

The pstool package

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

vo.8 2008/08/28

Contents

I	USER DOCUMENTATION	1	II	IMPLEMENTATION	7
1	Introduction	1	6	Macros	8
2	Processing modes	2	7	Command parsing	12
3	Package options	3	8	User commands	12
4	Miscellaneous details	4	9	The figure processing	13
5	Package information	5	10	User commands	15

Part I

User documentation

1 Introduction

While pdfL^AT_EX 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 pdfL^AT_EX 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 pdfL^AT_EX 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.

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 pdfL^AT_EX.

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

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.

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 command, which also supports 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.

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. This corresponds with the package option `[crop=preview]`, which is activated by default.

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 the `epstool` method of robust graphic cropping.

¹If `pstool` is loaded in a L^AT_EX document in D_VI mode, this is the option that is used since no external processing is required for these graphics.

²`pdfcrop` requires a Perl installation under Windows, freely available from <http://www>.

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 a 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’.

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

activestate.com/Products/activeperl/index.plex

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 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 `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').

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

```

1 \ProvidesPackage{pstool}[2008/08/28_v0.8
2   Wrapper_for_processing_PostScript/psfrag_figures]

```

External packages

```

3 \RequirePackage{%
4
5   catchfile,color,ifpdf,ifplatform,graphicx,psfrag,suffix,xkeyval}
6 \RequirePackage{inversepath}[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}{}}

```

5.1 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 }

process 21 \define@choicekey*{pstool.sty}{process}[\@tempa\@tempb]{%
22   all,none,auto}{%
23   \ifcase\@tempb\relax
24     \@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}}

43 \ifshellescape\else
44 \ExecuteOptionsX{process=none}
45 \PackageWarning{pstool}{^^J\space\space%
46 Package_option_[process=none]_activated^^J\space\space
47 because_-shell-escape_is_not_enabled.^^J%
48 This_warning_occurred}
49 \fi

50 \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 51 \def\pstool@echo#1{%
52 \if@pstool@verbose@
53 \pstool@echo@verbose{#1}%
54 \fi}

55 \def\pstool@echo@verbose#1{%

```



```

\pstool@echo@verbose 56 \immediate\write18{echo"#1"}%
57 }

58 \let\pstool@includegraphics\includegraphics

```

Command line abstractions between platforms:

```

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

```

Delete a file if it exists:

```

\pstool@rm 61 \newcommand\pstool@rm[1]{%
62 \OnlyIfFileExists{\ip@directpath#1}{%
63 \immediate\write18{%
64 cd"\ip@directpath"\pstool@cmdsep\pstool@rm@cmd"#1"}}%
65 }

```

Generic function to execute a command on the shell and pass its exit status back into L^AT_EX. 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 66 \newcommand\pstool@exe[3]{%
67 \pstool@echo{^^J===_pstool:#1_===}%
68 \pstool@writestatus{#2}{#3}%
69 \pstool@retrievestatus{#2}%
70 \ifnum\pstool@status_>_ \z@
71 \PackageWarning{pstool}{Execution_failed_during_
process:^^J\space\space#3^^JThis_warning_occurred}%
72 \expandafter\pstool@abort
73 \fi}

```

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

```

\pstool@error 74 \def\pstool@error{\fbox{\parbox{0.8\linewidth}{\color{red}%
\raggedright\ttfamily\scshape\small
75 An_error_occured_processing_graphic_\upshape'\ip@directpath%
\ip@lastelement'}}}

76 \def\pstool@abort#1\pstool@endprocess{\pstool@error\@gobble}

```

```
\pstool@abort 77 \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_latex supported input pipes, things might be different.)

```
\pstool@writestatus 78 \def\pstool@writestatus#1#2{%
79   \immediate\write18{%
80     cd"#1"\pstool@cmdsep
81     #2\pstool@cmdsep
82     \ifwindows
83       call_echo
84       \string^ \@percentchar ERRORLEVEL\string^ \@percentchar
85     \else
86       echo\detokenize{$?}
87     \fi
88     >\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.

```
89   \ifwindows\else
90     \immediate\write18{%
91       touch"#1\pstool@statusfile}%
92   \fi}
\pstool@statusfile 93 \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 94 \def\pstool@retrievestatus#1{%
95   \CatchFileEdef{\pstool@status}{#1\pstool@statusfile}{}%
96   \pstool@rm{\pstool@statusfile}%
97   \ifx\pstool@status\pstool@statusfail
98     \PackageWarning{pstool}{%
99       Status of process unable to be determined: ^^J#1^^J%
100     Trying to proceed...}%
\pstool@status 101   \def\pstool@status{0}%
102   \fi}
103 \def\pstool@statusfail{\par}% what results when TEX reads an empty
```

\pstool@statusfail file

6.1 File age detection

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

```
\pstool@ifnewerEPS 104 \def\pstool@ifnewerEPS{%
105   \edef\pstool@filenames{\ip@lastelement.eps\space\%
        \ip@lastelement.pdf\space}%
106   \immediate\write18{%
107     cd"\ip@directpath"\pstool@cmdsep
108     \ifwindows
109       dir_/T:W_/B_/O-D"\ip@lastelement.eps" "%
        \ip@lastelement.pdf">\pstool@statusfile
110     \else
111       ls-t"\ip@lastelement.eps" "\ip@lastelement.pdf">\%
        \pstool@statusfile
112     \fi
113   }%
114   \pstool@retrievestatus{\ip@directpath}%
115   \ifx\pstool@status\pstool@filenames
116     \expandafter\@firstoftwo
117   \else
118     \expandafter\@secondoftwo
119   \fi
120 }
```

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 121 \def\pstool@getpaths#1{%
122   \edef\@tempa{\unexpanded{\inversepath*}{#1}}%
123   \@tempa% calculate filename, path & inverse path
124   \ifx\ip@directpath\@empty
\ip@directpath 125     \def\ip@directpath{./}%
126   \fi
```

Strip off a possible wayward `.eps` suffix.

```
127   \edef\ip@lastelement{%
128     \expandafter\pstool@stripEPS\ip@lastelement.eps\@nil
129   }%
```

```
130 }
```

```
\pstool@stripEPS 131 \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 132 \newcommand\pstool@alwaysprocess[3] [] {%
```

```
133 \pstool@getpaths{#2}%
```

```
134 \pstool@process{#1}{#3}}
```

```
\pstool@neverprocess 135 \newcommand\pstool@neverprocess[3] [] {%
```

```
136 \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 137 \newcommand\pstool@maybeprocess[3] [] {%
```

```
138 \pstool@getpaths{#2}%
```

```
139 \IfFileExists{#2.pdf}{%
```

```
140 \pstool@ifnewerEPS{% needs info from \pstool@getpaths
```

```
141 \pstool@process{#1}{#3}%
```

```
142 }{%
```

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

```
144 }%
```

```
145 }{%
```

```
146 \pstool@process{#1}{#3}%
```

```
147 }}
```

8 User commands

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

```
148 \ifpdf
```

```
149 \if@pstool@always@
```

```
150 \let\pstool\pstool@alwaysprocess
```

```
151 \WithSuffix\def\pstool!\{\pstool@alwaysprocess}
```

```

\pstool* 152 \WithSuffix\def\pstool*{\pstool@alwaysprocess}
153 \else\if@pstool@never@
154 \let\pstool\pstool@neverprocess
\pstool 155 \WithSuffix\def\pstool!{\pstool@neverprocess}
\pstool* 156 \WithSuffix\def\pstool*{\pstool@neverprocess}
157 \else
158 \let\pstool\pstool@maybeprocess
\pstool 159 \WithSuffix\def\pstool!{\pstool@neverprocess}
\pstool* 160 \WithSuffix\def\pstool*{\pstool@alwaysprocess}
161 \fi\fi
162 \else
163 \let\pstool\pstool@neverprocess
\pstool 164 \WithSuffix\def\pstool!{\pstool@neverprocess}
\pstool* 165 \WithSuffix\def\pstool*{\pstool@neverprocess}
166 \fi

```

9 The figure processing

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

```

\pstool@jobname 167 \def\pstool@jobname{\ip@lastelement\pstool@suffix}

```

And this is the main macro.

```

\pstool@process 168 \newcommand\pstool@process[2]{%
169 \pstool@echo@verbose{^^J===\pstool:\begin_processing===}%
170 \pstool@write@processfile{#1}{\ip@directpath%
\ip@lastelement}{#2}%
171 \pstool@exe{auxiliary_process:\ip@lastelement\space}
172 {./}{latex
173 -shell-escape
174 -output-format=dvi
175 -output-directory="\ip@directpath"
176 -interaction=\pstool@mode\space
177 "\pstool@jobname.tex"}}%

```

Execute dvips in quiet mode if latex is run in (non/error)stop mode:

```

178 \pstool@exe{dvips}{\ip@directpath}{%
179 dvips\if@pstool@verbose@\else-q\fi-Ppdf"%
\pstool@jobname.dvi"}%

```

```

180 \if@pstool@pdfcrop@
181 \pstool@exe{ps2pdf}{\ip@directpath}{%
182     ps2pdf_\pstool@jobname.ps_\pstool@jobname.pdf}%
183 \pstool@exe{pdfcrop}{\ip@directpath}{%
184     pdfcrop_\pstool@jobname.pdf_\ip@lastelement.pdf}%
185 \else
186 \pstool@exe{ps2pdf}{\ip@directpath}{%
187     ps2pdf_\pstool@jobname.ps_\ip@lastelement.pdf}%
188 \fi
189 \pstool@echo{^^J===_pstool:_end_processing_===^^J}%
190 \pstool@endprocess{%
191     \pstool@cleanup
192     \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 193 \def\pstool@write@processfile#1#2#3{%
194     \immediate\openout\pstool@out_\#2\pstool@suffix.tex\relax
195     \immediate\write\pstool@out{%
196         \noexpand\pdfoutput=0^^J% force DVI mode if not already

```

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

```

197     \unexpanded{%
198         \let\origdocument\document^^J%
199         \let\EndPreamble\endinput^^J%
\document 200     \def\document{\endgroup\endinput}^^J}%
201     \noexpand\input{\jobname}^^J%

```

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

```

202     \if@pstool@pdfcrop@\else
203         \noexpand\usepackage[active,tightpage]{preview}^^J%
204     \fi
205     \unexpanded{%
206         \let\document\origdocument^^J%
207         \pagestyle{empty}^^J}%

```

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

```

208     \unexpanded{%
209         \begin{document}}^^J%
210     \centering\null\vfill^^J}%
211     \if@pstool@pdfcrop@\else
212         \noexpand\begin{preview}^^J%
213     \fi
214     \unexpanded{#3^^J}% this is the "psfrag" material
215     \noexpand\includegraphics[#1]{\ip@lastelement}^^J%
216     \if@pstool@pdfcrop@\else
217         \noexpand\end{preview}^^J%
218     \fi
219     \unexpanded{%
220         \vfill\end{document}}^^J%
221     }%
222     \immediate\closeout\pstool@out}

\pstool@cleanup 223 \def\pstool@cleanup{%
224     \@for\@ii:=\pstool@rm@files\do{%
225         \pstool@rm{\pstool@jobname\@ii}%
226     }}

\EndPreamble 227 \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 228 \newcommand\psfragfig[2] [] {\pstool@psfragfig{#1}{#2}{}}
\psfragfig* 229 \WithSuffix\newcommand\psfragfig*[2] [] {\pstool@psfragfig{#1}{%
    #2}{*}}
\psfragfig 230 \WithSuffix\newcommand\psfragfig![2] [] {\pstool@psfragfig{#1}{%
    #2}{!}}

```

Parse optional *<input definitions>*

```

\pstool@psfragfig 231 \newcommand\pstool@psfragfig[3]{%
232     \ifnextchar\bgroup{%
233         \pstool@@psfragfig{#1}{#2}{#3}%
234     }{%

```

```

235 \pstool@@psfragfig{#1}{#2}{#3}{}%
236 }%
237 }

```

Search for both $\langle filename \rangle$ and $\langle filename \rangle$ -psfrag inputs.

```

\pstool@@psfragfig 238 \newcommand\pstool@@psfragfig[4]{%
239 \IfFileExists{#2-psfrag.eps}{%
\pstool@eps 240 \def\pstool@eps{#2-psfrag}%
241 \OnlyIfFileExists{#2.eps}{%
242 \PackageWarning{pstool}{Graphic"#2.eps" exists but
"#2-psfrag.eps" is being used}%
243 }%
244 }{%
245 \IfFileExists{#2.eps}{%
\pstool@eps 246 \def\pstool@eps{#2}%
247 }{%
248 \PackageError{pstool}{%
249 No graphic"#2.eps" or"#2-psfrag.eps" found%
250 }{%
251 Check the path and whether the file exists.%
252 }%
253 }%
254 }%
255 \pstool#3[#1]{\pstool@eps}{%
256 \InputIfFileExists{#2-psfrag.tex}{%
257 \OnlyIfFileExists{#2.tex}{%
258 \PackageWarning{pstool}{%
259 File"#2.tex" exists that may contain macros for"%
\pstool@eps.eps"^^J%
260 But file"#2-psfrag.tex" is being used instead.%
261 }%
262 }%
263 }{%
264 \InputIfFileExists{#2.tex}{}{}%
265 }%
266 #4%
267 }%
268 }

\langle eof \rangle

```