The pstool package

Will Robertson and Zebb Prime

v0.4 2008/08/06

Contents

I Documentation 1

1 Introduction 2

2 Processing modes 2

3 Cropping graphics 3

4 Todo 3

II Implementation 4

4.1 File age detection 7

5 Command parsing 7

5.1 User commands 8

6 The figure processing 9

7 User commands 10

Part I
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 each relevant PostScript environment through DVI→PS→PDF. The resulting PDF versions of each image are then included into the pdfLaTeX document. The auto-pst-pdf package provided a wrapper to execute this separate process automatically.

The disadvantage in this route 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.

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 its own, unique, DVI→PS→PDF workflow using the preamble of the main document, and inserts it with optional ⟨*graphicx options*⟩. The third argument allows arbitrary ⟨*input definitions*⟩ (such as `\psfrag` directives) to be inserted before the figure as it is processed.

By default it can be used in the following modes:

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

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

`\pstool!` Never process this figure.

The package accepts options to override the above:

`[process=auto]` This is the default as described above;

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

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

It is possible 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. These commands all support the `*` or `!` suffices.

`\epsfig[`⟨*opts*⟩`]{`⟨*filename*⟩`}` Insert a plain EPS figure. It is more convenient than using, for example, the epstopdf package since it will regenerate the PDF if the EPS file changes.

`\psfragfig[`⟨*opts*⟩`]{`⟨*filename*⟩`}` Insert an EPS file with psfrag definitions contained within the file ⟨*filename*⟩`-psfrag.tex`. (Accepts an optional braced argument as shown next.)

`\psfragfig[`⟨*opts*⟩`]{`⟨*filename*⟩`}{`⟨*input definitions*⟩`}` Insert an EPS file with psfrag definitions contained either/or within the file ⟨*filename*⟩`-psfrag.tex` and supplied by the third argument ⟨*input definitions*⟩.

`\laprintfig[`⟨*opts*⟩`]{`⟨*filename*⟩`}` Insert figures that have been produced with MATLAB's laprint package.

`\mathpsfragfig[`⟨*opts*⟩`]{`⟨*filename*⟩`}` Insert figures that have been produced with MATHEMATICA's MathPSfrag package.

3 Cropping graphics

Graphics are cropped to the appropriate size with the preview package. Sometimes, however, this will not be good enough when an inserted label protrudes from the natural bounding box of the figure. 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.

4 Todo

1. Test `\laprint`, `\psfragfig`, `\matlabfig`, `\mathfig`, especially with figures in a relative path.

[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.

2. Cleanup temporary files (straight from auto-pst-pdf).
3. Generalise "olding" code for multiple files.
4. Support optional ⟨*input definitions*⟩ for all user commands??
5. (Maybe) support epstool for cropping the graphics.
6. Direct support for `\includegraphics` with EPS files.
7. Check for correct behaviour in shells other than bash.
8. More flexible usage (support things like `\begin{postscript}` in pst-pdf).
9. mylatex integration.

Part II
Implementation

```
\ProvidesPackage{pstool}[2008/08/06 v0.4
  Wrapper for processing PostScript/psfrag figures]
```

External packages

```
\RequirePackage{%
  catchfile,color,ifpdf,ifplatform,
  inversepath,graphicx,suffix,xkeyval}
```

Initialisations

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

Package options

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

\define@choicekey*{pstool.sty}{process}[\@tempa\@tempb]{%
      all,none,auto}{%
```

```
  \ifcase\@tempb\relax
    \@pstool@always@true
  \or
    \@pstool@never@true
  \or
  \fi}

\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
```

These are cute:

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

Command line abstractions between platforms:

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

\newcommand\pstool@try@rm[1]{%
  \begingroup
    \@for\@tempa:=#1\do{%
      \OnlyIfFileExists{\ip@directpath\@tempa}{%
        \immediate\write18{%
          cd␣"\ip@directpath"\pstool@cmdsep
          \pstool@rm␣"\@tempa"}}}%
  \endgroup}
```

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`.

```
\def\pstool@exe#1{%
```

```
  \immediate\write18{%
    cd␣"\ip@directpath"\pstool@cmdsep
    \pstool@writestatus{#1}%
  }%
  \pstool@retrievestatus\@tempa
  \ifnum\@tempa␣>␣0
    \PackageWarning{pstool}{%
          Execution␣failed␣during␣process:^^J␣␣#1^^J}%
    \expandafter\pstool@abort
  \fi}
```

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

\pstool@error
```
\def\pstool@error#1{\fbox{\color{red}\ttfamily\scshape
  An␣error␣occured␣processing␣this␣graphic.}}
```

\pstool@abort
```
\def\pstool@abort#1\pstool@endprocess{\pstool@error}
\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@statusfile
\pstool@writestatus
```
\def\pstool@statusfile{status-deleteme.txt}
\def\pstool@writestatus#1{%
  \ifwindows
    #1␣\pstool@cmdsep␣call␣echo
      \string^\@percentchar␣ERRORLEVEL\string^\@percentchar
      >␣\pstool@statusfile
  \else
    #1␣\pstool@cmdsep␣echo␣\detokenize{$?}␣>␣%
          \pstool@statusfile
  \fi
}
```

Read the exit status from the temporary file and delete it.

\pstool@retrievestatus
```
\def\pstool@retrievestatus#1{%
  \pstool@touchstatus
  \CatchFileEdef{#1}{\ip@directpath\pstool@statusfile}{}%
  \pstool@try@rm{\pstool@statusfile}%
}
```

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.

`\pstool@touchstatus`

```
\def\pstool@touchstatus{%
  \ifwindows\else
    \immediate\write18{%
      cd␣"\ip@directpath"\pstool@cmdsep
      touch␣\pstool@statusfile}%
  \fi
}
```

4.1 File age detection

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

`\pstool@datefiles`

```
\def\pstool@datefiles{%
  \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\@tempa
  \ifx\@tempa\pstool@filenames
    \@tempswatrue
  \else
    \@tempswafalse
  \fi
}
```

5 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][]{%
  \inversepath*{#2}% calculate filename, path & inverse path
  \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][]{%
  \inversepath*{#2}% calculate filename, path & inverse path
  \IfFileExists{#2.pdf}{%
    \pstool@datefiles
    \if@tempswa\expandafter\@firstoftwo
      \else\expandafter\@secondoftwo\fi{%
        \pstool@process[#1]{#2}{#3}%
      }{%
        \includegraphics[#1]{#2}}%
  }{%
    \pstool@process[#1]{#2}{#3}%
  }}
```

5.1 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
```

6 The figure processing

```
\pstool@process \newcommand{\pstool@process}[3][]{%
    \pstool@write@processfile{#1}{#2}{#3}%
    \pstool@exe{latex␣-shell-escape␣-output-format=dvi
        -interaction=batchmode␣"\ip@lastelement-process.tex"}%
    \pstool@exe{dvips␣"\ip@lastelement-process.dvi"}%
    \if@pstool@pdfcrop@
      \pstool@exe{ps2pdf␣"\ip@lastelement-process.ps"␣"%
            \ip@lastelement-process.pdf"}%
      \pstool@exe{pdfcrop␣"\ip@lastelement-process.pdf"␣"%
            \ip@lastelement.pdf"}%
    \else
      \pstool@exe{ps2pdf␣"\ip@lastelement-process.ps"␣"%
            \ip@lastelement.pdf"}%
    \fi
    \pstool@endprocess{\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).

```
ostool@write@processfile \def\pstool@write@processfile#1#2#3{%
    \immediate\openout\pstool@out␣#2-process.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:

```
      \if@pstool@nopreamble@
        \unexpanded{%
          \documentclass{minimal}
          \usepackage{graphicx}}
      \else
        \unexpanded{%
          \let\origdocument\document
```

```
\document
            \def\document{\endgroup\endinput}}%
          \noexpand
            \input{\ip@inversepath\jobname}%
        \fi
```

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
        \if@pstool@nopreamble@\else
          \unexpanded{%
            \let\document\origdocument
            \pagestyle{empty}}%
        \fi
```

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}
```

7 User commands

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

for plain EPS figures (no psfrag):

```
\epsfig
\epsfig*
\epsfig
\pstool@epsfig
```

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

\def\pstool@epsfig#1[#2]#3{%
  \begingroup
    \@pstool@nopreamble@true
    #1[#2]{#3}{}%
  \endgroup
}
```

for EPS figures with psfrag:

```
\psfragfig
\psfragfig*
\psfragfig
\pstool@psfragfig
```

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

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

```
\pstool@@psfragfig
```

```
\def\pstool@@psfragfig#1[#2]#3#4{%
  #1[#2]{#3}{%
    \InputIfFileExists{#3-psfrag}{}{}%
    #4}%
}
```

for Matlab's laprint:

```
\laprintfig
\laprintfig*
```

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

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

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

for Mathematica's MathPSfrag:

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

\pstool@mathpsfragfig \def\pstool@mathpsfragfig#1[#2]#3{%
                  #1[#2]{#3-psfrag}{\input{#3-psfrag}}%
                }
```

⟨*eof*⟩