The pstool package

Concept by Zebb Prime
Package by Will Robertson
wspr81@gmail.com

v0.7 2008/08/26

Contents

Part I
User documentation

1 Introduction

While pdfLaTeX is a great improvement in many ways over the 'old method' of DVI→PS→PDF, it loses the ability to interface with a generic PostScript workflow, used to great effect in numerous packages, most notably PSTricks and psfrag.

Until now, the best way to use these packages while running pdfLaTeX has been to use the pst-pdf package, which processes the entire document through a filter, sending the relevant PostScript environments through a single pass of DVI→PS→PDF. The resulting PDF versions of each image are then included into the pdfLaTeX document. The auto-pst-pdf package provides a wrapper to perform all of this automatically.

The disadvantage with this method is that for every document compilation, *every* graphic must be re-processed. The pstool package uses a different approach to allow each graphic to be processed only as-needed, speeding up and simplifying the typesetting of the main document.

More flexible usage to provide a complete replacement for pst-pdf (e.g., support the `\beginpostscript` environment) is planned for a possible future release. If you simply need to automatically convert plain EPS files to PDF, I recommend using the epstopdf package with the `[update,prepend]` package options. (The two packages should be completely compatible.)

2 Processing modes

The generic command provided by this package is

`\pstool[`⟨*graphicx options*⟩`]{`⟨*filename*⟩`}{`⟨*input definitions*⟩`}`

It converts the graphic ⟨*filename*⟩`.eps` to ⟨*filename*⟩`.pdf` through a unique DVI→PS→PDF process for each graphic, using the preamble of the main document. The resulting graphic is then inserted into the document, with optional ⟨*graphicx options*⟩. The third argument to `\pstool` allows arbitrary ⟨*input definitions*⟩ (such as `\psfrag` directives) to be inserted before the figure as it is processed.

By default `\pstool` can be used in the following modes:

`\pstool` Process the graphic ⟨*filename*⟩ if no PDF of the same name exists, or if the source EPS file is *newer* than the PDF;

`\pstool*` Always process this figure; and,

`\pstool!` Never process this figure.

It is useful to define higher-level commands with `\pstool` for including specific types of EPS graphics that take advantage of psfrag. As an example, this package defines the following commands (some of which use internal features of pstool. These commands all support the `*` or `!` suffixes.

`\psfragfig[`⟨*opts*⟩`]{`⟨*filename*⟩`}` This is the catch-all macro to support a wide range of graphics naming schemes. It insert an EPS file named either ⟨*filename*⟩`.eps` or ⟨*filename*⟩`-psfrag.eps` (in order of preference), and uses psfrag definitions contained within either the file ⟨*filename*⟩`.tex` or ⟨*filename*⟩`-psfrag.tex`.

This command can be used to insert figure produced by the MATHEMATICA package MathPSfrag or the MATLAB package matlabfrag. `\psfragfig` also accepts an optional braced argument as shown next.

`\psfragfig[`⟨*opts*⟩`]{`⟨*filename*⟩`}{`⟨*input definitions*⟩`}` As above, but inserts the arbitrary code ⟨*input definitions*⟩, which will usually be used for defining new or overriding existing psfrag commands.

`\laprintfig[`⟨*opts*⟩`]{`⟨*filename*⟩`}` Insert figures that have been produced with Matlab's laprint package. This package requires a special case because the psfrag output it produces is rather awkward to deal with.

3 Package options

3.1 Forcing/disabling graphics processing

While the suffixes `*` and `!` can be used to force or disable (respectively) the processing of each individual graphic, sometimes we want to do this on a global level. The following package options override *all* `\pstool` (and related) macros:

`[process=auto]` This is the default mode as described in the previous section, in which graphics are only (re-)processed if the eps file is newer or the pdf file does not exist;

`[process=all]` All `\pstool` graphics are processed; and,

`[process=none]` No `\pstool` graphics are processed.[1]

Also note that it would be nice to detect the age of files other than the eps and pdf graphics in order to affect the processing decisions. This is planned for a possible future release.

3.2 Cropping graphics

Graphics are cropped to the appropriate size with the preview package. Sometimes, however, this will not be sufficient, such as when an inserted label protrudes from the natural bounding box of the figure, or when the original bounding box of the figure is wrong. A good way to solve this problem is to use the pdfcrop program (requires a Perl installation under Windows). This can be activated in pstool with the `[pdfcrop]` package option.

3.3 Temporary files & cleanup

Each figure that is processed spawns an auxiliary LaTeX compilation through dvi→ps→pdf. This process is named after the name of the figure with a suffix;

[1] If pstool is loaded in a LaTeX document in dvi mode, this is the option that is used since no external processing is required for these graphics.

the default is [suffix={-pstool}]. All of these suffixed files are "temporary" in that they may be deleted once they are no longer needed.

As an example, if the figure is called ex.eps, the files that are created are ex-pstool.tex, ex-pstool.dvi, ... The [cleanup] package option declares via a list of filename suffixes which temporary files are to be deleted after processing.

The default is [cleanup={.tex, .dvi, .ps, .pdf, .log, .aux}]. To delete none of the temporary files, choose [cleanup={}] (useful for debugging).

3.4 Interaction mode of the auxiliary processes

Each graphic echoes the output of its auxiliary process to the console window; unless you are trying to debug errors there is little interest in seeing this information. The behaviour of these auxiliary processes are governed globally by the [mode] package option, which takes the following parameters:

[mode=batch] hide almost all of the LaTeX output (*default*);

[mode=nonstop] echo all LaTeX output but continues right past any errors; and

[mode=errorstop] prompt for user input when errors in the source are encountered.

These three package options correspond to the LaTeX command line options -interaction=batchmode, =nonstopmode, and =errorstopmode, respectively.

4 Miscellaneous details

4.1 The \EndPreamble command

At present, pstool scans the preamble of the main document by redefining \begin{document}, but this is rather fragile because many classes and packages do their own redefined which overwrites pstool's attempt. In this case, place the command

```
\EndPreamble
```

where-ever you'd like the preamble in the auxiliary processing to end. This is also handy to bypass anything in the preamble that will never be required for the figures but which will slow down or otherwise conflict with the auxiliary processing.

4.2 Cross-reference limitations

The initial release of this package does not support cross-references within the psfrag labels of the included graphics. (If, say, you wish to refer to an equation number within a figure.) A future release of pstool may fix this limitation.

4.3 A note on file paths

pstool does its best to ensure that you can put image files where-ever you like and the auxiliary processing will still function correctly. In order to ensure this, the external `pdflatex` compilation uses the `-output-directory` feature of pdfTEX. This command line option is definitely supported on all platforms in TeX Live 2008 and MiKTeX 2.7, but earlier distributions may not be supported.

One problem that pstool does not (currently) solve on its own is the inclusion of images that do not exist in subdirectories of the main document. For example, `\pstool{../Figures/myfig}` will not process by default because pdfTEX usually does not have permission to write into folders that are higher in the heirarchy than the main document. This can be worked around presently in two different ways: (although maybe only for Mac OS X and Linux)

1. Give `pdflatex` permission to write anywhere with the command:
   ```
   openout_any=a pdflatex ...
   ```
2. Create a symbolic link in the working directory to a point higher in the path: `ln -s ../../PhD ./PhD`, for example, and then refer to the graphics through this symbolic link.

I hope to directly solve this problem in the future by using a caching folder for the auxiliary processing in such cases.

5 Package information

The most recent publicly released version of pstool is available at CTAN:

`http://tug.ctan.org/pkg/pstool/`

Historical and developmental versions are available at GitHub:

`http://github.com/wspr/pstool/`

While general feedback at `wspr81@gmail.com` is welcomed, specific bugs should be reported through the bug tracker at FogBugz: `https://wspr.fogbugz.com/` (click 'TASKS: Enter a New Case').

Part II
Implementation

```
\ProvidesPackage{pstool}[2008/08/26␣v0.7
  Wrapper␣for␣processing␣PostScript/psfrag␣figures]
```

External packages

```
\RequirePackage{%
  catchfile,color,ifpdf,ifplatform,graphicx,suffix,xkeyval}
\RequirePackage{inversepath}[2008/07/31␣v0.2]
```

Allocations

```
 \if@pstool@always@    \newif\if@pstool@always@
  \if@pstool@never@    \newif\if@pstool@never@
\if@pstool@pdfcrop@    \newif\if@pstool@pdfcrop@
  \if@pstool@nofig@    \newif\if@pstool@nofig@
        \pstool@out    \newwrite\pstool@out
```

These are cute

```
 \OnlyIfFileExists    \providecommand\OnlyIfFileExists[2]{\IfFileExists{#1}{#2}{}}
  \NotIfFileExists    \providecommand\NotIfFileExists[2]{\IfFileExists{#1}{}{#2}}
```

5.1 Package options

```
pdfcrop    \DeclareOptionX{pdfcrop}{\@pstool@pdfcrop@true}

process    \define@choicekey*{pstool.sty}{process}[\@tempa\@tempb]{%
                 all,none,auto}{%
             \ifcase\@tempb\relax
               \@pstool@always@true
             \or
               \@pstool@never@true
             \or
             \fi
           }

   mode    \define@choicekey*{pstool.sty}{mode}
             [\@tempa\@tempb]{errorstop,nonstop,batch}{%
               \edef\pstool@mode{\@tempa␣mode}%
}
\ExecuteOptionsX{mode=batch}
```

cleanup
\pstool@rm@files

```
\DeclareOptionX{cleanup}{\def\pstool@rm@files{#1}}
\ExecuteOptionsX{cleanup={.tex,.dvi,.ps,.pdf,.log,.aux}}
```

suffix
\pstool@suffix

```
\DeclareOptionX{suffix}{\def\pstool@suffix{#1}}
\ExecuteOptionsX{suffix={-pstool}}
```

```
\ifshellescape\else
  \ExecuteOptionsX{process=none}
  \PackageWarning{pstool}{^^J\space\space%
    Package␣option␣[process=none]␣activated^^J\space\space
    because␣-shell-escape␣is␣not␣enabled.^^J%
    This␣warning␣occurred}
\fi
```

```
\ProcessOptionsX
```

6 Macros

Used to echo information to the console output. Can't use ecause it's asynchronous with any `\immediate\write18` processes (for some reason).

\pstool@echo

```
\def\pstool@echo#1{\immediate\write18{echo␣"#1"}}
```

Command line abstractions between platforms:

```
\edef\pstool@cmdsep{\ifwindows\string&\else\string;\fi\space}
\edef\pstool@rm@cmd{\ifwindows␣del␣\else␣rm␣--␣\fi}
```

Delete a file if it exists:

\pstool@rm

```
\newcommand\pstool@rm[1]{%
  \OnlyIfFileExists{\ip@directpath#1}{%
    \immediate\write18{%
      cd␣"\ip@directpath"\pstool@cmdsep\pstool@rm@cmd␣"#1"}}%
}
```

Generic function to execute a command on the shell and pass its exit status back into LaTeX. Any number of `\pstool@exe` statements can be made consecutively followed by `\pstool@endprocess`, which also takes an argument. If *any* of the shell calls failed, then the execution immediately skips to the end and expands `\pstool@error` instead of the argument to `\pstool@endprocess`.

\pstool@exe

```
\newcommand\pstool@exe[3]{%
  \pstool@echo{^^J␣===␣pstool:␣#1␣===}%
  \pstool@writestatus{#2}{#3}%
  \pstool@retrievestatus{#2}%
  \ifnum\pstool@status␣>␣\z@
    \PackageWarning{pstool}{Execution␣failed␣during␣
          process:^^J␣␣#3^^J}%
    \expandafter\pstool@abort
  \fi}
```

Edit this definition to print something else when graphic processing fails.

\pstool@error

```
\def\pstool@error{\fbox{\parbox{\linewidth}{\color{red}%
      \ttfamily\scshape
  An␣error␣occured␣processing␣graphic␣\upshape‘\ip@directpath%
        \ip@lastelement’}}}
```

\pstool@abort

```
\def\pstool@abort#1\pstool@endprocess{\pstool@error\@gobble}
\let\pstool@endprocess\@firstofone
```

It is necessary while executing commands on the shell to write the exit status to a temporary file to test for failures in processing. (If all versions of `pdflatex` supported input pipes, things might be different.)

\pstool@writestatus

```
\def\pstool@writestatus#1#2{%
  \immediate\write18{%
    cd␣"#1"␣\pstool@cmdsep
    #2␣\pstool@cmdsep
    \ifwindows
       call␣echo
         \string^\@percentchar␣ERRORLEVEL\string^\@percentchar
    \else
       echo␣\detokenize{$?}
    \fi
    >␣\pstool@statusfile}%
```

That’s the execution; now we need to flush the write buffer to the status file. This ensures the file is written to disk properly (allowing it to be read by `\CatchFileEdef`). Not necessary on Windows, whose file writing is evidently more crude/immediate.

```
                            \ifwindows\else
                              \immediate\write18{%
                                touch␣#1\pstool@statusfile}%
                            \fi}
\pstool@statusfile        \def\pstool@statusfile{pstool-statusfile.txt}
```

Read the exit status from the temporary file and delete it.
#1 is the path
Status is recorded in \pstool@status.

```
\pstool@retrievestatus    \def\pstool@retrievestatus#1{%
                            \CatchFileEdef{\pstool@status}{#1\pstool@statusfile}{}%
                            \pstool@rm{\pstool@statusfile}%
                            \ifx\pstool@status\pstool@statusfail
                              \PackageWarning{pstool}{%
                                Status␣of␣process␣unable␣to␣be␣determined:^^J␣␣#1^^J%
                                Trying␣to␣proceed...␣}%
\pstool@status                \def\pstool@status{0}%
                            \fi}
\pstool@statusfail        \def\pstool@statusfail{\par␣}% what results when TeX reads an empty
                                  file
```

6.1 File age detection

Use ls (or dir) to detect if the EPS is newer than the PDF.

```
\pstool@IfnewerEPS        \def\pstool@IfnewerEPS{%
                            \edef\pstool@filenames{\ip@lastelement.eps\space␣%
                                  \ip@lastelement.pdf\space}%
                            \immediate\write18{%
                              cd␣"\ip@directpath"\pstool@cmdsep
                              \ifwindows
                                dir␣/T:W␣/B␣/O-D␣"\ip@lastelement.eps"␣"%
                                      \ip@lastelement.pdf"␣>␣\pstool@statusfile
                              \else
                                ls␣-t␣"\ip@lastelement.eps"␣"\ip@lastelement.pdf"␣>␣%
                                      \pstool@statusfile
                              \fi
                            }%
                            \pstool@retrievestatus{\ip@directpath}%
                            \ifx\pstool@status\pstool@filenames
                              \expandafter\@firstoftwo
```

```
  \else
    \expandafter\@secondoftwo
  \fi
}
```

A wrapper for \inversepath*. Long story short, always need a relative path to a filename even if it's in the same directory.

\pstool@getpaths

\ip@directpath

```
\def\pstool@getpaths#1{%
  \edef\@tempa{\unexpanded{\inversepath*}{#1}}%
  \@tempa% calculate filename, path & inverse path
  \ifx\ip@directpath\@empty
    \def\ip@directpath{./}%
  \fi
}
```

7 Command parsing

User input is \pstool (with optional * or ! suffix) which turns into one of the following three macros depending on the mode.

\pstool@alwaysprocess

```
\newcommand\pstool@alwaysprocess[3][]{%
  \pstool@getpaths{#2}%
  \pstool@process{#1}{#2}{#3}}
```

\pstool@neverprocess

```
\newcommand\pstool@neverprocess[3][]{%
  \includegraphics[#1]{#2}}
```

For regular operation, which processes the figure only if the command is starred, or the PDF doesn't exist.

\pstool@maybeprocess

```
\newcommand\pstool@maybeprocess[3][]{%
  \pstool@getpaths{#2}%
  \IfFileExists{#2.pdf}{%
    \pstool@IfnewerEPS{% needs info from \pstool@getpaths
      \pstool@process{#1}{#2}{#3}%
    }{%
      \includegraphics[#1]{#2}%
    }%
  }{%
    \pstool@process{#1}{#2}{#3}%
  }}
```

8 User commands

Finally, define \pstool as appropriate for the mode:

```
          \ifpdf
            \if@pstool@always@
              \let\pstool\pstool@alwaysprocess
\pstool       \WithSuffix\def\pstool!{\pstool@alwaysprocess}
\pstool*      \WithSuffix\def\pstool*{\pstool@alwaysprocess}
            \else\if@pstool@never@
              \let\pstool\pstool@neverprocess
\pstool       \WithSuffix\def\pstool!{\pstool@neverprocess}
\pstool*      \WithSuffix\def\pstool*{\pstool@neverprocess}
            \else
              \let\pstool\pstool@maybeprocess
\pstool       \WithSuffix\def\pstool!{\pstool@neverprocess}
\pstool*      \WithSuffix\def\pstool*{\pstool@alwaysprocess}
            \fi\fi
          \else
            \let\pstool\pstool@neverprocess
\pstool     \WithSuffix\def\pstool!{\pstool@neverprocess}
\pstool*    \WithSuffix\def\pstool*{\pstool@neverprocess}
          \fi
```

9 The figure processing

\ip@lastelement is the filename of the figure stripped of its path (if any)

```
\pstool@jobname  \def\pstool@jobname{\ip@lastelement\pstool@suffix}
```

And this is the main macro.

```
\pstool@process  \newcommand\pstool@process[3]{%
                   \pstool@echo{^^J}%
                   \pstool@write@processfile{#1}{#2}{#3}%
                   \pstool@exe{auxiliary␣process:␣\ip@lastelement\space}
                     {./}{latex
                       -shell-escape
                       -output-format=dvi
                       -output-directory="\ip@directpath"
                       -interaction=\pstool@mode\space
                           "\pstool@jobname.tex"}%
```

```
  \pstool@exe{dvips}{\ip@directpath}{%
    dvips␣"\pstool@jobname.dvi"}%
  \if@pstool@pdfcrop@
    \pstool@exe{ps2pdf}{\ip@directpath}{%
      ps2pdf␣"\pstool@jobname.ps"␣"\pstool@jobname.pdf"}%
    \pstool@exe{pdfcrop}{\ip@directpath}{%
      pdfcrop␣"\pstool@jobname.pdf"␣"\ip@lastelement.pdf"}%
  \else
    \pstool@exe{ps2pdf}{\ip@directpath}{%
      ps2pdf␣"\pstool@jobname.ps"␣"\ip@lastelement.pdf"}%
  \fi
  \pstool@echo{^^J===␣pstool:␣end␣processing␣===^^J}%
  \pstool@endprocess{%
    \pstool@cleanup
    \includegraphics[#1]{#2}}}
```

The file that is written for processing is set up to read the preamble of the original document and set the graphic on an empty page (cropping to size is done either here with preview or later with pdfcrop).

pstool@write@processfile

```
\def\pstool@write@processfile#1#2#3{%
    \immediate\openout\pstool@out␣#2\pstool@suffix.tex\relax
    \immediate\write\pstool@out{%
      \noexpand\pdfoutput=0% force DVI mode if not already
```

Input the main document; redefine the document environment so only the preamble is read:

\document

```
      \unexpanded{%
        \let\origdocument\document
        \let\EndPreamble\endinput
        \def\document{\endgroup\endinput}}%
      \noexpand\input{\jobname}%
```

Now the preamble of the process file: (restoring document's original meaning; empty \pagestyle removes the page number)

```
      \if@pstool@pdfcrop@\else
        \noexpand\usepackage[active,tightpage]{preview}
      \fi
      \unexpanded{%
        \let\document\origdocument
        \pagestyle{empty}}%
```

And the document body to place the graphic on a page of its own:

```
      \unexpanded{%
        \begin{document}
        \centering\null\vfill}%
      \if@pstool@pdfcrop@\else
        \noexpand\begin{preview}%
      \fi
      \unexpanded{#3}% this is the "psfrag" material
      \if@pstool@nofig@\else
        \noexpand\includegraphics[#1]{\ip@lastelement}%
      \fi
      \if@pstool@pdfcrop@\else
        \noexpand\end{preview}%
      \fi
      \unexpanded{%
        \vfill\end{document}}%
      }%
    \immediate\closeout\pstool@out}
```

\pstool@cleanup

```
\def\pstool@cleanup{%
  \@for\@ii:=\pstool@rm@files\do{%
    \pstool@rm{\pstool@jobname\@ii}%
}}
```

\EndPreamble

```
\providecommand\EndPreamble{}
```

10 User commands

These all support the suffixes * and !, so each user command is defined as a wrapper to \pstool.

for EPS figures with psfrag:

\psfragfig

```
\newcommand\psfragfig[2][]{\pstool@psfragfig{#1}{#2}{}}
```

\psfragfig*

```
\WithSuffix\newcommand\psfragfig*[2][]{\pstool@psfragfig{#1}{%
      #2}{*}}
```

\psfragfig

```
\WithSuffix\newcommand\psfragfig![2][]{\pstool@psfragfig{#1}{%
      #2}{!}}
```

Parse optional ⟨*input definitions*⟩

\pstool@psfragfig

```
\newcommand\pstool@psfragfig[3]{%
  \@ifnextchar\bgroup{%
    \pstool@@psfragfig{#1}{#2}{#3}%
  }{%
    \pstool@@psfragfig{#1}{#2}{#3}{}%
  }%
}
```

Search for both ⟨*filename*⟩ and ⟨*filename*⟩-`psfrag` inputs.

\pstool@@psfragfig

```
\newcommand\pstool@@psfragfig[4]{%
  \IfFileExists{#2-psfrag.eps}{%
```

\pstool@eps

```
    \def\pstool@eps{#2-psfrag}%
    \OnlyIfFileExists{#2.eps}{%
      \PackageWarning{pstool}{Graphic␣"#2.eps"␣exists␣but␣
            "#2-psfrag.eps"␣is␣being␣used}%
    }%
  }{%
    \IfFileExists{#2.eps}{%
```

\pstool@eps

```
      \def\pstool@eps{#2}%
    }{%
      \PackageError{pstool}{%
        No␣graphic␣"#2.eps"␣or␣"#2-psfrag.eps"␣found%
      }{%
        Check␣the␣path␣and␣whether␣the␣file␣exists.%
      }%
    }%
  }%
  \pstool#3[#1]{\pstool@eps}{%
    \InputIfFileExists{#2-psfrag.tex}{%
      \OnlyIfFileExists{#2.tex}{%
        \PackageWarning{pstool}{%
          File␣"#2.tex"␣exists␣that␣may␣contain␣macros␣for␣"%
                \pstool@eps.eps"^^J%
          But␣file␣"#2-psfrag.tex"␣is␣being␣used␣instead.%
        }%
      }%
    }{%
      \InputIfFileExists{#2.tex}{}{}%
    }%
    #4%
  }%
}
```

for Matlab's laprint:

\laprintfig
\laprintfig*
\laprintfig

```
\newcommand\laprintfig[2][]{\pstool@laprintfig{#1}{#2}{}}
\WithSuffix\newcommand\laprintfig*[2][]{\pstool@laprintfig{%
      #1}{#2}{*}}
\WithSuffix\newcommand\laprintfig![2][]{\pstool@laprintfig{%
      #1}{#2}{!}}
```

Parse optional ⟨*input definitions*⟩

\pstool@laprintfig

```
\newcommand\pstool@laprintfig[3]{%
  \@ifnextchar\bgroup{%
    \pstool@@laprintfig{#1}{#2}{#3}%
  }{%
    \pstool@@laprintfig{#1}{#2}{#3}{}%
  }%
}
```

\pstool@@laprintfig
\resizebox
\includegraphics

```
\newcommand\pstool@@laprintfig[4]{%
  \begingroup
    \@pstool@nofig@true
    \renewcommand\resizebox[3]{##3}%
    \renewcommand\includegraphics[2][]{\pstool#3[#1]{##2}{}}%
    \input{#2}%
  \endgroup
}
```

⟨*eof*⟩