The pstool package

Concept by Zebb Prime
Package by Will Robertson*

v1.2 2009/05/24

Abstract

This package defines the `\psfragfig` user command for including EPS files that use psfrag features in a pdfLaTeX document. The command `\pstool` can be used to define other commands with similar behaviour.

Contents

Part I
User documentation

1 Introduction

While directly producing PDF output with 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

*wspr81@gmail.com

has been to use the pst-pdf package, which processes the entire document through a filter, sending the relevant PostScript environments (only) through a single pass of LaTeX producing DVI→PS→PDF. The resulting PDF versions of each graphic are then included into the pdfLaTeX document in a subsequent compilation. 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.

At present this package is designed solely as a replacement for pst-pdf in the rôle of supporting the psfrag package (which it loads) in pdfLaTeX.

More flexible usage to provide a complete replacement for pst-pdf (e.g., supporting the `\begin{postscript}` environment) is planned for a later 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 (epstopdf and pstool should be completely compatible).

2 Getting started

Load the package as usual; no package options are required by default, but there are a few useful options described later in section 3. Note that you do not need to load psfrag separately.

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.

The command `\pstool` can take an optional `*` or `!` suffix to change the behaviour of the command:

`\pstool` Process the graphic ⟨*filename*⟩`.eps` if ⟨*filename*⟩`.pdf` does not already exist, or if the EPS file is *newer* than the PDF;

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

`\pstool!` Never process this figure.

The behaviour in these three cases can be overridden globally by the package option `[process]` as described in section 3.1.

It is useful to define higher-level commands based on `\pstool` for including specific types of EPS graphics that take advantage of psfrag. As an example, this package defines the following command, which also supports the `*` or `!` suffixes described above.

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

This command can be used to insert figures 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 to define new or override existing psfrag commands.

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 with suffixes are only (re-)processed if the EPS file is newer or the PDF file does not exist;

`[process=all]` Suffixes are ignored and all `\pstool` graphics are processed;

`[process=none]` Suffixes are ignored and 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.

[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.

3.2 Cropping graphics

The default option `[crop=preview]` selects the preview package to crop graphics to the appropriate size for each auxiliary process.

However, when an inserted label protrudes from the natural bounding box of the figure, or when the original bounding box of the figure is wrong, the preview package will not always produce a good result (with parts of the graphic trimmed off the edge). A robust method to solve this problem is to use the pdfcrop program instead.[2] This can be activated in pstool with the `[crop=pdfcrop]` package option.

In the future I plan to also support epstool for doing the same thing.

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 an appended string suffix; 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. When `[mode=batch]` is activated, then dvips is also run in 'quiet mode'.

[2] pdfcrop requires a Perl installation under Windows, freely available from http://www.

3.5 Auxiliary processing command line options

The command line options passed to each program of the auxiliary processing can be changed with the following package options:

`[latex-options]` ;

`[dvips-options]` ;

`[ps2pdf-options]` ; and,

`[pdfcrop-options]` .

For the most part these will be unnecessary, although passing the correct options to `ps2pdf` can sometimes be a little obscure. For example, I use the following for generating figures in my thesis:

```
ps2pdf-options={-dCompatibilityLevel=1.4 -dPDFSETTINGS=/prepress}
```

I believe this incantation forces fonts to be embedded within the individual figure files, without which some printers and PDF viewers have trouble with the textual labels.

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 redefining 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 (although is must be placed before `\begin{document}` for obvious reasons). 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 will hopefully lift this limitation.

---

activestate.com/Products/activeperl/index.plex

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').

5.1 Licence

Part II
Implementation

LaTeX2e file 'pstool.sty' generated by the 'filecontents' environment from source 'pstool' on 2009/05/24.

```
\ProvidesPackage{pstool}[2009/05/24 v1.2
  Wrapper for processing PostScript/psfrag figures]
```

External packages

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

Allocations

```
\if@pstool@always@
\if@pstool@never@
\if@pstool@pdfcrop@
\if@pstool@verbose@
\pstool@out
```

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

These are cute

```
\OnlyIfFileExists
\NotIfFileExists
```

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

5.2 Package options

```
crop
```

```
\define@choicekey*{pstool.sty}{crop}[\@tempa\@tempb]{%
      preview,pdfcrop}{%
  \ifcase\@tempb\relax
    \@pstool@pdfcrop@false
  \or
    \@pstool@pdfcrop@true
  \or
  \fi
}

\define@choicekey*{pstool.sty}{process}[\@tempa\@tempb]{%
```

```
process             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}{%
      \ifnum\@tempb=2\relax
        \@pstool@verbose@false
      \else
        \@pstool@verbose@true
      \fi
      \edef\pstool@mode{\@tempa␣mode}%
  }
  \ExecuteOptionsX{mode=batch}

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

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

 latex-options  \DeclareOptionX{latex-options}{\def\pstool@latex@opts{#1}}
 dvips-options  \DeclareOptionX{dvips-options}{\def\pstool@dvips@opts{#1}}
ps2pdf-options  \DeclareOptionX{ps2pdf-options}{\def\pstool@pspdf@opts{#1}}
pdfcrop-options  \DeclareOptionX{pdfcrop-options}{\def\pstool@pdfcrop@opts{#1}}
  \ExecuteOptionsX{%
    latex-options={},
    dvips-options={},
    ps2pdf-options={},
    pdfcrop-options={}}

  \ifpdf\else
    \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
\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{%
  \if@pstool@verbose@
    \pstool@echo@verbose{#1}%
  \fi}
```

`\pstool@echo@verbose`

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

\let\pstool@includegraphics\includegraphics
```

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:
#1: path
#2: filename

`\pstool@rm`

```
\newcommand\pstool@rm[2]{%
  \OnlyIfFileExists{#1#2}{%
    \immediate\write18{%
      cd␣"#1"\pstool@cmdsep\pstool@rm@cmd␣"#2"}}%
}
```

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`.

#1: 'name' of process
#2: relative path where to execute the command
#3: the command itself

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

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

\pstool@error
```
\def\pstool@error{%
  \fbox{%
    \parbox{0.8\linewidth}{%
      \color{red}\raggedright\ttfamily\scshape\small
      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@shellexecute
```
\def\pstool@shellexecute#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{#1}{\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`

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

`\ip@directpath`

```
      \def\ip@directpath{./}%
    \fi
```

Strip off a possible wayward `.eps` suffix.

```
    \edef\ip@lastelement{%
      \expandafter\pstool@stripEPS\ip@lastelement.eps\@nil
    }%
  }
```

`\pstool@stripEPS`

```
  \def\pstool@stripEPS#1.eps#2\@nil{#1}
```

```
test.eps\@nil->test
test.eps.eps\@nil->test
```

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}{#3}}
```

```
  \ifpdf
```

`\pstool@neverprocess`

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

```
\pstool@neverprocess
    \begingroup
      #3%
      \pstool@includegraphics[#1]{#2}%
    \endgroup}
\fi
```

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

8 User commands

Finally, define \pstool as appropriate for the mode:

```
\ifpdf
  \if@pstool@always@
    \let\pstool\pstool@alwaysprocess
    \WithSuffix\def\pstool!{\pstool@alwaysprocess}
\pstool
    \WithSuffix\def\pstool*{\pstool@alwaysprocess}
\pstool*
  \else\if@pstool@never@
    \let\pstool\pstool@neverprocess
    \WithSuffix\def\pstool!{\pstool@neverprocess}
\pstool
    \WithSuffix\def\pstool*{\pstool@neverprocess}
\pstool*
  \else
    \let\pstool\pstool@maybeprocess
    \WithSuffix\def\pstool!{\pstool@neverprocess}
\pstool
    \WithSuffix\def\pstool*{\pstool@alwaysprocess}
\pstool*
  \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[2]{%
                   \pstool@echo@verbose{^^J^^J===␣pstool:␣begin␣processing␣
                         ===}%
                   \pstool@write@processfile{#1}{\ip@directpath%
                         \ip@lastelement}{#2}%
                   \pstool@exe{auxiliary␣process:␣\ip@lastelement\space}
                     {./}{latex
                       -shell-escape
                       -output-format=dvi
                       -output-directory="\ip@directpath"
                       -interaction=\pstool@mode\space
                       \pstool@latex@opts\space
                           "\pstool@jobname.tex"}%
```

Execute dvips in quiet mode if latex is not run in (non/error)stop mode:

```
                   \pstool@exe{dvips}{\ip@directpath}{%
                     dvips␣\if@pstool@verbose@\else␣-q␣\fi␣-Ppdf␣%
                          \pstool@dvips@opts\space␣"\pstool@jobname.dvi"}%
                   \if@pstool@pdfcrop@
                     \pstool@exe{ps2pdf}{\ip@directpath}{%
                       ps2pdf␣\pstool@pspdf@opts\space␣"\pstool@jobname.ps"␣"%
                            \pstool@jobname.pdf"}%
                     \pstool@exe{pdfcrop}{\ip@directpath}{%
                       pdfcrop␣\pstool@pdfcrop@opts\space␣"%
                            \pstool@jobname.pdf"␣"\ip@lastelement.pdf"}%
                   \else
                     \pstool@exe{ps2pdf}{\ip@directpath}{%
                       ps2pdf␣\pstool@pspdf@opts\space␣"\pstool@jobname.ps"␣"%
                            \ip@lastelement.pdf"}%
```

```
    \fi
    \pstool@endprocess{%
      \pstool@cleanup
      \pstool@includegraphics[#1]{\ip@directpath%
            \ip@lastelement}}%
    \pstool@echo@verbose{^^J=== pstool: end processing ===^^J}%
}
```

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^^J% 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^^J%
        \let\EndPreamble\endinput^^J%
        \def\document{\endgroup\endinput}^^J}%
      \noexpand\input{\jobname}^^J%
```

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}^^J%
      \fi
      \unexpanded{%
        \let\document\origdocument^^J%
        \pagestyle{empty}^^J}%
```

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

```
      \unexpanded{%
        \begin{document}^^J%
        \centering\null\vfill^^J}%
      \if@pstool@pdfcrop@\else
        \noexpand\begin{preview}^^J%
```

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

\pstool@cleanup
\def\pstool@cleanup{%
  \@for\@ii:=\pstool@rm@files\do{%
    \pstool@rm{\ip@directpath}{\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.

#1: possible graphicx options
#2: graphic name (possibly with path)
#3: \pstool suffix (i.e., ! or * or empty
#4: possible psfrag macros

\pstool@@psfragfig

```
\newcommand\pstool@@psfragfig[4]{%
```

Find the.eps file to use.

```
  \IfFileExists{#2-psfrag.eps}{%
    \edef\pstool@eps{#2-psfrag}%
    \OnlyIfFileExists{#2.eps}{%
      \PackageWarning{pstool}{Graphic␣"#2.eps"␣exists␣but␣
            "#2-psfrag.eps"␣is␣being␣used}%
    }%
  }{%
    \IfFileExists{#2.eps}{%
      \edef\pstool@eps{#2}%
    }{%
      \PackageError{pstool}{%
        No␣graphic␣"#2.eps"␣or␣"#2-psfrag.eps"␣found%
      }{%
        Check␣the␣path␣and␣whether␣the␣file␣exists.%
      }%
    }%
  }%
```

Find the .tex file to use.

```
  \IfFileExists{#2-psfrag.tex}{%
    \edef\pstool@tex{#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.%
      }%
    }%
  }{%
    \IfFileExists{#2.tex}{%
      \edef\pstool@tex{#2.tex}%
    }{%
```

```
      \let\pstool@tex\@empty
      \PackageWarning{pstool}{%
        No␣file␣"#2.tex"␣or␣"#2-psfrag.tex"␣can␣be␣found
        that␣may␣contain␣macros␣for␣"\pstool@eps.eps"%
      }%
    }%
  }%
  \ifx\pstool@tex\@empty
    \pstool#3[#1]{\pstool@eps}{#4}%
  \else
    \expandafter\pstool@@@psfragfig\expandafter{\pstool@tex}{%
          #3[#1]}{#4}%
  \fi
}
```

Break out the separate function in order to expand `\pstool@tex` before writing it.

`\pstool@@@psfragfig`

```
\newcommand\pstool@@@psfragfig[3]{%
  \pstool#2{\pstool@eps}{%
    \csname␣@input\endcsname{#1}%
    #3%
  }%
}
```

⟨*eof*⟩