

The pstool package

Concept by Zebb Prime
Package by Will Robertson*

v1.0 2008/08/31

Abstract

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

Contents

I	USER DOCUMENTATION	1	II	IMPLEMENTATION	7
1	Introduction	1	6	Macros	9
2	Getting started	2	7	Command parsing	12
3	Package options	3	8	User commands	13
4	Miscellaneous details	5	9	The figure processing	14
5	Package information	6	10	User commands	16

Part I

User documentation

1 Introduction

While directly producing PDF output with pdf \LaTeX is a great improvement in many ways over the ‘old method’ of $\text{DVI} \rightarrow \text{PS} \rightarrow \text{PDF}$, 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 pdf \LaTeX

*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 $\text{DVI} \rightarrow \text{PS} \rightarrow \text{PDF}$. The resulting PDF versions of each graphic are then included into the `pdf\LaTeX` 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 `pdf\LaTeX`.

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 $\text{DVI} \rightarrow \text{PS} \rightarrow \text{PDF}$ 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.

¹If `pstool` is loaded in a L^AT_EX 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 $\text{DVI} \rightarrow \text{PS} \rightarrow \text{PDF}$. 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 it 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.

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 pdf_latex compilation uses the `-output-directory` feature of pdf_lTeX. 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 pdf_lTeX usually does not have permission to write into folders that are higher in the hierarchy 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 pdf_latex 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

This package is freely modifiable and distributable under the terms and conditions of the L^AT_EX Project Public Licence, version 1.3c or greater (your choice). The latest version of this license is available at: <http://www.latex-project.org/lppl.txt>. This work is maintained by WILL ROBERTSON.

Part II

Implementation

LaTeX2e file ‘pstool.sty’ generated by the ‘filecontents’ environment from source ‘pstool’ on 2009/03/12.

```
1 \ProvidesPackage{pstool}[2008/08/31_v1.0
2   Wrapper_for_processing_PostScript/psfrag_figures]
```

External packages

```
3 \RequirePackage{%
4
5   catchfile,color,ifpdf,ifplatform,graphicx,psfrag,suffix,xkeyval}
6 \RequirePackage{inversepap}[2008/07/31_v0.2]
```

Allocations

```
\if@pstool@always@ 6 \newif\if@pstool@always@
\if@pstool@never@ 7 \newif\if@pstool@never@
\if@pstool@pdfcrop@ 8 \newif\if@pstool@pdfcrop@
\if@pstool@verbose@ 9 \newif\if@pstool@verbose@
\pstool@out 10 \newwrite\pstool@out
```

These are cute

```
\OnlyIfFileExists 11 \providecommand\OnlyIfFileExists[2]{\IfFileExists{#1}{#2}{}}
\NotIfFileExists 12 \providecommand\NotIfFileExists[2]{\IfFileExists{#1}{#2}{}}
```

5.2 Package options

```
crop 13 \define@choicekey*{pstool.sty}{crop}[\@tempa\@tempb]{%
14   preview,pdfcrop}{%
15     \ifcase\@tempb\relax
16     \@pstool@pdfcrop@false
17     \or
18     \@pstool@pdfcrop@true
19     \or
20     \fi
21 }

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

```

process          all,none,auto}{%
22  \ifcase\@tempb\relax
23  \@pstool@always@true
24  \or
25  \@pstool@never@true
26  \or
27  \fi
28  }

mode 29  \define@choicekey*{pstool.sty}{mode}
30  [ \@tempa\@tempb]{errorstop,nonstop,batch}{%
31  \ifnum\@tempb=2\relax
32  \@pstool@verbose@false
33  \else
34  \@pstool@verbose@true
35  \fi
36  \edef\pstool@mode{\@tempa_mode}%
37  }
38  \ExecuteOptionsX{mode=batch}

cleanup 39  \DeclareOptionX{cleanup}{\def\pstool@rm@files{#1}}
\pstool@rm@files 40  \ExecuteOptionsX{cleanup={.tex,.dvi,.ps,.pdf,.log,.aux}}

suffix 41  \DeclareOptionX{suffix}{\def\pstool@suffix{#1}}
\pstool@suffix 42  \ExecuteOptionsX{suffix={-pstool}}

latexOptions 43  \DeclareOptionX{`Options.options0`Options`0-
options.options0`Optionslatex-options}{\def%
\pstool@latex@opts  \pstool@latex@opts{#1}}
dvipsOptions 44  \DeclareOptionX{`Options.options0`Options`0-
options.options0`Optionsdvips-options}{\def%
\pstool@dvips@opts  \pstool@dvips@opts{#1}}
ps2pdfOptions 45  \DeclareOptionX{`Options.options0`Options`0-
options.options0`Optionsps2pdf-options}{\def%
\pstool@pspdf@opts  \pstool@pspdf@opts{#1}}
pdfcropOptions 46  \DeclareOptionX{`Options.options0`Options`0-
options.options0`Optionspdfcrop-options}{\def%
\pstool@pdfcrop@opts  \pstool@pdfcrop@opts{#1}}
47  \ExecuteOptionsX{%
48  latex-options={},
49  dvips-options={},

```



```

50   ps2pdf-options={},
51   pdfcrop-options={}}

52 \ifshellescape\else
53   \ExecuteOptionsX{process=none}
54   \PackageWarning{pstool}{^^J\space\space%
55     Package_option_[process=none]_activated^^J\space\space
56     because_-shell-escape_is_not_enabled.^^J%
57     This_warning_occurred}
58 \fi

59 \ProcessOptionsX

```

6 Macros

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

```

\pstool@echo 60 \def\pstool@echo#1{%
61   \if@pstool@verbose@
62     \pstool@echo@verbose{#1}%
63   \fi}

\pstool@echo@verbose 64 \def\pstool@echo@verbose#1{%
65   \immediate\write18{echo_"#1"}%
66   }

67 \let\pstool@includegraphics\includegraphics

```

Command line abstractions between platforms:

```

68 \edef\pstool@cmdsep{\ifwindows\string&\else\string;\fi\space}
69 \edef\pstool@rm@cmd{\ifwindows\del_\else\rm_\fi}

```

Delete a file if it exists (only the filename is supplied and the path is inferred):

```

\pstool@rm 70 \newcommand\pstool@rm[2]{%
71   \OnlyIfFileExists{#1#2}{%
72     \immediate\write18{%
73       cd_"#1"\pstool@cmdsep\pstool@rm@cmd_"#2"}}%
74   }

```

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 75 \newcommand\pstool@exe[3]{%
76   \pstool@echo{^^J===_pstool:_#1_===}%
77   \pstool@shellexecute{#2}{#3}%
78   \pstool@retrievestatus{#2}%
79   \ifnum\pstool@status_>_ \z@
80     \PackageWarning{pstool}{Execution_ failed_ during_
      process:^^J\space\space#3^^JThis_ warning_ occurred}%
81     \expandafter\pstool@abort
82   \fi}
```

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

```
\pstool@error 83 \def\pstool@error{%
84   \fbox{%
85     \parbox{0.8\linewidth}{%
86       \color{red}\raggedright\ttfamily\scshape\small
87       An_ error_ occurred_ processing_ graphic_ \upshape‘%
        \ip@directpath\ip@lastelement’}}}
```

```
\pstool@abort 88 \def\pstool@abort#1\pstool@endprocess{\pstool@error\@gobble}
89 \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 pdf_lat_ex supported input pipes, things might be different.)

```
\pstool@shellexecute 90 \def\pstool@shellexecute#1#2{%
91   \immediate\write18{%
92     cd_ "#1"_ \pstool@cmdsep
93     #2_ \pstool@cmdsep
94     \ifwindows
95       call_ echo
96       \string^ \@percentchar_ ERRORLEVEL \string^ \@percentchar
97     \else
98       echo_ \detokenize{${?}}
```

```

99      \fi
100      >\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.

```

101      \ifwindows\else
102          \immediate\write18{%
103              touch_#1\pstool@statusfile}%
104      \fi}
\pstool@statusfile 105 \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 106 \def\pstool@retrievestatus#1{%
107     \CatchFileEdef{\pstool@status}{#1\pstool@statusfile}{}%
108     \pstool@rm{#1}{\pstool@statusfile}%
109     \ifx\pstool@status\pstool@statusfail
110         \PackageWarning{pstool}{%
111             Status_of_process_unable_to_be_determined:^^J_#1^^J%
112             Trying_to_proceed..._}%
\pstool@status 113     \def\pstool@status{0}%
114     \fi}
\pstool@statusfail 115 \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 116 \def\pstool@IfnewerEPS{%
117     \edef\pstool@filenames{\ip@lastelement.eps\space_
118         \ip@lastelement.pdf\space}%
119     \immediate\write18{%
120         cd_"\ip@directpath"\pstool@cmdsep
121         dir_/T:W_/B_/O-D_"\ip@lastelement.eps"_"%
122         \ip@lastelement.pdf">\pstool@statusfile
123     \else

```

```

123      ls_t "\ip@lastelement.eps" "\ip@lastelement.pdf" > \pstoool@statusfile
124      \fi
125    }%
126    \pstoool@retrievestatus{\ip@directpath}%
127    \ifx\pstoool@status\pstoool@filenames
128      \expandafter\@firstoftwo
129    \else
130      \expandafter\@secondoftwo
131    \fi
132  }

```

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

```

\pstoool@getpaths 133 \def\pstoool@getpaths#1{%
134   \edef\@tempa{\unexpanded{\inversepath*}{#1}}%
135   \@tempa% calculate filename, path & inverse path
136   \ifx\ip@directpath\@empty
\ip@directpath 137   \def\ip@directpath{./}%
138   \fi

```

Strip off a possible wayward .eps suffix.

```

139   \edef\ip@lastelement{%
140     \expandafter\pstoool@stripEPS\ip@lastelement.eps\@nil
141   }%
142 }

```

```

\pstoool@stripEPS 143 \def\pstoool@stripEPS#1.eps#2\@nil{#1}

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

```

7 Command parsing

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

```

\pstoool@alwaysprocess 144 \newcommand\pstoool@alwaysprocess[3] [] {%
145   \pstoool@getpaths{#2}%
146   \pstoool@process{#1}{#3}}

147 \newcommand\pstoool@neverprocess[3] [] {%

```

```
\pstool@neverprocess 148 \pstool@includegraphics[#1]{#2}}
```

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

```
\pstool@maybeprocess 149 \newcommand\pstool@maybeprocess[3][]{%
150 \pstool@getpaths{#2}%
151 \IfFileExists{#2.pdf}{%
152 \pstool@ifnewerEPS{% needs info from \pstool@getpaths
153 \pstool@process{#1}{#3}%
154 }{%
155 \pstool@includegraphics[#1]{#2}%
156 }%
157 }{%
158 \pstool@process{#1}{#3}%
159 }}
```

8 User commands

Finally, define `\pstool` as appropriate for the mode:

```
160 \ifpdf
161 \if@pstool@always@
162 \let\pstool\pstool@alwaysprocess
\pstool 163 \WithSuffix\def\pstool!\{\pstool@alwaysprocess}
\pstool* 164 \WithSuffix\def\pstool*\{\pstool@alwaysprocess}
165 \else\if@pstool@never@
166 \let\pstool\pstool@neverprocess
\pstool 167 \WithSuffix\def\pstool!\{\pstool@neverprocess}
\pstool* 168 \WithSuffix\def\pstool*\{\pstool@neverprocess}
169 \else
170 \let\pstool\pstool@maybeprocess
\pstool 171 \WithSuffix\def\pstool!\{\pstool@neverprocess}
\pstool* 172 \WithSuffix\def\pstool*\{\pstool@alwaysprocess}
173 \fi\fi
174 \else
175 \let\pstool\pstool@neverprocess
\pstool 176 \WithSuffix\def\pstool!\{\pstool@neverprocess}
\pstool* 177 \WithSuffix\def\pstool*\{\pstool@neverprocess}
178 \fi
```

9 The figure processing

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

```
\pstool@jobname 179 \def\pstool@jobname{\ip@lastelement\pstool@suffix}
```

And this is the main macro.

```
\pstool@process 180 \newcommand\pstool@process[2]{%
181   \pstool@echo@verbose{^^J===\pstool:\begin_processing_===}%
182   \pstool@write@processfile{#1}{\ip@directpath%
      \ip@lastelement}{#2}%
183   \pstool@exe{auxiliary_process:\ip@lastelement\space}
184   {./}{latex
185     -shell-escape
186     -output-format=dvi
187     -output-directory="\ip@directpath"
188     -interaction=\pstool@mode\space
189     \pstool@latex@opts\space
190     "\pstool@jobname.tex"%
```

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

```
191   \pstool@exe{dvips}{\ip@directpath}{%
192     dvips_\if@pstool@verbose@else_q_\fi_Ppdf_%
      \pstool@dvips@opts\space"\pstool@jobname.dvi"%
193   \if@pstool@pdfcrop@
194     \pstool@exe{ps2pdf}{\ip@directpath}{%
195       ps2pdf_\pstool@pspdf@opts\space"\pstool@jobname.ps"%
      \pstool@jobname.pdf"%
196     \pstool@exe{pdfcrop}{\ip@directpath}{%
197       pdfcrop_\pstool@pdfcrop@opts\space"%
      \pstool@jobname.pdf"\ip@lastelement.pdf"%
198   \else
199     \pstool@exe{ps2pdf}{\ip@directpath}{%
200       ps2pdf_\pstool@pspdf@opts\space"\pstool@jobname.ps"%
      \ip@lastelement.pdf"%
201   \fi
202   \pstool@echo{^^J===\pstool:\end_processing_===^^J}%
203   \pstool@endprocess{%
204     \pstool@cleanup
205     \pstool@includegraphics[#1]{\ip@directpath%
      \ip@lastelement}}}
```

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 206 \def\pstool@write@processfile#1#2#3{%
207     \immediate\openout\pstool@out_#2\pstool@suffix.tex\relax
208     \immediate\write\pstool@out{%
209         \noexpand\pdfoutput=0^^J% force DVI mode if not already

```

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

```

210         \unexpanded{%
211             \let\origdocument\document^^J%
212             \let\EndPreamble\endinput^^J%
\document 213         \def\document{\endgroup\endinput}^^J}%
214         \noexpand\input{\jobname}^^J%

```

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

```

215         \if@pstool@pdfcrop@else
216             \noexpand\usepackage[active,tightpage]{preview}^^J%
217         \fi
218         \unexpanded{%
219             \let\document\origdocument^^J%
220             \pagestyle{empty}^^J}%

```

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

```

221         \unexpanded{%
222             \begin{document}^^J%
223             \centering\null\vfill^^J}%
224         \if@pstool@pdfcrop@else
225             \noexpand\begin{preview}^^J%
226         \fi
227         \unexpanded{#3^^J}% this is the "psfrag" material
228         \noexpand\includegraphics[#1]{\ip@lastelement}^^J%
229         \if@pstool@pdfcrop@else
230             \noexpand\end{preview}^^J%
231         \fi
232         \unexpanded{%

```

```

233         \vfill\end{document}}^~J%
234     }%
235     \immediate\closeout\pstool@out}

\pstool@cleanup 236 \def\pstool@cleanup{%
237     \@for\@ii:=\pstool@rm@files\do{%
238         \pstool@rm{\ip@directpath}{\pstool@jobname\@ii}%
239     }}

\EndPreamble 240 \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 241 \newcommand\psfragfig[2] [] {\pstool@psfragfig{#1}{#2}{}}
\psfragfig* 242 \WithSuffix\newcommand\psfragfig*[2] [] {\pstool@psfragfig{#1}{%
    #2}{*}}
\psfragfig 243 \WithSuffix\newcommand\psfragfig![2] [] {\pstool@psfragfig{#1}{%
    #2}{!}}

```

Parse optional *<input definitions>*

```

\pstool@psfragfig 244 \newcommand\pstool@psfragfig[3] {%
245     \@ifnextchar\bgroup{%
246         \pstool@@psfragfig{#1}{#2}{#3}%
247     }{%
248         \pstool@@psfragfig{#1}{#2}{#3}{}%
249     }%
250 }

```

Search for both *<filename>* and *<filename>-psfrag* inputs.

```

\pstool@@psfragfig 251 \newcommand\pstool@@psfragfig[4] {%
252     \IfFileExists{#2-psfrag.eps}{%
\pstool@eps 253     \def\pstool@eps{#2-psfrag}%
254     \OnlyIfFileExists{#2.eps}{%
255         \PackageWarning{pstool}{Graphic "#2.eps" exists but
            "#2-psfrag.eps" is being used}%

```



```

256     }%
257 }{%
258   \IfFileExists{#2.eps}{%
\pstool@eps 259     \def\pstool@eps{#2}%
260   }{%
261     \PackageError{pstool}{%
262       No graphic "#2.eps" or "#2-psfrag.eps" found%
263     }{%
264       Check the path and whether the file exists.%
265     }%
266   }%
267 }%
268 \pstool#3[#1]{\pstool@eps}{%
269   \InputIfFileExists{#2-psfrag.tex}{%
270     \OnlyIfFileExists{#2.tex}{%
271       \PackageWarning{pstool}{%
272         File "#2.tex" exists that may contain macros for "%
          \pstool@eps.eps"^^J%
273         But file "#2-psfrag.tex" is being used instead.%
274       }%
275     }%
276   }{%
277     \InputIfFileExists{#2.tex}{}{}%
278   }%
279   #4%
280 }%
281 }

```

<eof>