LATEX ENVIRONMENTS

LTXIMG

TO IMAGE FORMAT

v1.7 — 2019-08-24*

CTAN: http://www.ctan.org/pkg/ltximg
GIT: 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 in individual files using `ghostscript` and `poppler-utils`. Generates a file with only extracted environments and another with environments converted to `\includegraphics`.

Contents

1 Motivation

The original idea was to extend the functionality of the `pst2pdf` script (only for `pspicture` and `postscript`) to work with `tikzpicture` and other environments.

The tikz 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 separate files for environments and converted into images.
2. Generate a file with only the extracted environments.
3. Generate a file replacing the environments by `\includegraphics`.

From the side of TEX there are some packages that cover several of these points such as the preview, xcomment, external and cachepic 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`. Finding the correct *regular expressions* and writing *documentation* would be the great mission (which does not end yet).

*This file describes a documentation for version 1.7, last revised 2019-08-24.
†E-mail: «pablgonz@yahoo.com»

2 Required Software

For the complete operation of `ltximg` you need to have a modern TEX distribution such as TEXLive or MiKTEX, 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 TEXLive 2019 for `Windows` includes `ltximg` and all requirements, MiKTEX users must install the appropriate software for full operation.

The script has been tested on `Windows` (version 10) and `Linux` (fedora 30) in x64 architecture using `ghostscript` v9.26, `poppler-utils` v0.52 to v0.73 and `perl` from v5.28 to v5.30.

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 ⟨*verbatim content*⟩, the ⟨*input file*⟩, the ⟨*output file*⟩ and the ⟨*steps process*⟩.

3.1 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` can only be commented lines and after `\end{document}` can go any type of content, internally will split the ⟨*input file*⟩ at this points.

If the ⟨*input file*⟩ contains files using `\input` or `\include` 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 `latexpand` first and then process the file.

Like `\input` or `\include`, 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` and `latexindent`, especially if you want to extract the source code of the environments.

An example of the ⟨*input file*⟩:

```
% some commented lines at begin document
\documentclass{article}
\usepackage{tikz}
\begin{document}
Some text
\begin{tikzpicture}
Some code
\end{tikzpicture}
Always use \verb|\begin{tikzpicture}|
and \verb|\end{tikzpicture}| to open
and close environment
\begin{tikzpicture}
Some code
\end{tikzpicture}
Some text
\end{document}
% some lines after end document
```

3.2 Verbatim contents

One of the greatest capabilities of `ltximg` script is to skip the complications that *verbatim style* content produces with the extraction of environments. In order to skip the complications, the verbatim content is classified into three types:

- Verbatim in line
- Verbatim standard
- Verbatim write

Each of these classifications works differently within the creation and extraction process using different regular expressions for it.

3.2.1 Verbatim in line

The small pieces of code written in the same line using a verbatim command are considered ⟨*verbatim in line*⟩, such as `\verb|<code>|`. Most verbatim commands provide by packages minted, fancyvrb and listings have been tested and are fully supported. They are automatically detected the verbatim command generates by `\newmint` and `\newmintinline` and the following command list:

- `\mint`
- `\spverb`
- `\qverb`
- `\fverb`
- `\verb`
- `\Verb`
- `\lstinline`
- `\pyginline`
- `\pygment`
- `\Scontents`
- `\tcboxverb`
- `\mintinline`

Some packages define abbreviated versions for verbatim commands 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 commands, such as shortvrb or doc for example in which there is no explicit command 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*⟩:

```
\UndefineShortVerb{\|}
\DefineShortVerb{\|}
```

depending on the package you are using. If your verbatim command is not supported by default or can not detect, use the options described in 6.2 and 6.3.

3.2.2 Verbatim standard

These are the classic environments for writing code are considered ⟨*verbatim standard*⟩, such as `verbatim` and `lstlisting` environments. The following list is considered as ⟨*verbatim standard*⟩ environments:

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

They are automatically detected ⟨*verbatim standard*⟩ environments generates by commands:

- `\DefineVerbatimEnvironment`
- `\NewListingEnvironment`
- `\DeclareTCBListing`
- `\ProvideTCBListing`
- `\lstnewenvironment`
- `\newtabverbatim`
- `\specialcomment`
- `\includecomment`
- `\newtcblisting`
- `\NewTCBListing`
- `\newverbatim`
- `\NewProgram`
- `\newminted`

If any of the ⟨*verbatim standard*⟩ environments is not supported by default or can not detected, you can use the options described in 6.2 and 6.3.

3.2.3 Verbatim write

Some environments have the ability to write external files or memory directly, these environments are considered ⟨*verbatim write*⟩, such as `filecontents` or `VerbatimOut` environments. The following list is considered as ⟨*verbatim write*⟩ environments:

- scontents
- filecontents
- tcboutputlisting
- tcbexternal
- tcbwritetmp
- extcolorbox
- extikzpicture
- VerbatimOut
- verbatimwrite
- filecontentsdef
- filecontentshere
- filecontentsdefmacro

They are automatically detected ⟨*verbatim write*⟩ environments generates by commands:

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

If any of the ⟨*verbatim write*⟩ environments is not supported by default or can not detected, you can use the options described in 6.2 and 6.3.

3.3 Steps process

For creation of the image formats, extraction of code 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`, the working directory are `/workdir`, the directory for images are `/workdir/images` and the user's temporary directory is `/tmp` and we want to generate images in `pdf` format together with the source codes of the environments.

Comment and ignore
The first step is read and validated [⟨*options*⟩] from the command line and `test.tex`, verifying that `test.tex`, `test-out` and the directory `/images` are in `/workdir`, create the directory `/workdir/images` if it does not exist and a temporary directory `/tmp/hG45uVklv9`. The entire file `test.tex` is loaded in memory and 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}` and the files `test-fig-1.tex`, `test-fig-2.tex`, ... are generated and saved in `/images`.

Create random file
In the second step, with the file already loaded in memory, creating a temporary file with a random number (1981 for example) 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 `/workdir`.

2. If script is call *whit* `-n`,`--noprew` options, all code to extract its put inside the `preview` environment. 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 `/workdir` whit the same preamble of `test.tex` but the body only contains code that you want to extract.

Generate image formats
In the third 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`, separate in individual files `test-fig-1.pdf`, `test-fig-2.pdf`, ... and copy to `/workdir/images/`. The file `test-fig-1981.tex` is moved to the `/workdir/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` you must keep this in mind if you use `arara`.

Create output file
In the fourth 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:

```
\usepackage{graphicx}
\graphicspath{{images/}}
\usepackage{grfext}
\PrependGraphicsExtensions*{.pdf}
```

If the packages graphicx and grfext are already loaded and the command `\graphicspath` is found in the input file were detected automatically and only the changes will be added then proceed to run:

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

generating the file `test-out.pdf`.

Now the script read the files `test-fig-1981.fls` and `test-out.fls`, extract the information from the temporary files generated in the process and then delete them together with the directory `/tmp/hG45uVklv9`. An example for input and output file:

```
\documentclass{article}
\usepackage{tikz}
\begin{document}
Some text
\begin{tikzpicture}
Some code
\end{tikzpicture}
Always use \verb|\begin{tikzpicture}|
and \verb|\end{tikzpicture}| to open
and close environment
\begin{tikzpicture}
some code
\end{tikzpicture}
Some text
\end{document}
```

test.tex

```
\documentclass{article}
\usepackage{tikz}
\usepackage{graphicx}
\graphicspath{{images/}}
\usepackage{grfext}
\PrependGraphicsExtensions*{.pdf}
\begin{document}
Some text
\includegraphics[scale=1]{test-fig-1}
Always use \verb|\begin{tikzpicture}|
and \verb|\end{tikzpicture}| to open
and close environment
\includegraphics[scale=1]{test-fig-2}
Some text
\end{document}
```

test-out.tex

4 Extract content

The script provides two ways to extract content from ⟨*input file*⟩, using ⟨*environments*⟩ and ⟨*docstrip tags*⟩. Some environment (including a starred * version) are supported by default and if the environments are nested, the outermost will be extracted.

4.1 Default environments

`\begin{preview}` ⟨*env content*⟩ `\end{preview}` — Environment provide by preview package. If `preview` environments found in the input file will be extracted and converted these. Internally converts all environments to extract in `preview` environments. Is better comment this package in preamble unless the option `-n,--noprew` is used.

`\begin{pspicture}` ⟨*env content*⟩ `\end{pspicture}` — Environment provide by pstricks package. The plain syntax `\pspicture ... \endpspicture` its converted to `\begin{pspicture} ... \end{pspicture}`.

`\begin{psgraph}` ⟨*env content*⟩ `\end{psgraph}` — Environment provide by pst-plot package. The plain syntax `\psgraph ... \endpsgraph` its converted to `\begin{psgraph} ... \end{psgraph}`.

`\begin{postscript}` ⟨*env content*⟩ `\end{postscript}` — Environment provide by pst-pdf and auto-pst-pdf packages. Since the pst-pdf and auto-pst-pdf packages internally use the preview package, is better comment this in preamble.

`\begin{tikzpicture}` ⟨*env content*⟩ `\end{tikzpicture}` — Environment provide by tikz package. The plain syntax `\tikzpicture ... \tikzpicture` its converted to `\begin{tikzpicture} ... \end{tikzpicture}` but no a short `\tikz...;`.

`\begin{pgfpicture}` ⟨*env content*⟩ `\end{pgfpicture}` — Environment provide by pgf package. Since the script uses a *recursive regular expression* to extract environments, no presents problems if present `pgfinterruptpicture`.

`\begin{PSTexample}` ⟨*env content*⟩ `\end{PSTexample}` — Environment provide by pst-exa packages. The script automatically detects the `\begin{PSTexample}`

`...\end{PSTexample}` environments and processes them as separately compiled files. The user should have loaded the package with the [swpl] or [tcb] option and run the script using `--latex` or `--xetex`.

If you need to extract more environments you can use one of the options described in 6.2 or 6.3.

4.2 Extract whit docstrip tags

`%<*ltximg>` ⟨*content*⟩ `%</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.

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

4.3 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, for example auxiliary files to generate a graphic. The script provides a convenient way to solve this situation.

`\begin{nopreview}` ⟨*env content*⟩ `\end{nopreview}`

Environment provide by preview package. Internally the script converts all no extract environments to `\begin{nopreview} ... \end{nopreview}`. Is better comment this package in preamble unless the option `-n,--noprew` is used.

`%<*noltximg>` ⟨*content*⟩ `%</noltximg>`

All content betwen `%<*noltximg> ... %</noltximg>` are ignored and no extract. The start and closing of the tag must be at the beginning of the line.

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

`%<*remove>` ⟨*content*⟩ `%</remove>`

All content betwen `%<*remove> ... %</remove>` are deleted in the ⟨*output file*⟩. The start and closing of the tag must be at the beginning of the line.

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

If you want to remove specific environments automatically you can use one of the options described in 6.2 or 6.3.

5 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 `pdftoeps`. 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
```

6 How to use

6.1 Syntax

The syntax for `ltximg` is simple:

```
[user@machine ~:]$ ltximg ⟨compiler⟩ [⟨options⟩] [--] ⟨file.ext⟩
```

The extension ⟨*ext*⟩ 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.

6.2 Options in command line

`ltximg` provides a *command line interface* with short and long option names. They may be given before the name of the file. Also, the order of specifying the options is significant. Certain options accept a list separate by commas, this require a separated by white space or equals sign `=` between option and list and if it's the last option need `--` at the end. Multiple short options can be bundling.

`-h, --help` ⟨bolean⟩ (default: off)

Display a command line help text and exit.

`-l, --license` ⟨bolean⟩ (default: off)

Display a license text and exit.

`-v, --version` ⟨bolean⟩ (default: off)

Display the current version (1.7) and exit.

`-d, --dpi` ⟨int⟩ (default: 150)

Dots per inch for images files.

`-t, --tif` ⟨bolean⟩ (default: off)

Create a `.tif` images files using `ghostscript`.

`-b, --bmp` ⟨bolean⟩ (default: off)

Create a `.bmp` images files using `ghostscript`.

`-j, --jpg` ⟨bolean⟩ (default: off)

Create a `.jpg` images files using `ghostscript`.

`-p, --png` ⟨bolean⟩ (default: off)

Create a `.png` transparent image files using `ghostscript`.

`-e, --eps` ⟨bolean⟩ (default: off)

Create a `.eps` image files using `pdftops`.

`-s, --svg` ⟨bolean⟩ (default: off)

Create a `.svg` image files using `pdftocairo`.

`-P, --ppm` ⟨bolean⟩ (default: off)

Create a `.ppm` image files using `pdftoppm`.

`-g, --gray` ⟨bolean⟩ (default: off)

Create a gray scale for all images using `ghostscript`. The line behind this options is:

```
[user@machine ~:]$ gs -q -dNOSAFER -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress       \
                      -sColorConversionStrategy=Gray -dProcessColorModel=/DeviceGray
```

`-f, --force` ⟨bolean⟩ (default: off)

Try to capture `\psset{...}` and `\tikzset{...}` to extract. When using the `--force` option the script will try to capture `\psset{...}` or `\tikzset{...}` and leave it inside the `preview` environment, any line that is between `\psset{...}` and `\begin{pspicture}` or between `\tikzset{...}` and `\begin{tikzpicture}` will be captured.

`-n, --noprew` ⟨bolean⟩ (default: off)

Create images files without preview package. The `\begin{preview}...\end{preview}` lines are only used as delimiters for extracting the content *without* using the package preview. Sometimes it is better to use it together with `--force`.

`-m, --margin` ⟨numeric⟩ (default: 0)

Set margins in bp for `pdfcrop`.

`-o, --output` ⟨output file name⟩ (default: empty)

Create ⟨*output file name*⟩ whit all extracted environments/contents converted to `\includegraphics`. The ⟨*output file name*⟩ must not contain extension.

`--imgdir` ⟨string⟩ (default: images)

The name of directory for save images and source code.

`--zip` ⟨bolean⟩ (default: off)

Compress only the files generated by the script during the process in `/images` in `.zip` format. Does not include ⟨*output file*⟩.

`--tar` ⟨bolean⟩ (default: off)

Compress only the files generated by the script during the process in `/images` in `.tar.gz` format. Does not include ⟨*output file*⟩.

`--verbose` ⟨bolean⟩ (default: off)

Show verbose information in screen and change `-interaction` for compiler.

`--srcenv` ⟨bolean⟩ (default: off)

Create separate files whit *only code* for all extracted environments, is mutually exclusive whit `--subenv`.

`--subenv` ⟨bolean⟩ (default: off)

Create sub files whit *preamble* and code for all extracted environments, is mutually exclusive whit `--srcenv`.

`--arara` ⟨bolean⟩ (default: off)

Use `arara` for compiler files, need to pass `-recorder` option to ⟨*input file*⟩:

```
% arara : <compiler> : { options: [-recorder] }
```

`--xetex` ⟨bolean⟩ (default: off)

Using `xelatex` compiler ⟨*input file*⟩ and ⟨*output file*⟩.

`--latex` ⟨bolean⟩ (default: off)

Using `latex»dvips»ps2pdf` compiler in ⟨*input file*⟩ and `pdflatex` for ⟨*output file*⟩.

`--dvips` ⟨bolean⟩ (default: off)

Using `latex»dvips»ps2pdf` for compiler ⟨*input file*⟩ and ⟨*output file*⟩.

`--dvipdf` ⟨bolean⟩ (default: off)

Using `latex»dvipdfmx` for compiler ⟨*input file*⟩ and ⟨*output file*⟩.

`--luatex` ⟨bolean⟩ (default: off)

Using `lualatex` for compiler ⟨*input file*⟩ and ⟨*output file*⟩.

`--prefix` ⟨string⟩ (default: fig)

Add prefix append to each files created.

`--norun` ⟨bolean⟩ (default: off)

Run script, but no create images. This option is designed to debug the file and when you only need to extract the code

`--nopdf` ⟨bolean⟩ (default: off)

Don't create a `.pdf` image files.

`--nocrop` ⟨bolean⟩ (default: off)

Don't run `pdfcrop` in image files.

`--clean` ⟨doc|pst|tkz|all|off⟩ (default: doc)

Removes specific content in ⟨*output file*⟩. Valid values for this option are:

`doc` All content after `\end{document}` is removed.
`pst` All `\psset{...}` and pstricks package is removed.
`tkz` All `\tikzset{...}` is removed.
`all` Activates doc, pst and tkz.
`off` Deactivate all.

`--verbcmd` ⟨command name⟩ (default: myverb)

Set custom verbatim command `\myverb|<code>|`.

`--extrenv` ⟨list separate by comma⟩ (default: empty)

List of environments to extract, need `--` at end.

`--skipenv` ⟨list separate by comma⟩ (default: empty)

List of environments that should not be extracted and that the script supports by default, need `--` at end.

`--verbenv` ⟨list separate by comma⟩ (default: empty)

List of ⟨*verbatim standard*⟩ environment support, need `--` at end.

`--writenv` ⟨list separate by comma⟩ (default: empty)

List of ⟨*verbatim write*⟩ environment support, need `--` at end.

`--deltenv` ⟨list separate by comma⟩ (default: empty)

List of environment deleted in ⟨*output file*⟩, need `--` at end.

6.3 Options in input file

Many of the ideas in this section are inspired by the `arara` program (I adore it). 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 option. 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:

```
%<*remove>
% ltximg : options : {png,srcenv,xetex}
% ltximg : extrenv : {description}
%</remove>
```

7 Examples

7.1 From command line

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

Create a `/images` directory whit all extracted environments converted to image formats (`pdf`, `svg`) in individual files, an ⟨*output file*⟩ `test-out.ltx` whit all supported 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*⟩.

7.2 From input file

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

```
%<*remove>
% ltximg : options : {output = file-out, noprew, imgdir = pics, prefix = env, clean = doc}
% ltximg : skipenv : {tikzpicture}
% ltximg : deltenv : {filecontents}
%</remove>
```

and run:

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

Create a `/pics` directory 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 `pdflatex` and preview package for ⟨*input file*⟩ and ⟨*output file*⟩.

8 Change history

Some of the notable changes in the history of the `ltximg` along with the versions, both development (devp) and public (ctan).

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` v5.4x.x

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)
- Suport `minted` and `tcolorbox` package
- Escape some characters in regex according to `perl` v5.2x.x
- 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

9 Index of Documentation