</code> | • <code>verbatimtab</code> | • <code>programl</code> |
| • <code>PcenterExample</code> | • <code>tcblisting</code> | • <code>listingcont</code> | • <code>programL</code> |
| • <code>PSideBySideExample</code> | • <code>spverbatim</code> | • <code>boxedverbatim</code> | • <code>programs</code> |
| • <code>verbatim</code> | • <code>minted</code> | • <code>demo</code> | • <code>programf</code> |
| • <code>Verbatim</code> | • <code>listing</code> | • <code>sourcecode</code> | • <code>programsc</code> |
| • <code>BVerbatim</code> | • <code>lstlisting</code> | • <code>xcomment</code> | • <code>programt</code> |
| • <code>LVerbatim</code> | • <code>alltt</code> | • <code>pygmented</code> | |

They are automatically detected *verbatim standard* environments (including *** argument) generates by commands:

- | | |
|---|--------------------------------|
| • <code>\DefineVerbatimEnvironment</code> | • <code>\includecomment</code> |
| • <code>\NewListingEnvironment</code> | • <code>\newtcblisting</code> |
| • <code>\DeclareTCBListing</code> | • <code>\NewTCBListing</code> |
| • <code>\ProvideTCBListing</code> | • <code>\newverbatim</code> |
| • <code>\lstnewenvironment</code> | • <code>\NewProgram</code> |
| • <code>\newtabverbatim</code> | • <code>\newminted</code> |
| • <code>\specialcomment</code> | |

If any of the *verbatim standard* environments is not supported by default or can not detected, you can use the options described in 10.2 and 10.4.

⁶Only T_EX can understand T_EX, all other languages and programs are just lines in a file.

6.3 Verbatim write

Some environments have the ability to write “external files” or “store content” in memory, these environments are considered *verbatim write*, such as `scontents`, `filecontents` or `VerbatimOut` environments. The following list is considered (including *** argument) as *verbatim write* environments:

- `scontents`
- `filecontents`
- `tcboutputlisting`
- `tcbexternal`
- `tcbwritetmp`
- `extcolorbox`
- `exttikzpicture`
- `VerbatimOut`
- `verbatimwrite`
- `filecontentsdef`
- `filecontentshere`
- `filecontentsdefmacro`
- `filecontentsdefstarred`
- `filecontentsgdef`
- `filecontentsdefmacro`
- `filecontentsgdefmacro`

They are automatically detected *verbatim write* (including *** argument) environments generates by commands:

- `\renewtcbexternalizetcolorbox`
- `\renewtcbexternalizeenvironment`
- `\newtcbexternalizeenvironment`
- `\newtcbexternalizetcolorbox`
- `\newenvsc`

If any of the *verbatim write* environments is not supported by default or can not detected, you can use the options described in 10.2 and 10.4.

7 Steps process

For creation of the image formats, extraction of source code of environments and creation of an *output file*, `ltximg` need a various steps. Let’s assume that the *input file* is `test.tex`, *output file* is `test-out.tex`, the working directory are `.`, the directory for images are `./images`, the temporary directory is `/tmp` and we want to generate images in `pdf` format and *standalone files* with the source code of the environments.

We will use the following code as `test.tex`

```

1 % Some commented lines at begin file
2 \documentclass{article}
3 \usepackage{tikz}
4 \begin{document}
5 Some text
6 \begin{tikzpicture}
7   Some code
8 \end{tikzpicture}
9 Always use \verb|\begin{tikzpicture}| and \verb|\end{tikzpicture}| to open
10 and close environment
11 \begin{tikzpicture}
12   Some code
13 \end{tikzpicture}
14 Some text
15 \begin{verbatim}
16 \begin{tikzpicture}
17   Some code
18 \end{tikzpicture}
19 \end{verbatim}
20 Some text
21 \end{document}
22 Some lines that will be ignored by the script

```

7.1 Validating Options

The first step is read and validated [*options*] from the command line and `test.tex`, verifying that `test.tex` contains *some* environment to extract, check the name and extension of `test-out.tex`, check the directory `./images` if it doesn’t exist create it and create a temporary directory `/tmp/hG45uVklv9`.

The entire `test.tex` file is loaded into memory and split to start the extraction process.

7.2 Comment and ignore

In the second step, once the file `test.tex` is loaded and divided in memory, proceeds (in general terms) as follows:

Search the words `\begin{` and `\end{` in verbatim standard, verbatim write, verbatim in line and commented lines, if it finds them, converts to `\BEGIN{` and `\END{`, then places all code to extract inside the `\begin{preview} ... \end{preview}`.

At this point “all” the code you want to extract is inside `\begin{preview} ... \end{preview}`.

7.3 Creating files and extracting

In the third step, the script generate *(standalone files)* `test-fig-1.tex`, `test-fig-2.tex` and saved in `./images`. A temporary file with a random number (1981 for example) with all environments is created and proceed in two ways according to the `[options]` passed to the script:

1. If script is call `whitout -n, --noprew` options, adds the following lines to the beginning of the `test.tex` (in memory):

```
\AtBeginDocument{%
\RequirePackage[active,tightpage]{preview}
\renewcommand\PreviewBbAdjust{-60pt -60pt 60pt 60pt}}%
% rest of input file
```

And save in a temporary file `test-fig-1981.tex` in `./`.

2. If script is call `whit -n, --noprew` options, the `\begin{preview} ... \end{preview}` lines are only used as delimiters for extracting the content *without* using the package `preview`.

Creating a temporary file `test-fig-1981.tex` in `./` whit the same preamble of `test.tex`, but the *(body)* only contains code that you want to extract.

If `--norun` is passed, the temporary file `test-fig-1981.tex` is renamed to `test-fig-all.tex` and moved to `./images`.

7.4 Generate image formats

In the fourth step, the script run:

```
[user@machine ~:]$ <compiler> -recorder -shell-escape test-fig-1981.tex
```

generating the file `test-fig-1981.pdf` whit all code extracted, move `test-fig-1981.pdf` to `/tmp/hG45uVklv9` and rename to `test-fig-all.pdf`, separate in individual files `test-fig-1.pdf` and `test-fig-2.pdf` and copy to `./images`. The file `test-fig-1981.tex` is moved to the `./images` and rename to `test-fig-all.tex`.

Note the options passed to `<compiler>` does not include `-output-directory` (it is not supported) and always use `-recorder -shell-escape`.

7.5 Create output file

In the fifth step, the script creates the output file `test-out.tex` converting all extracted code to `\includegraphics` and adding the following lines at end of preamble:

```
1 \usepackage{graphicx}
2 \graphicspath{{images/}}
3 \usepackage{grfext}
4 \PrependGraphicsExtensions*{.pdf}
```

The script will try to detect whether the `graphicx` package and the `\graphicspath` command are in the preamble of the *(output file)*. If it is not possible to find it, it will read the log file generated by the temporary file. Once the detection is complete, the package `grfext` and `\PrependGraphicsExtensions*` will be added at the end of the preamble, then proceed to run:

```
[user@machine ~:]$ <compiler> -recorder -shell-escape test-out.tex
```

generating the file `test-out.pdf`.

7.6 Clean temporary files and dirs

In the sixth step, the script read the files `test-fig-1981.flx` and `test-out.flx`, extract the information from the temporary files and dirs generated in the process in `./` and then delete them together with the directory `/tmp/hG45uVklv9`.

Finally the output file `test-out.tex` looks like this:

```

1 % some commented lines at begin document
2 \documentclass{article}
3 \usepackage{tikz}
4 \graphicspath{{images/}}
5 \usepackage{grfext}
6 \PrependGraphicsExtensions*{.pdf}
7 \begin{document}
8 Some text
9 \includegraphics[scale=1]{test-fig-1}
10 Always use \verb|\begin{tikzpicture}| and \verb|\end{tikzpicture}| to open
11 and close environment
12 \includegraphics[scale=1]{test-fig-2}
13 Some text
14 \begin{verbatim}
15 \begin{tikzpicture}
16   Some code
17 \end{tikzpicture}
18 \end{verbatim}
19 Some text
20 \end{document}
```

8 Extract content

The script provides two ways to *extract* content from *input file*, using *environments* and *docstrip tags*. Some environment (including `*` argument) are supported by default. If environments are nested, the outermost one will be extracted.

8.1 Default environments

<code>\begin{preview}</code> <i>env content</i> <code>\end{preview}</code>	Environment provide by <code>preview[1]</code> package. If <code>preview</code> environments found in the <i>input file</i> will be extracted and converted these. Internally the script converts all environments to extract in <code>preview</code> environments. Is better comment this package in preamble unless the option <code>-n,--noprew</code> is used.
<code>\begin{pspicture}</code> <i>env content</i> <code>\end{pspicture}</code>	Environment provide by <code>pstricks[3]</code> package. The plain T _E X syntax <code>\pspicture ... \endpspicture</code> its converted to L ^A T _E X syntax <code>\begin{pspicture} ... \end{pspicture}</code> if not within the <code>PSTexample</code> environment.
<code>\begin{psgraph}</code> <i>env content</i> <code>\end{psgraph}</code>	Environment provide by <code>pst-plot[4]</code> package. The plain T _E X <code>\psgraph ... \endpsgraph</code> its converted to L ^A T _E X syntax <code>\begin{psgraph} ... \end{psgraph}</code> if not within the <code>PSTexample</code> environment.
<code>\begin{postscript}</code> <i>env content</i> <code>\end{postscript}</code>	Environment provide by <code>pst-pdf[5]</code> , <code>auto-pst-pdf[6]</code> and <code>auto-pst-pdf-lua[7]</code> packages. Since the <code>pst-pdf</code> and <code>auto-pst-pdf</code> packages internally use the <code>preview</code> package, is better comment this in preamble. This “encapsulation” environment is removed when processed by the script.
<code>\begin{tikzpicture}</code> <i>env content</i> <code>\end{tikzpicture}</code>	Environment provide by <code>tikz[2]</code> package. The plain T _E X <code>\tikzpicture ... \tikzpicture</code> its converted to L ^A T _E X syntax <code>\begin{tikzpicture} ... \end{tikzpicture}</code> but no a short syntax <code>\tikz ... ;</code> .
<code>\begin{pgfpicture}</code> <i>env content</i> <code>\end{pgfpicture}</code>	Environment provide by <code>pgf[2]</code> package. Since the script uses a “recursive regular expression” to extract environments, no presents problems if present <code>pgfinterruptpicture</code> .
<code>\begin{PSTexample}</code> <i>env content</i> <code>\end{PSTexample}</code>	Environment provide by <code>pst-exa[8]</code> packages. The script automatically detects the <code>\begin{PSTexample} ... \end{PSTexample}</code> environments and processes them as separately compiled files. The user should have loaded the package with the <code>[swpl]</code> or <code>[tcb]</code> option and run the script using <code>--latex</code> or <code>--xetex</code> .

If you need to extract other environments you can use one of the options described in 10.2 or 10.4.

8.2 Extract whit docstrip tags

`%<*ltximg>` All content included between `%<*ltximg>` ... `%</ltximg>` is extracted. The tags can *not* be nested and should be at the beginning of the line and in separate lines. Internally the script converts all this tags to `preview` environments.

```
% no space before open tag %<*
%<*ltximg>
code to extract
%</ltximg>
% no space before close tag %</
```

8.3 Prevent extraction and remove

Sometimes you do not want to “extract all” the environments from *(input file)* or you want to remove environments or arbitrary content. The script provides a convenient way to solve this situation.

`\begin{nopreview}` Environment provide by `preview` package. Internally the script converts all “skip” environments to `\begin{nopreview}` ... `\end{nopreview}`. Is better comment this package in preamble unless the option `-n,--noprew` is used.

`%<*noltximg>` All content between `%<*noltximg>` ... `%</noltximg>` are ignored and no extract. The tags can *not* be nested and should be at the beginning of the line and in separate lines. Internally the script converts all this tags to `nopreview` environments.

```
% no space before open tag %<*
%<*noltximg>
no extract this
%</noltximg>
% no space before close tag %</
```

`%<*remove>` All content between `%<*remove>` ... `%</remove>` are deleted in the *(output file)*. The tags can *not* be nested and should be at the beginning of the line and in separate lines.

```
% no space before open tag %<*
%<*remove>
lines removed in output file
%</remove>
% no space before close tag %</
```

The content will be deleted if it is “not” within a *(verbatim)* or *(verbatim write)* environment. If you want to remove specific environments automatically you can use one of the options described in 10.2 or 10.4.

9 Image Formats

The *(image formats)* generated by the `ltximg` using `ghostscript` and `poppler-utils` are the following command lines:

pdf The image format generated using `ghostscript`. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAfer -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress
```

eps The image format generated using `pdftops`. The line executed by the system is:

```
[user@machine ~:]$ pdftops -q -eps
```

png The image format generated using `ghostscript`. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAfer -sDEVICE=pngalpha -r 150
```

jpg The image format generated using `ghostscript`. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAfer -sDEVICE=jpeg -r 150 -dJPEGQ=100 \
-dGraphicsAlphaBits=4 -dTextAlphaBits=4
```

ppm The image format generated using `pdftoppm`. The line executed by the system is:

```
[user@machine ~:]$ pdftoppm -q -r 150
```


tif The image format generated using `ghostscript`. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAfer -sDEVICE=tiff32nc -r 150
```

svg The image format generated using `pdftocairo`. The line executed by the system is:

```
[user@machine ~:]$ pdftocairo -q -r 150
```

bmp The image format generated using `ghostscript`. The line executed by the system is:

```
[user@machine ~:]$ gs -q -dNOSAfer -sDEVICE=bmp32b -r 150
```

The script auto detects the `ghostscript`, but not `poppler-utils`. You should keep this in mind if you are using the script directly and not the version provided in your T_EX distribution

10 How to use

10.1 Syntax

The syntax for `ltximg` is simple, if your use the version provided in your T_EX distribution:

```
[user@machine ~:]$ ltximg <compiler> [<options>] [--] <input file>
```

If the development version is used:

```
[user@machine ~:]$ perl ltximg <compiler> [<options>] [--] <input file>
```

The extension valid for *<input file>* are `.tex` or `.ltx`, relative or absolute paths for files and directories is not supported. If used without *<compiler>* and *[<options>]* the extracted environments are converted to `pdf` image format and saved in the `./images` directory using `pdflatex` and `preview` package.

10.2 Command line interface

The script provides a *command line interface* with short `-` and long `--` option, they may be given before the name of the *<input file>*, the order of specifying the options is not significant. Options that accept a *<value>* require either a blank space `␣` or `=` between the option and the *<value>*. Multiple short options can be bundling and if the last option takes a *<comma separated list>* you need `--` at the end.

<code>-h, --help</code>	<i><boolean></i>	(default: off)
	Display a command line help and exit.	
<code>-l, --log</code>	<i><boolean></i>	(default: off)
	Write a <code>.log</code> file with debug information.	
<code>-v, --version</code>	<i><boolean></i>	(default: off)
	Display the current version (1.8) and exit.	
<code>-V, --verbose</code>	<i><boolean></i>	(default: off)
	Show verbose information in terminal.	
<code>-d, --dpi</code>	<i><integer></i>	(default: 150)
	Dots per inch for images files. Values are positive integers less than or equal to 2500	
<code>-t, --tif</code>	<i><boolean></i>	(default: off)
	Create a <code>.tif</code> images files using <code>ghostscript</code> .	
<code>-b, --bmp</code>	<i><boolean></i>	(default: off)
	Create a <code>.bmp</code> images files using <code>ghostscript</code> .	
<code>-j, --jpg</code>	<i><boolean></i>	(default: off)
	Create a <code>.jpg</code> images files using <code>ghostscript</code> .	
<code>-p, --png</code>	<i><boolean></i>	(default: off)
	Create a <code>.png</code> transparent image files using <code>ghostscript</code> .	
<code>-e, --eps</code>	<i><boolean></i>	(default: off)
	Create a <code>.eps</code> image files using <code>pdftops</code> .	

-s, --svg	<i><boolean></i>	(default: off)
Create a .svg image files using pdftocairo .		
-P, --ppm	<i><boolean></i>	(default: off)
Create a .ppm image files using pdftoppm .		
-g, --gray	<i><boolean></i>	(default: off)
Create a gray scale for all images using ghostscript . The line behind this options is:		
<pre>[user@machine ~:]\$ gs -q -dNOSAfer -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress \ -sColorConversionStrategy=Gray -dProcessColorModel=/DeviceGray</pre>		
-f, --force	<i><boolean></i>	(default: off)
Try to capture \psset{<code>} and \tikzset{<code>} to extract. When using the --force option the script will try to capture \psset{<code>} or \tikzset{<code>} and leave it inside the preview environment, any line that is between \psset{<code>} and \begin{pspicture} or between \tikzset{<code>} and \begin{tikzpicture} will be captured.		
-n, --noprew	<i><boolean></i>	(default: off)
Create images files without preview package. The \begin{preview}...end{preview} lines are only used as delimiters for extracting the content <i>without</i> using the package preview . Sometimes it is better to use it together with --force .		
-m, --margins	<i><integer></i>	(default: 0)
Set margins in bp for pdfcrop .		
-r, --runs	<i><integer></i>	(default: 1)
Set the number of times the compiler will run on the <i><input file></i> for environment extraction. Possible values are 1, 2 or 3.		
-o, --output	<i><file name></i>	(default: empty)
Create <i><file name></i> whit all extracted environments converted to \includegraphics . Only <i><file name></i> must be passed <i>without</i> relative or absolute paths.		
--prefix	<i><string></i>	(default: fig)
Set <i><prefix></i> append to each generated files.		
--myverb	<i><macro name></i>	(default: myverb)
Set custom verbatim command \myverb . Just pass the <i><name></i> of the macro <i>without</i> “\”.		
--imgdir	<i><string></i>	(default: images)
Set the name of directory for save generated files. Only the name of directory must be passed <i>without</i> or absolute paths.		
--zip	<i><boolean></i>	(default: off)
Compress the files generated by the script in ./images in .zip format. Does not include <i><output file></i> .		
--tar	<i><boolean></i>	(default: off)
Compress the files generated by the script in ./images in .tar.gz format. Does not include <i><output file></i> .		
--srcenv	<i><boolean></i>	(default: off)
Create separate files whit “ <i>only code</i> ” for all extracted environments. This option is designed to generate <i><standalone files></i> with “ <i>only code</i> ” of the environments, is mutually exclusive whit --subenv .		
--subenv	<i><boolean></i>	(default: off)
Create sub files whit “ <i>preamble and code</i> ” for all extracted environments. This option is designed to generate <i><standalone compilable files></i> for each extracted environment, is mutually exclusive whit --srcenv .		
--norun	<i><boolean></i>	(default: off)
Execute the script, but do not create image files. This option is designed to be used in conjunction with --srcenv or --subenv and to debug the <i><output file></i> .		
--nopdf	<i><boolean></i>	(default: off)
Don't create a .pdf image files.		
--nocrop	<i><boolean></i>	(default: off)

	Don't run <code>pdfcrop</code> in image files.	
<code>--arara</code>	<i><boolean></i>	(default: off)
	Use <code>arara</code> for compiler files. See 11 for more information.	
<code>--xetex</code>	<i><boolean></i>	(default: off)
	Using <code>xelatex</code> compiler <i><input file></i> and <i><output file></i> .	
<code>--latex</code>	<i><boolean></i>	(default: off)
	Using <code>latex»dvips»ps2pdf</code> compiler in <i><input file></i> and <code>pdflatex</code> for <i><output file></i> .	
<code>--dvips</code>	<i><boolean></i>	(default: off)
	Using <code>latex»dvips»ps2pdf</code> for compiler <i><input file></i> and <i><output file></i> .	
<code>--dvi lua</code>	<i><boolean></i>	(default: off)
	Using <code>dvi lua latex»dvips»ps2pdf</code> for compiler <i><input file></i> and <code>lua latex</code> for <i><output file></i> .	
<code>--dvi pdf</code>	<i><boolean></i>	(default: off)
	Using <code>latex»dvipdfmx</code> for compiler <i><input file></i> and <i><output file></i> .	
<code>--latexmk</code>	<i><boolean></i>	(default: off)
	Using <code>latexmk</code> for process <i><output file></i> . Need a compiler option.	
<code>--luatex</code>	<i><boolean></i>	(default: off)
	Using <code>lua latex</code> for compiler <i><input file></i> and <i><output file></i> .	
<code>--clean</code>	<i><doc pst tkz all off></i>	(default: doc)
	Removes specific content in <i><output file></i> . Valid values for this option are:	
	<code>doc</code> All content after <code>\end{document}</code> is removed.	
	<code>pst</code> All <code>\psset{<code>}</code> and <code>pstricks</code> package is removed in <i><preamble></i> and <i><body></i> .	
	<code>tkz</code> All <code>\tikzset{<code>}</code> is removed in <i><body></i> .	
	<code>all</code> Activates <code>doc</code> , <code>pst</code> and <code>tkz</code> .	
	<code>off</code> Deactivate all.	
<code>--extenv</code>	<i><comma separated list></i>	(default: empty)
	Add environments to extract, if it's the last option passed need <code>--</code> at the end.	
<code>--skipenv</code>	<i><comma separated list></i>	(default: empty)
	Add environments that should “not be extracted” and that the script supports by default, if it's the last option passed need <code>--</code> at the end.	
<code>--verbenv</code>	<i><comma separated list></i>	(default: empty)
	Add <i><verbatim standard></i> environment support, if it's the last option passed need <code>--</code> at the end.	
<code>--writenv</code>	<i><comma separated list></i>	(default: empty)
	Add <i><verbatim write></i> environment support, if it's the last option passed need <code>--</code> at the end.	
<code>--deltenv</code>	<i><comma separated list></i>	(default: empty)
	Add environments to deleted in <i><output file></i> . The environments are delete only in <i><body></i> , if it's the last option passed need <code>--</code> at the end.	

10.3 Passing options from command line

An example of usage from command line:

```
[user@machine ~:]$ ltximg --latex -s -o test-out test-in.ltx
```

Create a `./images` directory (if it does not exist) whit all extracted environments converted to image formats (pdf, svg) in individual files, an *<output file>* `test-out.ltx` whit all extracted environments converted to `\includegraphics` and a single file `test-in-fig-all.ltx` with only the extracted environments using `latex»dvips»ps2pdf` and `preview` package for *<input file>* and `pdflatex` for *<output file>*.

10.4 Options from input file

Many of the ideas in this section are inspired by the [arara](#)⁷ tool. A very useful way to pass options to the script is to place them in commented lines at the beginning of the file, very much in the “style of [arara](#)”.

```
% ltximg: <argument>: {<option one, option two, option three, ...>}
```

```
%!ltximg: <argument>: {<option one, option two, option three, ...>}
```

The vast majority of the *<options>* can be passed into the *<input file>*. These should be put at the beginning of the file in commented lines and everything must be on the same line, the exclamation mark `!` deactivates the *<options>*. When passing options from the *<input file>* you should be aware that they must “*not*” contain `-` or `--`, the `=` sign between an option and its value is mandatory, short names are disabled and options found in the *<input file>* overwrite those passed on the command line.

Valid values for *<argument>* are the following:

```
% ltximg: options: {<option one = value, option two = value, option three = value, ...>}
```

This line is to indicate to the script which options need to process.

```
% ltximg: extrenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments, not supported by default, are extracted.

```
% ltximg: skipenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments, of the ones supported by default, should not be extracted.

```
% ltximg: verbenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments, its considerate a *<verbatim standard>*.

```
% ltximg: writenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments its consider *<verbatim write>*.

```
% ltximg: deltenv: {<environment one, environment two, environment three, ...>}
```

This line is to indicate to the script which environments are deleted.

If you are going to create an *<output file>* and you do not want these lines to remain, it is better to place them inside the `%<★remove> ... %</remove>`. Like this:

```
1 %<★remove>
2 % ltximg: options: { png, srcenv, xetex }
3 % ltximg: extrenv: { description }
4 %</remove>
```

10.5 Passing options from input file

Adding the following lines to the beginning of the file `file-in.tex`:

```
1 % ltximg: options: { luatex, output = file-out, imgdir = pics, prefix = env }
2 % ltximg: skipenv: { tikzpicture }
3 % ltximg: deltenv: { filecontents }
```

and run:

```
[user@machine~:]$ ltximg file-in.tex
```

Create a `./pics` directory (if it does not exist) whit all extracted environments, except `tikzpicture`, converted to image formats (pdf) in individual files, an *<output file>* `file-out.tex` whit all extracted environments converted to `\includegraphics` and environment `filecontents` removed, a single file `test-in-env-all.ltx` with only the extracted environments using `lualatex` and `preview` package for *<input file>* and *<output file>*.

11 The way of arara

By design, the script only runs “*one or more compilation*” on top of the *<input file>*, but, sometimes you need to process in a specific mode the *<input file>* or needs to be processed with something other than \LaTeX , \XeTeX , \pdfTeX or \LuaTeX engine.

This is where [arara](#)^[19] comes in, this “*great little tool*”, is able to have complete control over the compilation of the *<input file>*, we just have to keep a few considerations in mind:

⁷<https://ctan.org/pkg/arara>

1. Read the documentation (this always comes first).
2. Add `{ options: [-recorder] }` to “rule” for clean temporary files.
3. Avoiding the use of `: clean: { extensions: [...] }`.
4. Don’t set `-jobname` and `-output-directory` in any “rule”.

When the `--arara` option is passed to the script, the line that runs in the system is:

```
[user@machine~:]$ arara --log file.tex
```

If you have several “rules” within the file they will all be executed, to avoid this we must add:

```
1 % arara: halt
```

After the last “rule” you have at the beginning of the file. With all these considerations in mind it is possible to extract and convert environments from *any file*.

For example, by adding these lines at the beginning of the file:

```
1 % arara: luatex: { options: [-recorder] }
2 % arara: luatex: { options: [-recorder] }
3 %<*remove>
4 % ltximg: options: { arara, output = file-out, prefix = tkz}
5 %</remove>
```

and run:

```
[user@machine~:]$ ltximg test.tex
```

Create a `./images` directory (if it does not exist) whit all extracted environments converted to image format (pdf) in individual files, an `<output file>` `file-out.tex` whit all exatracted environments converted to `\includegraphics`, a single file `test-tkz-all.tex` with only the extracted environments using `preview` package and `luatex` “two times” for `<input file>` and `<output file>`.

Remember that the `<input file>` and `<output file>` will be compiled using the same “rule”. One *trick* to get around this situation is to use:

```
1 %<*remove>
2 % arara: luatex: { options: [-recorder] }
3 % arara: luatex: { options: [-recorder] }
4 % arara: halt
5 % ltximg: options: { arara, output = file-out, prefix = tkz}
6 %</remove>
7 % arara: xelatex: { options: [-recorder] }
8 % arara: xelatex: { options: [-recorder] }
```

The content between `%<*remove> ... %</remove>` are remove from `<output file>` before compiling. Thus, the `<output file>` will be compiled using `xelatex` “two times”.

As a final consideration, `ltximg` passes options to the `preview` package and the `pdfcrop` script according to the engine used. When using `--arara` it will “try” to detect the used engine by means of a regular expression, if the detection fails the default values will be used.

This does not affect the process of creating `<standalone files>` and can be prevented by using `--noprew` or `--nocrop` at the cost of not having the images cropped.

In this way we can `<compile>` and `<convert>` any document as long as the conditions of the `<input file>` are met and the correct “rule” are used.

12 Note for dvisvgm users

By design, the image format `svg` is created using `pdftocairo` over the generated `pdf` file, but, if you want to have a good `svg` file it is best to use `dvisvgm`⁸ which is included in every modern T_EX distribution. The best results of `dvisvgm`[20] are obtained when processing the file in `.dvi` or `.xdv` format, there are three possible ways to do this:

1. Pass the necessary options to `arara` and let him do the job⁹.
2. Execute the script using `--subenv` and `--norun` to generate `<standalone files>`, move to `./images` and generate `.dvi` or `.xdv` files runing:

```
[user@machine~:]$ for i in *.tex; do <compiler> [ <options> ] $i;done
```

⁸<https://ctan.org/pkg/dvisvgm>

⁹The `dvisvgm` rule in version 5.1.3 of `arara` only supports a `dvi` extention

then:

```
[user@machine~:]$ for i in *.dvi; do dvisvgm [options] $i;done
```

3. Execute the script using `--norun`, move to `./images` and generate `.dvi` or `.xdv` file running:

```
[user@machine~:]$ compiler [options] test-fig-all.tex
```

then:

```
[user@machine~:]$ dvisvgm [options] test-fig-all.dvi
```

With this we can generate `svg` files that preserve our typographic fonts.

13 Using arara and dvisvgm

An example¹⁰ the next code which requires two compilations and use `arara` to generate an image in `pdf` format. We'll save our example as `test.tex`.

```
1 % arara: lualatex: { options: [-recorder] }
2 % arara: lualatex: { options: [-recorder] }
3 \documentclass{article}
4 \usepackage[osf]{libertinus}
5 \usepackage{tikz}
6 \usetikzlibrary{calc,tikzmark}
7 \begin{document}
8 By taking logarithms of both sides:
9
10 %<*ltximg>
11 \[
12 t = \frac{30\cdot\ln(3/22)}{\ln(15/22)}
13 \tikzmark{calculator}\approx\tikzmark{otherside}
14 156
15 \]
16 \begin{tikzpicture}[overlay,remember picture]
17 \coordinate (target) at ($(pic cs:calculator)!1/2!(pic cs:otherside) - (0,.5ex)$);
18 \draw[arrows=>] (target) ++(0,-2ex) node [anchor=north] {use calculator} -- (target);
19 \end{tikzpicture}
20 %</ltximg>
21 \end{document}
```

Now we just run:

```
[user@machine~:]$ ltximg --arara test.tex
```

And we already have our image `test-fig-1.pdf` on `./images`.

Let's modify the example¹¹ to generate an image in `svg` format using `dvisvgm` and `arara`.

```
1 % arara: lualatex: { options: [--output-format=dvi] }
2 % arara: dvisvgm: { options: [--exact-bbox, -o test-fig-1.svg] }
3 % ltximg: extenv: {picture}
4 % ltximg: options: {arara,norun,noprew}
5 \documentclass{article}
6 \begin{document}
7 The best airplane ever drawn by David Carlisle. No packages used, just the classic and perhaps
8 forgotten \verb|\begin{picture} ... \end{picture}|.
9
10 \begin{picture}(200,100)
11 \put(30,40){\line(1,0){150}}
12 \put(30,40){\line(0,1){60}}
13 \put(30,100){\line(1,0){20}}
14 \put(50,100){\line(1,-4){10}}
15 \put(60,60){\line(1,0){100}}
16 \put(160,60){\line(1,-1){20}}
17 \put(100,50){\line(0,-1){80}}
18 \put(130,50){\line(0,-1){80}}
```

¹⁰Adapted from [How to get tikzmark to work](#)

¹¹Adapted from [Draw an aircraft with Tikz](#)

```

19 \put(100,-30){\line(1,0){30}}
20 \put(100,61){\line(0,1){49}}
21 \put(130,61){\line(0,1){49}}
22 \put(100,110){\line(1,0){30}}
23 \end{picture}
24 \end{document}

```

We now run:

```

[user@machine~:]$ ltximg test.tex
[user@machine~:]$ cd images/
[user@machine~:]$ arara test-fig-all.tex

```

And we already have our image `test-fig-1.svg`.

14 Final notes

The process and operations required to generate the various types of *image formats* or *standalone files* have been described throughout the documentation, but, as discussed in section 11, sometimes the requirements are a *little different*.

Here are some of the ways you can use `ltximg` in conjunction with `arara`. If you are a user of `latexmk`¹², another great utility that automates the compilation process, the procedure is analogous, you just have to search (or write) the necessary “rule”.

This is the best way to extend the capabilities of the `ltximg`. Although many tasks can be *automated*, in the end only the user knows what the document contains and how it should be generated.

Finding the correct “regular expressions” and writing a “good documentation” would be the great mission (which does not end yet).

15 Change history

The most recent publicly released of `ltximg` is available at CTAN: <https://www.ctan.org/pkg/ltximg>. Historical and developmental versions are available at <https://github.com/pablgonz/ltximg>.

While general feedback via email is welcomed, specific bugs or feature requests should be reported through the issue tracker: <https://github.com/pablgonz/ltximg/issues>.

This is a short list of some of the notable changes in the history of the `ltximg` along with the versions, both development (devp) and public (ctan).

- v1.8 (ctan), 2020-07-24**
 - It is now possible to extract any environment.
 - Fix verbose option.
 - Add runs, latexmk and dvilua options.
 - Add log option.
 - All calls to the system are captured.
 - Re-write source code according to perl v5.3x.
 - Review of documentation.
- v1.7 (ctan), 2019-08-24**
 - Add scontents environment support.
 - Add filecontentsdefmacro environment support.
 - Fix regex in source code.
 - Update documentation.
- v1.6 (ctan), 2019-07-13**
 - Add zip and tar options.
 - Add new Verb from fvextra.
 - Fix and update source code and documentation.
- v1.5 (ctan), 2018-04-12**
 - Use GitHub to control version.
 - Rewrite and optimize most part of source code and options.
 - Change pdf2svg for pdftocairo.
 - Complete support for pst-exa package.
 - Escape characters in regex according to perl v5.2x.
- v1.4 (devp), 2016-11-29**
 - Remove and rewrite code for regex and system call.
 - Add arara compiler, clean and comment code.
 - Add dvips and dvipdfm(x) for creation images.
 - Add bmp, tiff image format.
- v1.3 (devp), 2016-08-14**
 - Rewrite some part of code (norun, nocrop, clean).

¹²<https://www.ctan.org/pkg/latexmk>

- Suport minted and tcolorbox package.
 - Escape some characters in regex according to perl v5.2x.
 - All options read from command line and input file.
 - Use /tmp dir for work process.
- v1.2 (ctan), 2015-04-22**
 - Remove unused modules.
 - Add more image format.
 - Fix regex.
- v1.1 (ctan), 2015-04-21**
 - Change mogrify to gs for image formats.
 - Create output file.
 - Rewrite source code and fix regex.
 - Change format date to iso format.
- v1.0 (ctan), 2013-12-01**
 - First public release.

16 References

- [1] KASTRUP, DAVID. “The preview package for L^AT_EX”. Available from CTAN, <https://www.ctan.org/pkg/preview>, 2017.
- [2] TANTAU, TILL. “The TikZ and PGF Packages”. Available from CTAN, <https://www.ctan.org/pkg/pgf>, 2020.
- [3] VAN ZANDT, TIMOTHY. “PSTricks - PostScript macros for generic T_EX”. Available from CTAN, <https://www.ctan.org/pkg/pstricks-base>, 2007.
- [4] VAN ZANDT, TIMOTHY. “pst-plot – Plot data using PSTricks”. Available from CTAN, <https://www.ctan.org/pkg/pst-plot>, 2019.
- [5] NIEPRASCHK, ROLF. “The pst-pdf Packages”. Available from CTAN, <https://www.ctan.org/pkg/pst-pdf>, 2019.
- [6] ROBERTSON, WILL. “The auto-pst-pdf Packages”. Available from CTAN, <https://www.ctan.org/pkg/auto-pst-pdf>, 2009.
- [7] VOß, HERBERT. “auto-pst-pdf-lua - Using LuaL^AT_EX with PSTricks”. Available from CTAN, <https://www.ctan.org/pkg/auto-pst-pdf-lua>, 2018.
- [8] VOß, HERBERT. “pst-exa - Typeset PSTricks examples, with pdfT_EX”. Available from CTAN, <https://www.ctan.org/pkg/pst-exa>, 2017.
- [9] VOß, HERBERT. “pst2pdf - A script to compile PSTricks documents via pdfT_EX”. Available from CTAN, <https://www.ctan.org/pkg/pst2pdf>, 2017.
- [10] THE L^AT_EX₃ PROJECT. “graphics – Enhanced support for graphics”. Available from CTAN, <https://www.ctan.org/pkg/graphicx>, 2017.
- [11] OBERDIEK, HEIKO. “The grfext package”. Available from CTAN, <https://www.ctan.org/pkg/grfext>, 2019.
- [12] VAN ZANDT, TIMOTHY. “The xcomment package”. Available from CTAN, <https://www.ctan.org/pkg/xcomment>, 2010.
- [13] ADRIAENS, HENDRI. “The extract package”. Available from CTAN, <https://www.ctan.org/pkg/extract>, 2019.
- [14] TRZECIAK, TOMASZ M. “The cachepic package”. Available from CTAN, <https://www.ctan.org/pkg/cachepic>, 2009.
- [15] MITTELBACH, FRANK. “The doc and shortvrb Packages”. Available from CTAN, <https://www.ctan.org/pkg/doc>, 2020.
- [16] VAN ZANDT, TIMOTHY. “The fancyvrb package - Fancy Verbatims in L^AT_EX”. Available from CTAN, <https://www.ctan.org/pkg/fancyvrb>, 2020.
- [17] HOFFMANN, JOBST. “The listings package”. Available from CTAN, <https://www.ctan.org/pkg/listings>, 2020.
- [18] POORE, GEOFFREY M. “The minted package - Highlighted source code in L^AT_EX”. Available from CTAN, <https://www.ctan.org/pkg/minted>, 2017.
- [19] THE ISLAND OF T_EX. “arara - The cool T_EX automation tool”. Available from CTAN, <https://www.ctan.org/pkg/arara>, 2020.
- [20] GIESEKING, MARTIN. “dvisvgm - A fast DVI to SVG converter”. Available from CTAN, <https://ctan.org/pkg/dvisvgm>, 2020.

17 Index of Documentation

The *italic* numbers denote the pages where the corresponding entry is described.

A	
article (class)	2
auto-pst-pdf (package)	6
auto-pst-pdf-lua (package)	6
B	
book (class)	2
C	
cachepic (package)	2
Compiler	
arara	10
dvi _l ualatex	10
dvi _p dfmx	10
dvips	10
latex	10
lualatex	10–12
pdf _l atex	8, 10
xelatex	10, 12
Compiler options	
-output-directory	5
-recorder	5
-shell-escape	5
D	
\DeclareTCBListing	3
\DefineShortVerb	3
\DefineVerbatimEnvironment	3
doc (package)	3
Docstrip tag	
ltximg	7
noltximg	7
remove	7
Document class	
article	2
book	2
letter	2
report	2
E	
Environments suport by default	
PSTexample	6
nopreview	7
pgfpicture	6
postscript	6
preview	6
psgraph	6
pspicture	6
tikzpicture	6
Environments	
BVerbatim	3
CenterExample	3
Example	3
LTXexample	3
LVerbatim	3
PCenterExample	3
PSTcode	3
PSTexample	6
PSideBySideExample	3
SaveVerbatim	3
SideBySideExample	3
VerbatimOut	4
Verbatim	3
alltt	3
boxedverbatim	3
chklisting	3
comment	3
demo	3
extcolorbox	4
extikzpicture	4
filecontens	2
filecontentsdefmacro	4
filecontentsdefstarred	4
filecontentssdef	4
filecontentssgdefmacro	4
filecontentssgdef	4
filecontentsshere	4
filecontents	4, 11
listingcont	3
listing	3
lstlisting	3
minted	3
nopreview	7
pgfinterruptpicture	6
preview	6, 7, 9
programL	3
programf	3
programl	3
programsc	3
programs	3
programt	3
program	3
pyglist	3
pygmented	3
scontents	4
sourcecode	3
sverbatim	3
tcexternal	4
tcblisting	3
tcboutputlisting	4
tcbwritetmp	4
tikzpicture	1, 11
verbatimtab	3
verbatimwrite	4
verbatim	3
xcomment	3
extract (package)	2
F	
fancyvrb (package)	3
File extentions	
.bmp	8
.dvi	12, 13

.eps	8
.jpg	8
.log	8
.ltx	8
.pdf	9
.png	8
.ppm	9
.svg	9
.tar.gz	9
.tex	8
.tif	8
.xdv	12, 13
.zip	9
\fverb	3

G

\graphicspath	5
graphicx (package)	2, 5
grfext (package)	2, 5

I

Image formats

bmp	8
eps	7
jpg	7
pdf	4, 7, 8, 10–13
png	7
ppm	7
svg	8, 10, 12, 13
tif	8
\include	2
\includecomment	3
\includegraphics	1, 5, 9–12
\input	2

L

letter (class)	2
listings (package)	3
\lstinline	3
\lstMakeShortInline	3
\lstnewenvironment	3

M

\MakeSpecialShortVerb	3
\mint	3
minted (package)	3
\mintinline	3

N

\newenvsc	4
\NewListingEnvironment	3
\newmint	3
\newminted	3
\newmintinline	3
\NewProgram	3
\newtabverbatim	3
\newtcexternalizeenvironment	4
\newtcexternalizetcolorbox	4
\NewTCBListing	3
\newtcblisting	3
\newverbatim	3

O

Operating system

Linux	2
Windows	2

ltximg options in command line

--arara	10, 12
--bmp	8
--clean	10
--deltenv	10
--dpi	8
--dvilua	10
--dvi pdf	10
--dvips	10
--eps	8
--extrenv	10
--force	9
--gray	9
--help	8
--imgdir	9
--jpg	8
--latexmk	10
--latex	6, 10
--log	8
--luatex	10
--margins	9
--myverb	9
--nocrop	9, 12
--nopdf	9
--noprew	5–7, 9, 12
--norun	5, 9, 12, 13
--output	9
--png	8
--ppm	9
--prefix	9
--runs	9
--skipenv	10
--srcenv	9
--subenv	9, 12
--svg	9
--tar	9
--tif	8
--verbenv	10
--verbose	8
--version	8
--writenv	10
--xetex	6, 10
--zip	9

ltximg options in input file

deltenv	11
extrenv	11
options	11
skipenv	11
verbenv	11
writenv	11

P

Package options

swpl	6
tcb	6

Packages

auto-pst-pdf-lua	6
------------------	---

auto-pst-pdf	6	pstricks (package)	1, 6, 10
cacheptic	2	\pyginline	3
doc	3	\pygment	3
extract	2		
fancyvrb	3	Q	
graphicx	2, 5	\qverb	3
grfext	2, 5		
listings	3	R	
minted	3	\renewtcexternalizeenvironment	4
pgf	6	\renewtcexternalizetcolorbox	4
preview	2, 5–12	report (class)	2
pst-exa	6		
pst-pdf	6	S	
pst-plot	6	\Scontents	3
pstricks	1, 6, 10	Scripts	
shortvrb	3	latexindent	2
tikz	1, 6	latexmk	10, 14
xcomment	2	latexpand	2
\pagestyle	2	pdfcrop	2, 9, 10, 12
pgf (package)	6	ps2pdf	10
\PrependGraphicsExtensions*	5	pst2pdf	1, 2
preview (package)	2, 5–12	shortvrb (package)	3
Programs		\specialcomment	3
arara	12, 14	\spverb	3
chktex	2	swpl (package option)	6
dvisvgm	12, 13		
ghostscript	1, 2, 7–9	T	
pdftocairo	8, 9, 12	tcb (package option)	6
pdftoeps	7	\tcboxverb	3
pdftoppm	7, 9	\thispagestyle	2
pdftops	8	tikz (package)	1, 6
perl	1, 2		
poppler-utils	1, 2, 7, 8	V	
\ProvideTCBListing	3	\Verb	3
pst-exa (package)	6	\verb	3
pst-pdf (package)	6		
pst-plot (package)	6	X	
		xcomment (package)	2