

# The pstool package

Concept by Zebb Prime  
Package by Will Robertson\*

v1.5b      2013/03/11

## 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	<b>1</b>	II	IMPLEMENTATION	<b>9</b>
1	Introduction	<b>1</b>	6	Package information	<b>9</b>
2	Getting started	<b>2</b>	7	Code	<b>9</b>
3	User commands	<b>2</b>	8	Macros	<b>13</b>
4	Package options	<b>3</b>	9	Command parsing	<b>16</b>
5	Miscellaneous details	<b>7</b>	10	User commands	<b>17</b>
			11	The figure processing	<b>18</b>
			12	User commands	<b>23</b>

## 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`.

---

\*wspr81@gmail.com

Until now, the best way to use these packages while running pdfL<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X producing DVI→PS→PDF. The resulting PDF versions of each graphic are then included into the pdfL<sup>A</sup>T<sub>E</sub>X 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 pdfL<sup>A</sup>T<sub>E</sub>X.

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

Processing pdfL<sup>A</sup>T<sub>E</sub>X documents with pstool requires the ‘shell escape’ feature of pdfT<sub>E</sub>X to be activated. This allows execution of auxiliary commands from within L<sup>A</sup>T<sub>E</sub>X, a feature which is often disabled by default for security reasons. If shell escape is not enabled, a warning will be issued in console output when the package is loaded. Depending how you compile your L<sup>A</sup>T<sub>E</sub>X document, shell escape is enabled in different ways.<sup>1</sup>

Load the package as usual; no package options are required by default, but there are a few useful options described later in section 4. Note that you do not need to load psfrag separately. You also do not need to load graphicx separately, but if you do so, ensure that you do *not* include driver information (such as `[pdftex]`); this will play havoc with pstool’s automatic processing stage.

## 3 User commands

The low-level generic command provided by this package is

```
\pstool <suffix> [<options>] {<filename>} {<input definitions>}
```

It converts the graphic `<filename>.eps` to `<filename>.pdf` with psfrag macros in `<filename>.tex` through a unique DVI→PS→PDF process for each graphic, using

---

<sup>1</sup>On the command line, use the `-shell-escape` switch. Otherwise, you’re on your own.

the preamble of the main document. The resulting graphic is then inserted into the document, with  $\langle options \rangle$  consisting of options for `graphicx` (e.g., `angle`, `scale`) or for `pstool` (as described later in Section 4). Note that these optional arguments take effect in the *processing stage*; if you change the  $\langle options \rangle$ , you must manually re-process the figure. The third argument to `\pstool` allows arbitrary  $\langle input definitions \rangle$  (such as `\psfrag` directives) to be inserted before the figure as it is processed.

By default, `\pstool` processes the graphic  $\langle filename \rangle$ .eps if  $\langle filename \rangle$ .pdf does not already exist, or if the EPS file is *newer* than the PDF. Additionally, if one or more macro files are associated with the graphic, they are also checked whether they have changed since the PDF was generated. The macro file(s) can be defined per-graphic as for the `\psfragfig` command (see below), and/or globally as for the `[macro-file=...]` package option described in Section 4.1.

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

`\pstool*` Always process the figure;  
`\pstool!` Never process the figure.

The behaviour in all three cases can be overridden globally by the package option `[process]` as described in section 4.2.

### 3.1 The main `\psfragfig` command

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

`\psfragfig`  $\langle suffix \rangle$  [ $\langle opts \rangle$ ] { $\langle filename \rangle$ }

This catch-all macro is designed to support a wide range of graphics naming schemes. It inserts an EPS file named either  $\langle filename \rangle$ -psfrag.eps or  $\langle filename \rangle$ .eps (in that order of preference), and uses `psfrag` definitions contained within either  $\langle filename \rangle$ -psfrag.tex or  $\langle filename \rangle$ .tex. The `\psfragfig` 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:

`\psfragfig`  $\langle suffix \rangle$  [ $\langle opts \rangle$ ] { $\langle filename \rangle$ } { $\langle input definitions \rangle$ }

The command behaves as above, but also inserts the arbitrary code  $\langle input definitions \rangle$  into the processing stage; this additional code will usually be used to define new or override existing `psfrag` commands.

## 4 Package options

Package options can be set or overridden at any time with `\pstoolsetup{<pstool settings>}`. As mentioned in the previous section, these options also may be set in the optional argument to `\pstool` and `\psfragfig`, in which case they apply to that figure alone.

### 4.1 Macro file(s)

**New in v1.5.** As mentioned above, macro files can be used to store commands for processing `psfrag` graphics. If they change, these macro files can trigger a pre-compilation of the graphics. While usually the macro files will be defined per-graphic (such as `foo.eps` having a `foo-psfrag.tex` file), `pstool` will also load a ‘master’ macro file for each graphic if it exists.

`[macro-file=...]`

The default is `[macro-file=<jobname>-pstool.tex]`; if this file does not exist then no macro file is loaded. That is, if your document is called `thesis.tex`, the master macro file will be loaded in each graphic as `thesis-pstool.tex`, if it exists.

This option is useful if you have macro definitions in a single file that are used by multiple graphics. By updating the definitions file, the graphics in the document will be automatically updated. (Note that this file can contain plain  $\text{\LaTeX}$  definitions; the `\psfrag` commands can still be located in the per-graphic `.tex` files.)

To suppress the loading of a master macro file in all cases, use an empty argument for the package option, as in `[macro-file={}]`.

### 4.2 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` macros:

`[process=auto]` This is the default mode as described in the previous section, in which graphics without 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.<sup>2</sup>

---

<sup>2</sup>If `pstool` is loaded in a  $\text{\LaTeX}$  document in `DVI` mode, this is the option that is used since no external processing is required for these graphics.

### 4.3 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.<sup>3</sup> This can be activated in `pstool` with the `[crop=pdfcrop]` package option.

### 4.4 Temporary files & cleanup

Each figure that is processed spawns an auxiliary  $\text{\LaTeX}$  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}]`. Most 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}]`. To delete none of the temporary files, choose `[cleanup={}]` (useful for debugging). Note that if you want cross-referencing to work correctly for labels in figures, etc., then you must not delete the `.aux` file (see Section 5.2).

### 4.5 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  $\text{\LaTeX}$  output (*default*);  
`[mode=nonstop]` echo all  $\text{\LaTeX}$  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  $\text{\LaTeX}$  command line options `-interaction=batchmode`, `=nonstopmode`, and `=errorstopmode`, respectively. When `[mode=batch]` is activated, then `dvips` is also run in ‘quiet mode’.

---

<sup>3</sup>`pdfcrop` requires a Perl installation under Windows, freely available from <http://www.activestate.com/Products/activeperl/index.plex>

## 4.6 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=...] (if applicable).
```

For the most part these will be unnecessary, although passing the correct options to ps2pdf can sometimes be a little obscure.<sup>4</sup> For example, I use the following for generating figures in my thesis:

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

This forces the ‘base fourteen’ fonts to be embedded within the individual figure files, without which some printers and PDF viewers have trouble with the textual labels. In fact, from v1.3 of pstool, this option is now the default. Note that subsequent calls to [ps2pdf-options=...] will override the pstool default; use ps2pdf-options={} to erase ps2pdf’s defaults if necessary.

**New in 1.5:** recently, the behaviour of ps2pdf has changed under Windows. In the past, options to ps2pdf needed to be quoted and use = to assign its options. Something about this has now changed, and it appears the best way to set ps2pdf options to use the # character instead; therefore, pstool attempts to be clever and replaces all instances of = within a ps2pdf option into # (under Windows only). No quotes are added. Windows users can therefore continue to use = to set ps2pdf options and allow pstool to make the substitution; their documents will still compile correctly on Mac OS X or Linux platforms.

## 4.7 Compression of bitmap data

In the conversion using ps2pdf, bitmap images are stored using either lossy or lossless compression. The default behaviour for pstool is to force lossless compression, because we believe that to be the most commonly desired use case (you don’t want scientific graphics rendered with possible compression artifacts). This behaviour can be adjusted using one of these options:<sup>5</sup>

```
[bitmap=auto] : Do whatever ps2pdf does by default, which seems to be to use  
                lossy compression most, if not all, of the time;  
[bitmap=lossy] : Bitmap images are compressed like JPG; this is only really  
                suitable for photographs;
```

---

<sup>4</sup>The manual is here: <http://pages.cs.wisc.edu/~ghost/doc/cvs/Ps2pdf.htm>

<sup>5</sup>Technical details are given in section 5.4.

`[bitmap=lossless]` (*default*) : Bitmap images are compressed like PNG; this is suitable for screenshots and generated data such as a surface plot within Matlab.

These are just special cases of the `[ps2pdf-options=...]` option, but using `[bitmap=...]` is much more convenient since the `ps2pdf` options to effect this behaviour are quite verbose. Note that the `auto` and `lossy` outputs differ in quality; `lossy` is lower quality than `auto` even when the latter uses a lossy compression scheme. Adjusting the quality for these options is only possible with relatively complex Ghostscript options.

## 5 Miscellaneous details

### 5.1 The `\EndPreamble` command

The `pstool` package scans the beginning of the main document to insert its preamble into each graphic that is converted. This feature hasn't been well-tested and there are certain cases in which it is known to fail. (For example, if `\begin{document}` doesn't appear on a line by itself.) If you need to indicate the end of the preamble manually because this scanning has failed, 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.

### 5.2 Cross-referencing

**New in v1.5:** `pstool` now supports cross-referencing within graphics. That is, you can use `\ref` and `\cite`, etc., within `psfrag` commands. In fact, references to page numbers within an external figure should now resolve correctly; e.g., you can use `\thepage` within a `psfrag` command. (I haven't really tested, but this should allow any package that writes information to the `.aux` file and refers to the page number to work correctly.)

The implementation to achieve this is somewhat convoluted and difficult to extend, but the user interface should work just as you would expect, mostly. The main gotcha to keep in mind is that when cross-referencing is used, the graphics will need multiple compilations to resolve all the cross-references properly. Therefore, I recommend when setting such figures up in your document to use the `\psfragfig*` command, which forces graphics compilation every time, and remove the star only when you're sure the graphic is now correct. Alternatively, don't worry about the resolving of the cross-references until the very end, and then load the package with the `[process=all]` option.

### 5.3 A note on file paths

pstool tries to ensure that you can put image files in subdirectories of the main document and the auxiliary processing will still function correctly. In order to ensure this, the external pdf<sub>l</sub>atex compilation uses the `-output-directory` feature of pdf<sub>l</sub>TeX. This command line option is definitely supported on all platforms from TeX Live 2008 and MiKTeX 2.7 onwards, but earlier distributions may not be supported.

One problem that pstool does not solve on its own is the inclusion of images that do not exist in subdirectories of the main document. For example, `\pstool{../Figures/myfig}` can not process by default because pdf<sub>l</sub>TeX 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<sub>l</sub>atex 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.

### 5.4 Technical details on ps2pdf's bitmap options

The `[bitmap=auto]` pstool option does not set any ps2pdf options; use this if you wish to set the following ps2pdf options manually.

For both `[bitmap=lossless]` (default) and `[bitmap=lossy]`, the following ps2pdf options are set:

```
-dAutoFilterColorImages=false
-dAutoFilterGrayImages=false
```

Then for lossless image encoding, the following options are set:

```
-dColorImageFilter=/FlateEncode
-dGrayImageFilter=/FlateEncode
```

Instead for lossy encoding, these are the options used:

```
-dColorImageFilter=/DCTEncode
-dGrayImageFilter=/DCTEncode
```

If there are more ps2pdf options that you frequently use, please let me know and it may be a good idea to add pstool wrappers to make them more convenient.



## Part II

# Implementation

## 6 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 via email is welcomed, specific bugs or feature requests should be reported through the issue tracker: <http://github.com/wspr/pstool/issues>.

### 6.1 Licence

This package is freely modifiable and distributable under the terms and conditions of the L<sup>A</sup>T<sub>E</sub>X Project Public Licence, version 1.3c or greater (your choice).<sup>6</sup> This work consists of the files `pstool.tex` and the derived files `pstool.sty`, `pstool.ins`, and `pstool.pdf`. This work is maintained by WILL ROBERTSON.

## 7 Code

Note that the following code is typeset in a non-verbatim manner; indentation is controlled automatically by some hastily written macros (and will sometimes not indent as might be done manually). Furthermore, spaces before comment signs are lost—when in doubt, consult the source directly!

TODO: convert this package into `expl3` syntax (will save many lines of code).

```
3 \ProvidesPackage{pstool}[2013/03/11 v1.5b
4 Wrapper for processing PostScript/psfrag figures]
```

External packages:

```
7 \RequirePackage{
8   catchfile,color,ifpdf,ifplatform,filemod,
9   graphicx,psfrag,suffix,trimspaces,xkeyval,expl3
10 }
```

Add an additional command before `trimspaces.sty` is updated formally:

```
13 \providecommand*\trim@multiple@spaces@in}[1]{%
14   \let\trim@temp#1%
15   \trim@spaces@in#1%
```

---

<sup>6</sup><http://www.latex-project.org/lppl.txt>

```

16 \ifx\trim@temp#1%
17   \else
18   \expandafter\trim@multiple@spaces@in\expandafter#1%
19 \fi
20 }

```

## 7.1 Allocations

```

23 \expandafter\newif\csname if@pstool@pdfcrop@\endcsname
24 \expandafter\newif\csname if@pstool@verbose@\endcsname
25 \expandafter\newif\csname if@pstool@write@aux\endcsname

27 \newwrite\pstool@out
28 \newread\pstool@mainfile@ior
29 \newread\pstool@auxfile@ior

```

Macro used to store the name of the graphic's macro file:

```

32 \let\pstool@tex\@empty

```

## 7.2 Package options

```

36 \define@choicekey*{pstool.sty}{crop}
37 [\@tempa\@tempb]{preview,pdfcrop}{%
38   \ifcase\@tempb\relax
39     \@pstool@pdfcrop@false
40   \or
41     \@pstool@pdfcrop@true
42   \or
43   \fi
44 }

46 \define@choicekey*{pstool.sty}{process}
47 [\@tempa\pstool@process@choice]{all,none,auto}{}
48 \ExecuteOptionsX{process=auto}

50 \define@choicekey*{pstool.sty}{mode}
51 [\@tempa\@tempb]{errorstop,nonstop,batch}{%
52   \ifnum\@tempb=2\relax
53     \@pstool@verbose@false
54   \else

```

```

55     \@pstool@verbose@true
56   \fi
57   \edef\pstool@mode{\@tempa mode}%
58 }
59 \ExecuteOptionsX{mode=batch}

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

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

    There is an implicit \space after the bitmap options.
68 \define@choicekey*{pstool.sty}{bitmap}
69 [\@tempa\@tempb]{auto,lossless,lossy}{%
70   \ifcase\@tempb
71     \let\pstool@bitmap@opts\@empty
72   \or
73     \def\pstool@bitmap@opts{%
74       -dAutoFilterColorImages=false
75       -dAutoFilterGrayImages=false
76       -dColorImageFilter=/FlateEncode
77       -dGrayImageFilter=/FlateEncode% space
78     }
79   \or
80     \def\pstool@bitmap@opts{%
81       -dAutoFilterColorImages=false
82       -dAutoFilterGrayImages=false
83       -dColorImageFilter=/DCTEncode
84       -dGrayImageFilter=/DCTEncode% space
85     }
86   \fi
87 }
88 \ExecuteOptionsX{bitmap=lossless}

90 \DeclareOptionX{latex-options}{\def\pstool@latex@opts{#1}}
91 \DeclareOptionX{dvips-options}{\def\pstool@dvips@opts{#1}}
92 \DeclareOptionX{ps2pdf-options}{\def\pstool@pspdf@opts{#1}}
93 \DeclareOptionX{pdfcrop-options}{\def\pstool@pdfcrop@opts{#1}}

95 \ExecuteOptionsX{

```

```

96 latex-options={},
97 dvips-options={},
98 ps2pdf-options={-dPDFSETTINGS=/prepress},
99 pdfcrop-options={}
100 }

102 \DeclareOptionX{macro-file}{%
103   \IfFileExists{#1}
104   {\def\pstool@macrofile{#1}}
105   {%
106     \let\pstool@macrofile\@empty
107     \PackageError{pstool}{^^J\space\space%
108       No file ‘#1’ found for "macro-file" package option.^^J
109       This warning occurred}
110   }
111 }

Default:
114 \IfFileExists{\jobname-pstool.tex}
115 {\edef\pstool@macrofile{\jobname-pstool.tex}}
116 {\let\pstool@macrofile\@empty}

119 \ifpdf
120   \ifshellescape\else
121     \ExecuteOptionsX{process=none}
122     \PackageWarning{pstool}{^^J\space\space%
123       Package option [process=none] activated^^J\space\space
124       because -shell-escape is not enabled.^^J%
125       This warning occurred}
126   \fi
127 \fi

129 \ProcessOptionsX

A command to set pstool options after the package is loaded.
132 \newcommand\pstoolsetup{%
133   \setkeys{pstool.sty}%
134 }

```

## 8 Macros

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

```
140 \def\pstool@echo#1{%
141   \if@pstool@verbose@
142     \pstool@echo@verbose{#1}%
143   \fi
144 }

146 \def\pstool@echo@verbose#1{%
147   \immediate\write18{echo "#1"}%
148 }

150 \let\pstool@includegraphics\includegraphics
```

Command line abstractions between platforms:

```
153 \edef\pstool@cmdsep{\ifwindows\string&\else\string;\fi\space}
154 \edef\pstool@rm@cmd{\ifwindows del \else rm - \fi}
155 \edef\pstool@cp@cmd{\ifwindows copy \else cp - \fi}
```

Delete a file if it exists:

#1: path

#2: filename

```
160 \newcommand\pstool@rm[2]{%
161   \IfFileExists{#1#2}{%
162     \immediate\write18{%
163       cd "#1"\pstool@cmdsep\pstool@rm@cmd "#2"
164     }%
165   }{}%
166 }
```

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

```

172 \newcommand\pstool@exe[3]{%
173   \pstool@echo{^^J=== pstool: #1 ===}%
174   \pstool@shellexecute{#2}{#3}%
175   \pstool@retrievestatus{#2}%
176   \ifnum\pstool@status > \z@
177     \PackageWarning{pstool}{%
178       Execution failed during process:^^J\space\space
179       #3^^JThis warning occurred%
180     }%
181     \expandafter\pstool@abort
182   \fi
183 }

```

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

```

186 \def\pstool@error{%
187   \fbox{%
188     \parbox{0.8\linewidth}{%
189       \color{red}\centering\ttfamily\scshape\small
190       An error occured processing graphic:\\
191       \upshape'%
192       \detokenize\expandafter{\pstool@path}%
193       \detokenize\expandafter{\pstool@filestub}.eps%
194       '\\\bigskip
195       \tiny
196       Check the log file for compilation errors:\\
197       '%
198       \detokenize\expandafter{\pstool@path}%
199       \detokenize\expandafter{\pstool@filestub}-pstool.log%
200       '\\
201     }%
202   }%
203 }

205 \def\pstool@abort#1\pstool@endprocess{\pstool@error\@gobble}
206 \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<sub>l</sub>at<sub>e</sub>x supported input pipes, things might be different.)

```

209 \def\pstool@shellexecute#1#2{%
210   \immediate\write18{%

```

```

211     cd "#1" \pstool@cmdsep
212     #2 \pstool@cmdsep
213     \ifwindows
214         call echo
215         \string^ \@percentchar ERRORLEVEL\string^ \@percentchar
216     \else
217         echo \detokenize{${?}}
218     \fi
219 > \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.

```

221     \ifwindows\else
222         \immediate\write18{%
223         touch #1\pstool@statusfile}%
224     \fi
225 }
226 \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`.

```

231 \def\pstool@retrievestatus#1{%
232     \CatchFileEdef{\pstool@status}{#1\pstool@statusfile}{}%
233     \pstool@rm{#1}{\pstool@statusfile}%
234     \ifx\pstool@status\pstool@statusfail
235         \PackageWarning{pstool}{%
236             Status of process unable to be determined:^^J  #1^^J%
237             Trying to proceed... }%
238     \def\pstool@status{0}%
239     \fi
240 }
241 \def\pstool@statusfail{\par }% what results when TeX reads an empty file

```

Grab filename and filepath. Always need a relative path to a filename even if it's in the same directory.

```

244 \def\pstool@getpaths#1{%
245     \filename@parse{#1}%
246     \ifx\filename@area\@empty
247     \def\pstool@path{./}%
248     \else

```

```

249 \let\pstool@path\filename@area
250 \fi
251 \let\pstool@filestub\filename@base
252 }

```

The filename of the figure stripped of its path, if any:  
(analogous to standard `\jobname`)

```

256 \def\pstool@jobname{\pstool@filestub\pstool@suffix}

```

## 9 Command parsing

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

```

262 \newcommand\pstool@alwaysprocess[3][]{\%
263   \pstool@getpaths{#2}%
264   \pstool@process{#1}{#3}%
265 }

267 \ifpdf
268   \newcommand\pstool@neverprocess[3][]{\%
269     \pstool@includegraphics{#2}%
270   }
271   \else
272     \newcommand\pstool@neverprocess[3][]{\%
273       \begingroup
274       \setkeys*{pstool.sty}{#1}%
275       #3%
276       \expandafter\pstool@includegraphics\expandafter[\XKV@rm]{#2}%
277     \endgroup
278   }
279 \fi

```

Process the figure when:

- the PDF file doesn't exist, or
- the EPS is newer than the PDF, or
- the TeX file is new than the PDF.

```

285 \ExplSyntaxOn
286 \newcommand\pstool@maybeprocess[3]{}
287 {

```



```

288 \pstool_if_should_process:nTF {#2}
289 { \pstool@process{#1}{#3} }
290 { \pstool@includegraphics{#2} }
291 }

293 \prg_set_conditional:Nnn \pstool_if_should_process:n {TF}
294 {
295   \pstool@getpaths{#1}

297   \file_if_exist:nF { #1.pdf }
298   { \use_i_delimit_by_q_stop:nw \prg_return_true: }

300   \filemodCmp {\pstool@path\pstool@filestub.eps}
301   {\pstool@path\pstool@filestub.pdf}
302   { \use_i_delimit_by_q_stop:nw \prg_return_true: } {}

304   \exp_args:Nx \clist_map_inline:nn { \pstool@macrofile , \pstool@tex }
      empty entries are ignored in clist mappings, so no need to filter here
306   {
307     \filemodCmp {##1} {\pstool@path\pstool@filestub.pdf}
308     {
309       \clist_map_break:n { \use_i_delimit_by_q_stop:nw \prg_return_true: }
310     }
311   } {}
312 }

314 \filemodCmp {\pstool@path\pstool@filestub.tex}
315 {\pstool@path\pstool@filestub.pdf}
316 { \use_i_delimit_by_q_stop:nw \prg_return_true: } {}

318 \use_i_delimit_by_q_stop:nw \prg_return_false:
319 \q_stop
320 }
321 \ExplSyntaxOff

```

## 10 User commands

Finally, define `\pstool` as appropriate for the mode: (all, none, auto, respectively)

```

325 \ifpdf
326   \newcommand\pstool{%

```

```

327 \ifcase\pstool@process@choice\relax
328 \expandafter \pstool@alwaysprocess \or
329 \expandafter \pstool@neverprocess \or
330 \expandafter \pstool@maybeprocess
331 \fi
332 }
333 \WithSuffix\def\pstool!{%
334 \ifcase\pstool@process@choice\relax
335 \expandafter \pstool@alwaysprocess \or
336 \expandafter \pstool@neverprocess \or
337 \expandafter \pstool@neverprocess
338 \fi
339 }
340 \WithSuffix\def\pstool*{%
341 \ifcase\pstool@process@choice\relax
342 \expandafter \pstool@alwaysprocess \or
343 \expandafter \pstool@neverprocess \or
344 \expandafter \pstool@alwaysprocess
345 \fi
346 }
347 \else
348 \let\pstool\pstool@neverprocess
349 \WithSuffix\def\pstool!\{\pstool@neverprocess}
350 \WithSuffix\def\pstool*\{\pstool@neverprocess}
351 \fi

```

## 11 The figure processing

And this is the main macro.

```

356 \newcommand\pstool@process[2]{%
357 \begingroup
358 \setkeys*{pstool.sty}{#1}%
359 \pstool@echo@verbose{%
360 ^^J^^J=== pstool: begin processing ===}%
361 \pstool@write@processfile{#1}
362 {\pstool@path\pstool@filestub}{#2}%
363 \pstool@exe{auxiliary process: \pstool@filestub\space}
364 {./}{latex
365 -shell-escape
366 -output-format=dvi

```

```

367     -output-directory="\pstool@path"
368     -interaction=\pstool@mode\space
369     \pstool@latex@opts\space
370     "\pstool@jobname.tex"}%
Execute dvips in quiet mode if latex is not run in (non/error)stop mode:
372     \pstool@exe{dvips}{\pstool@path}{%
373         dvips \if@pstool@verbose@else -q \fi -Ppdf
374         \pstool@dvips@opts\space "\pstool@jobname.dvi"}%
Pre-process ps2pdf options for Windows (sigh):
376     \pstool@pspdf@opts@preprocess \pstool@bitmap@opts
377     \pstool@pspdf@opts@preprocess \pstool@pspdf@opts
Generate the PDF:
379     \if@pstool@pdfcrop@
380         \pstool@exe{ps2pdf}{\pstool@path}{%
381             ps2pdf \pstool@bitmap@opts \pstool@pspdf@opts \space
382             "\pstool@jobname.ps" "\pstool@jobname.pdf"}%
383         \pstool@exe{pdfcrop}{\pstool@path}{%
384             pdfcrop \pstool@pdfcrop@opts\space
385             "\pstool@jobname.pdf" "\pstool@filestub.pdf"}%
386         \else
387             \pstool@exe{ps2pdf}{\pstool@path}{%
388                 ps2pdf \pstool@bitmap@opts \pstool@pspdf@opts \space
389                 "\pstool@jobname.ps" "\pstool@filestub.pdf"}%
390         \fi
Finish up: (implies success!)
392     \pstool@endprocess{%
393         \pstool@includegraphics{\pstool@path\pstool@filestub}%
Emulate \include (sort of) and have the main document load the auxiliary aux file, in a
manner of speaking:
395         \pstool@write@aux
396         \pstool@cleanup
397     }%
398     \pstool@echo@verbose{^^J=== pstool: end processing ===^^J}%
399     \endgroup
400 }

402 \newcommand\pstool@write@aux{%
403     \endlinechar=-1
404     \@tempwattrue
405     \@pstool@write@auxfalse
406     \in@false

```

```

407 \openin \pstool@auxfile@ior "\pstool@path\pstool@jobname.aux"\relax
408 \@whilesw \if@tempswa \fi {%
409   \readline \pstool@auxfile@ior to \@tempa
410   \ifeof \pstool@auxfile@ior
411     \@tempswafalse
412   \else
413     \edef\@tempb{\detokenize\expandafter{\pstool@auxmarker@text/}}%
414     \ifx\@tempa\@tempb
415       \@tempswafalse
416     \else
417       \if@pstool@write@aux
418         \immediate\write\@mainaux{\unexpanded\expandafter{\@tempa}}%
419       \fi
420       \edef\@tempb{\detokenize\expandafter{\pstool@auxmarker@text*}}%
421       \ifx\@tempa\@tempb
422         \@pstool@write@auxtrue
423       \fi
424     \fi
425   \fi
426 }
427 \closein \pstool@auxfile@ior
428 }

430 \ExplSyntaxOn
431 \cs_new:Npn \pstool@pspdf@opts@preprocess #1
432 {
433   \ifwindows
434     \exp_args:NNnx \tl_replace_all:Nnn #1 {=} { \cs_to_str:N \# }
435   \fi
436 }
437 \ExplSyntaxOff

```

For what's coming next.

```

440 \edef\@begindocument@str{\detokenize\expandafter{\string\begin{document}}}
441 \edef\@endpreamble@str{\string\EndPreamble}
442 \def\in@first#1#2{\in@{NEVEROCCUR!#1}{NEVEROCCUR!#2}}

```

We need to cache the aux file, so here goes.

This is because the aux file is cleared for writing after \begindocument.

```

446 \ifpdf
447   \IfFileExists{\jobname.oldaux}{\immediate\write18{\pstool@rm@cmd \jobname.oldaux}}{}

```

```

448 \immediate\write18{\pstool@cp@cmd \jobname.aux \jobname.oldaux}
449 \AtEndDocument{\immediate\write18{\pstool@rm@cmd \jobname.oldaux}}
450 \fi

452 \edef\pstool@auxmarker#1{\string\@percentchar\space <#1PSTOOLLABELS>}
453 \edef\pstool@auxmarker@text#1{\@percentchar <#1PSTOOLLABELS>}

```

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).

```

456 \def\pstool@write@processfile#1#2#3{%
457 \immediate\openout\pstool@out #2\pstool@suffix.tex\relax
  Put down a label so we can pass through the current page number:
459 \edef\pstool@label{\pstool-\pstool@path\pstool@filestub}%
460 \protected@write\@auxout{%
461 {\string\newlabel{\pstool@label}{\@currentlabel}{\the\c@page}}}%
  And copy the main file's bbl file too: (necessary only for biblatex)
463 \IfFileExists{\jobname.bbl}{%
464 \IfFileExists{\pstool@path\pstool@jobname.bbl}{%
465 \immediate\write18{\pstool@rm@cmd \pstool@path\pstool@jobname.bbl}%
466 }{%
467 \immediate\write18{\pstool@cp@cmd \jobname.bbl \pstool@path\pstool@jobname.bbl}%
468 }{}%
  Scan the main document line by line; print preamble into auxiliary file until the document
  begins or \EndPreamble is found:
470 \endlinechar=-1
471 \def\@tempa{\pdfoutput=0\relax}%
472 \in@false
473 \openin\pstool@mainfile@ior "\jobname"\relax
474 \@whilesw \unless\ifin@ \fi {%
475 \immediate\write\pstool@out{\unexpanded\expandafter{\@tempa}}%
476 \readline\pstool@mainfile@ior to\@tempa
477 \let\@tempc\@tempa
478 \trim@multiple@spaces@in\@tempa
479 \expandafter\expandafter\expandafter\in@first
480 \expandafter\expandafter\expandafter{%
481 \expandafter\@begindocument@str
482 \expandafter}%
483 \expandafter{\@tempa}%
484 \unless\ifin@
485 \expandafter\expandafter\expandafter\in@first

```

```

486     \expandafter\expandafter\expandafter{%
487     \expandafter\@endpreamble@str
488     \expandafter}%
489     \expandafter{\@tempa}%
490 \fi
491 }
492 \closein\pstool@mainfile@ior
Now the preamble of the process file:
494 \immediate\write\pstool@out{%
495     \if@pstool@pdfcrop@\else
496     \noexpand\usepackage[active,tightpage]{preview}^^J%
497 \fi
498 \unexpanded{%
499     \pagestyle{empty}^^J^^J% remove the page number
500 }%
501 \noexpand\makeatletter^^J%

```

Sort out the page numbering here.

Force the pagestyle locally to output an integer so it can be written to the external file inside a \setcounter command.

```

504     \expandafter\ifx\csname r@\pstool@label\endcsname\relax\else
505     \def\noexpand\thepage{\unexpanded\expandafter{\thepage}}^^J%
506     \noexpand\setcounter{page}{%
507     \expandafter\expandafter\expandafter
508     \@secondoftwo\csname r@\pstool@label\endcsname
509     }^^J%
510 \fi

```

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

```

512 \noexpand\@input{\jobname.oldaux}^^J%
513 \noexpand\makeatother^^J^^J%
514 \noexpand\begin{document}^^J%
515 \noexpand\makeatletter^^J%
516 \unexpanded{\immediate\write\@mainaux}{\pstool@auxmarker*}^^J%
517 \noexpand\makeatother^^J^^J%
518 \unexpanded{%
519     \centering\null\vfill^^J%
520 }%
521 ^^J%
522 \if@pstool@pdfcrop@\else
523     \noexpand\begin{preview}^^J%
524 \fi
525 \unexpanded{#3^^J}

```

```

526 \noexpand\includegraphics
527 [\unexpanded\expandafter{\XKV@rm}]
528 {\pstool@filestub}^^J%
529 \if@pstool@pdfcrop@else
530 \noexpand\end{preview}^^J%
531 \fi
532 ^^J%
533 \unexpanded{\vfill^^J^^J\makeatletter^^J\immediate\write\@mainaux}{\pstool@auxmarker
534 \unexpanded{\makeatother^^J\end{document}}^^J%
535 }%
536 \immediate\closeout\pstool@out
537 }

539 \def\pstool@cleanup{%
540 \@for\@ii:=\pstool@rm@files\do{%
541 \pstool@rm{\pstool@path}{\pstool@jobname\@ii}%
542 }%
543 }

545 \providecommand\EndPreamble{}

```

## 12 User commands

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

For EPS figures with psfrag:

```

551 \newcommand\psfragfig[2] [] {\pstool@psfragfig{#1}{#2}{}}
552 \WithSuffix\newcommand\psfragfig*[2] [] {%
553 \pstool@psfragfig{#1}{#2}{*}%
554 }
555 \WithSuffix\newcommand\psfragfig![2] [] {%
556 \pstool@psfragfig{#1}{#2}{!}%
557 }

```

Parse optional input definitions:

```

560 \newcommand\pstool@psfragfig[3] {%
561 \@ifnextchar\bgroup{%
562 \pstool@psfragfig{#1}{#2}{#3}%
563 }{%
564 \pstool@psfragfig{#1}{#2}{#3}{}%

```

```

565 }%
566 }

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 (or other) macros
574 \newcommand\pstool@@psfragfig[4]{%
Find the .eps file to use.
576 \IfFileExists{#2-psfrag.eps}{%
577 \edef\pstool@eps{#2-psfrag}%
578 \IfFileExists{#2.eps}{%
579 \PackageWarning{pstool}{%
580 Graphic "#2.eps" exists but "#2-psfrag.eps" is being used%
581 }%
582 }{}%
583 }{%
584 \IfFileExists{#2.eps}{%
585 \edef\pstool@eps{#2}%
586 }{%
587 \PackageError{pstool}{%
588 No graphic "#2.eps" or "#2-psfrag.eps" found%
589 }{%
590 Check the path and whether the file exists.%
591 }%
592 }%
593 }%
Find the .tex file to use.
595 \IfFileExists{#2-psfrag.tex}{%
596 \edef\pstool@tex{#2-psfrag.tex}%
597 \IfFileExists{#2.tex}{%
598 \PackageWarning{pstool}{%
599 File "#2.tex" exists that may contain macros
600 for "\pstool@eps.eps"^^J%
601 But file "#2-psfrag.tex" is being used instead.%
602 }%
603 }{}%
604 }{%
605 \IfFileExists{#2.tex}{%
606 \edef\pstool@tex{#2.tex}%

```



```

607     }{%
608     \PackageWarning{pstool}{%
609         No file "#2.tex" or "#2-psfrag.tex" can be found
610         that may contain macros for "\pstool@eps.eps"%
611     }%
612 }%
613 }%

```

Perform the actual processing step, skipping it entirely if an EPS file hasn't been found.  
(In which case an error would have been called above; this is to clean up nicely in scrollmode, for example.)

```

616 \ifx\pstool@eps\@undefined\else
617 \edef\@tempa{%
618     \unexpanded{\pstool#3[#1]}\{\pstool@eps}{%
619         \ifx\pstool@macrofile\@empty\else
620             \unexpanded{\csname @input\endcsname}\{\pstool@macrofile}%
621         \fi
622         \ifx\pstool@tex\@empty\else
623             \unexpanded{\csname @input\endcsname}\{\pstool@tex}%
624         \fi
625         \unexpanded{#4}%
626     }%
627 }\@tempa
628 \fi
629 }

```

—The End—