The pstool package

Concept by Zebb Prime
Package by Will Robertson*

v1.5c 2014/05/11

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.

*wspr81@gmail.com

Until now, the best way to use these packages while running pdfLaTeX has been to use the pst-pdf package, which processes the entire document through a filter, sending 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 are compatible, but **only** if epstopdf is loaded first).

2 Getting started

Processing pdfLaTeX documents with pstool requires the 'shell escape' feature of pdfTeX to be activated. This allows execution of auxiliary commands from within LaTeX, 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 LaTeX document, shell escape is enabled in different ways.[1]

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

[1] 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 ⟨*options*⟩ 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 ⟨*options*⟩, you must manually re-process the figure. 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` processes the graphic ⟨*filename*⟩`.eps` if ⟨*filename*⟩`.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` ⟨***suffix***⟩ **`[`**⟨***opts***⟩**`]`** **`{`**⟨***filename***⟩**`}`**

This catch-all macro is designed 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`. 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` ⟨***suffix***⟩ **`[`**⟨***opts***⟩**`]`** **`{`**⟨***filename***⟩**`}`** **`{`**⟨***input definitions***⟩**`}`**

The command behaves as above, but also inserts the arbitrary code ⟨*input definitions*⟩ 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 LaTeX defintions; 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.[2]

[2] 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.

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.[3] 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 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}]`. 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 5.3).

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 LaTeX output (*default*);
`[mode=nonstop]` echo all LaTeX output but continues right past any errors; and
`[mode=errorstop]` prompt for user input when errors 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'.

[3] 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.[4] For example, I used 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 uses 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:[5]

`[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;

[4] The manual is here: http://pages.cs.wisc.edu/~ghost/doc/cvs/Ps2pdf.htm

[5] Technical details are given in section 5.5.

`[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 Conditional preamble or setup commands

It can be necessary to use a slightly different preamble for the main document compared to the auxiliary file used to process each graphic individually. To have preamble material be directed at only one or the other, use the `\ifpdf` command (automatically loaded from the ifpdf package) as follows:

```
\ifpdf
  % main preamble only
\else
  % graphics preamble only
\fi
```

For example, when using beamer and showing navigation symbols on each slide, you want to suppress these in the pstool-generated graphics (else they'll show up twice!). In this case, the preamble snippet would look something like:

```
\ifpdf\else
  \setbeamertemplate{navigation symbols}{}
\fi
```

It would be possible to provide specific pstool commands or environments to do this; let me know if the `\ifpdf` approach doesn't work for you. For larger amount of preamble material that should be omitted for each graphic, the `\EndPreamble` command (see next) might also help.

5.2 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.

5.3 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 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.

5.4 A note on file paths

The pstool package 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 `pdflatex` compilation uses the `-output-directory` feature of pdfTEX. 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 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.

5.5 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 lossly 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 LaTeX Project Public Licence, version 1.3c or greater (your choice).[6] 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). When in doubt, consult the source directly! TODO: convert this package into expl3 syntax (will save many lines of code).

```
\ProvidesPackage{pstool}[2014/05/11 v1.5c
  Wrapper for processing PostScript/psfrag figures]
```

External packages:

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

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

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

[6] http://www.latex-project.org/lppl.txt

```
      \else
        \expandafter\trim@multiple@spaces@in\expandafter#1%
    \fi
}
```

7.1 Allocations

```
\expandafter\newif\csname if@pstool@pdfcrop@\endcsname
\expandafter\newif\csname if@pstool@verbose@\endcsname
\expandafter\newif\csname if@pstool@write@aux\endcsname

\newwrite\pstool@out
\newread\pstool@mainfile@ior
\newread\pstool@auxfile@ior
```

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

```
\let\pstool@tex\@empty
```

7.2 Package options

```
\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\pstool@process@choice]{all,none,auto}{}
\ExecuteOptionsX{process=auto}

\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}

\DeclareOptionX{cleanup}{%
    \edef\pstool@rm@files{\zap@space #1 \@empty}%
    \@for\@ii:=\pstool@rm@files\do{%
        \edef\@tempa{\@ii}%
        \def\@tempb{.aux}%
        \ifx\@tempa\@tempb
            \PackageWarning{pstool}{^^J\space\space%
              You have requested that ".aux" files be deleted.^^J\space\space
              Cross-referencing within pstool graphics therefore disabled.^^J%
            This warning occurred}
        \fi
    }
}
\ExecuteOptionsX{cleanup={.tex,.dvi,.ps,.pdf,.log}}

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

There is an implicit \space after the bitmap options.

```
\define@choicekey*{pstool.sty}{bitmap}
    [\@tempa\@tempb]{auto,lossless,lossy}{%
    \ifcase\@tempb
        \let\pstool@bitmap@opts\@empty
      \or
        \def\pstool@bitmap@opts{%
            -dAutoFilterColorImages=false
            -dAutoFilterGrayImages=false
            -dColorImageFilter=/FlateEncode
            -dGrayImageFilter=/FlateEncode % space
        }
      \or
        \def\pstool@bitmap@opts{%
            -dAutoFilterColorImages=false
            -dAutoFilterGrayImages=false
            -dColorImageFilter=/DCTEncode
            -dGrayImageFilter=/DCTEncode % space
```

```
        }
    \fi
}
\ExecuteOptionsX{bitmap=lossless}

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

\ExecuteOptionsX{
    latex-options={},
    dvips-options={},
    ps2pdf-options={-dPDFSETTINGS=/prepress},
    pdfcrop-options={}
}

\DeclareOptionX{macro-file}{%
    \IfFileExists{#1}
      {\def\pstool@macrofile{#1}}
      {%
          \let\pstool@macrofile\@empty
          \PackageError{pstool}{^^J\space\space%
            No file `#1' found for "macro-file" package option.^^J
          This warning occurred}
      }
}
```

Default:

```
\IfFileExists{\jobname-pstool.tex}
  {\edef\pstool@macrofile{\jobname-pstool.tex}}
  {\let\pstool@macrofile\@empty}


\ifpdf
    \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
```

A command to set pstool options after the package is loaded.

```
\newcommand\pstoolsetup{%
    \setkeys{pstool.sty}%
}
```

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).

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

\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}
\edef\pstool@cp@cmd{\ifwindows copy \else cp - \fi}
```

Delete a file if it exists:
#1: path
#2: filename

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

```
}
```

Copy a file if it exists:
#1: path
#2: filename
#3: new filename

```
\newcommand\pstool@cp[3]{%
    \IfFileExists{#1#2}{%
        \immediate\write18{%
            cd "#1"\pstool@cmdsep\pstool@cp@cmd "#2" "#3"
        }%
    }{}%
}
```

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

```
\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.

```
\def\pstool@error{%
    \fbox{%
        \parbox{0.8\linewidth}{%
            \color{red}\centering\ttfamily\scshape\small
            An error occured processing graphic:\\
```

```
            \upshape`%
              \detokenize\expandafter{\pstool@path}%
              \detokenize\expandafter{\pstool@filestub}.eps%
            '\\\bigskip
            \tiny
            Check the log file for compilation errors:\\
            `%
              \detokenize\expandafter{\pstool@path}%
              \detokenize\expandafter{\pstool@filestub}-pstool.log%
            '\\
        }%
    }%
}

\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.)

```
\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
}
\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.

```
\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... }%
        \def\pstool@status{0}%
    \fi
}
\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.

```
\def\pstool@getpaths#1{%
  \filename@parse{#1}%
  \ifx\filename@area\@empty
    \def\pstool@path{./}%
  \else
    \let\pstool@path\filename@area
  \fi
  \let\pstool@filestub\filename@base
}
```

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

```
\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.

```
\newcommand\pstool@alwaysprocess[3][]{%
    \pstool@getpaths{#2}%
    \pstool@process{#1}{#3}%
}
```

```
\ifpdf
    \newcommand\pstool@neverprocess[3][]{%
        \pstool@includegraphics{#2}%
    }
  \else
    \newcommand\pstool@neverprocess[3][]{%
        \begingroup
          \setkeys*{pstool.sty}{#1}%
          #3%
          \expandafter\pstool@includegraphics\expandafter[\XKV@rm]{#2}%
        \endgroup
    }
\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.

```
\ExplSyntaxOn
\newcommand\pstool@maybeprocess[3][]
 {
    \pstool_if_should_process:nTF {#2}
     { \pstool@process{#1}{#3} }
     { \pstool@includegraphics{#2} }
 }

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

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

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

  \exp_args:Nx \clist_map_inline:nn { \pstool@macrofile , \pstool@tex
}
  % empty entries are ignored in clist mappings, so no need to filter here
   {
```

```
    \filemodCmp {##1} {\pstool@path\pstool@filestub.pdf}
     {
        \clist_map_break:n { \use_i_delimit_by_q_stop:nw \prg_return_true: }
     }
     {}
   }

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

  \use_i_delimit_by_q_stop:nw \prg_return_false:
  \q_stop
 }
\ExplSyntaxOff
```

10 User commands

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

```
\ifpdf
    \newcommand\pstool{%
        \ifcase\pstool@process@choice\relax
            \expandafter \pstool@alwaysprocess \or
            \expandafter \pstool@neverprocess  \or
            \expandafter \pstool@maybeprocess
        \fi
    }
    \WithSuffix\def\pstool!{%
        \ifcase\pstool@process@choice\relax
            \expandafter \pstool@alwaysprocess \or
            \expandafter \pstool@neverprocess  \or
            \expandafter \pstool@neverprocess
        \fi
    }
    \WithSuffix\def\pstool*{%
        \ifcase\pstool@process@choice\relax
            \expandafter \pstool@alwaysprocess \or
            \expandafter \pstool@neverprocess  \or
            \expandafter \pstool@alwaysprocess
```

```
        \fi
    }
  \else
    \let\pstool\pstool@neverprocess
    \WithSuffix\def\pstool!{\pstool@neverprocess}
    \WithSuffix\def\pstool*{\pstool@neverprocess}
\fi
```

11 The figure processing

And this is the main macro.

```
\newcommand\pstool@process[2]{%
    \begingroup
    \setkeys*{pstool.sty}{#1}%
    \pstool@echo@verbose{%
        ^^J^^J=== pstool: begin processing ===}%
    \pstool@write@processfile{#1}
        {\pstool@path\pstool@filestub}{#2}%
    \pstool@exe{auxiliary process: \pstool@filestub\space}
      {./}{latex
          -shell-escape
          -output-format=dvi
          -output-directory="\pstool@path"
          -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}{\pstool@path}{%
        dvips \if@pstool@verbose@\else -q \fi -Ppdf
          \pstool@dvips@opts\space "\pstool@jobname.dvi"}%
```

Pre-process `ps2pdf` options for Windows (sigh):

```
    \pstool@pspdf@opts@preprocess \pstool@bitmap@opts
    \pstool@pspdf@opts@preprocess \pstool@pspdf@opts
```

Generate the PDF:

```
    \if@pstool@pdfcrop@
        \pstool@exe{ps2pdf}{\pstool@path}{%
            ps2pdf \pstool@bitmap@opts \pstool@pspdf@opts \space
              "\pstool@jobname.ps" "\pstool@jobname.pdf"}%
        \pstool@exe{pdfcrop}{\pstool@path}{%
            pdfcrop \pstool@pdfcrop@opts\space
```

```
                "\pstool@jobname.pdf" "\pstool@filestub.pdf"}%
      \else
        \pstool@exe{ps2pdf}{\pstool@path}{%
            ps2pdf \pstool@bitmap@opts \pstool@pspdf@opts \space
              "\pstool@jobname.ps" "\pstool@filestub.pdf"}%
    \fi
```
Finish up: (implies success!)
```
    \pstool@endprocess{%
        \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:
```
        \pstool@write@aux
        \pstool@cleanup
    }%
    \pstool@echo@verbose{^^J=== pstool: end processing ===^^J}%
    \endgroup
}

\newcommand\pstool@write@aux{%
    \endlinechar=-1
    \@tempswatrue
    \@pstool@write@auxfalse
    \in@false
    \openin \pstool@auxfile@ior "\pstool@path\pstool@jobname.aux"\relax
    \@whilesw \if@tempswa \fi {%
        \readline \pstool@auxfile@ior to \@tempa
        \ifeof \pstool@auxfile@ior
            \@tempswafalse
          \else
            \edef\@tempb{\detokenize\expandafter{\pstool@auxmarker@text/}}%
            \ifx\@tempa\@tempb
                \@tempswafalse
              \else
                \if@pstool@write@aux
                    \immediate\write\@mainaux{\unexpanded\expandafter{\@tempa}}%
                \fi
                \edef\@tempb{\detokenize\expandafter{\pstool@auxmarker@text*}}%
                \ifx\@tempa\@tempb
                    \@pstool@write@auxtrue
                \fi
            \fi
```

```
        \fi
    }
    \closein \pstool@auxfile@ior
}

\ExplSyntaxOn
\cs_new:Npn \pstool@pspdf@opts@preprocess #1
 {
    \ifwindows
        \exp_args:NNnx \tl_replace_all:Nnn #1 {=} { \cs_to_str:N \# }
    \fi
 }
\ExplSyntaxOff
```

For what's coming next.

```
\edef\@begindocument@str{\detokenize\expandafter{\string\begin{document}}}
\edef\@endpreamble@str{\string\EndPreamble}
\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`.

```
\ifpdf
    \pstool@rm{}{\jobname.oldaux}
    \pstool@cp{}{\jobname.aux}{\jobname.oldaux}
    \AtEndDocument{\pstool@rm{}{\jobname.oldaux}}
\fi

\edef\pstool@auxmarker#1{\string\@percentchar\space <#1PSTOOLLABELS>}
\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).

```
\def\pstool@write@processfile#1#2#3{%
    \immediate\openout\pstool@out #2\pstool@suffix.tex\relax
```

Put down a label so we can pass through the current page number:

```
    \edef\pstool@label{pstool-\pstool@path\pstool@filestub}%
    \protected@write\@auxout{}%
      {\string\newlabel{\pstool@label}{{\@currentlabel}{\the\c@page}}}%
```

And copy the main file's bbl file too: (necessary only for biblatex but do it always)

```
    \pstool@rm{\pstool@path}{\pstool@jobname.bbl}%
```

```
    \pstool@cp{}{\jobname.bbl}{\pstool@path\pstool@jobname.bbl}%
```
Scan the main document line by line; print preamble into auxiliary file until the document begins or `\EndPreamble` is found:
```
    \endlinechar=-1
    \def\@tempa{\pdfoutput=0\relax}%
    \in@false
    \openin\pstool@mainfile@ior "\jobname"\relax
    \@whilesw \unless\ifin@ \fi {%
        \immediate\write\pstool@out{\unexpanded\expandafter{\@tempa}}%
        \readline\pstool@mainfile@ior to\@tempa
        \let\@tempc\@tempa
        \trim@multiple@spaces@in\@tempa
        \expandafter\expandafter\expandafter\in@first
        \expandafter\expandafter\expandafter{%
          \expandafter\@begindocument@str
        \expandafter}%
        \expandafter{\@tempa}%
        \unless\ifin@
            \expandafter\expandafter\expandafter\in@first
            \expandafter\expandafter\expandafter{%
              \expandafter\@endpreamble@str
            \expandafter}%
            \expandafter{\@tempa}%
        \fi
    }
    \closein\pstool@mainfile@ior
```
Now the preamble of the process file:
```
    \immediate\write\pstool@out{%
        \if@pstool@pdfcrop@\else
            \noexpand\usepackage[active,tightpage]{preview}^^J%
        \fi
        \unexpanded{%
            \pagestyle{empty}^^J^^J% remove the page number
        }%
        \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.
```
        \expandafter\ifx\csname r@\pstool@label\endcsname\relax\else
            \def\noexpand\thepage{\unexpanded\expandafter{\thepage}}^^J%
            \noexpand\setcounter{page}{%
```

```
                \expandafter\expandafter\expandafter
                  \@secondoftwo\csname r@\pstool@label\endcsname
            }^^J%
        \fi
```

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

```
        \noexpand\@input{\jobname.oldaux}^^J%
        \noexpand\makeatother^^J^^J%
        \noexpand\begin{document}^^J%
        \noexpand\makeatletter^^J%
        \unexpanded{\immediate\write\@mainaux}{\pstool@auxmarker*}^^J%
        \noexpand\makeatother^^J^^J%
        \unexpanded{%
            \centering\null\vfill^^J%
        }%
        ^^J%
        \if@pstool@pdfcrop@\else
            \noexpand\begin{preview}^^J%
        \fi
        \unexpanded{#3^^J}
        \noexpand\includegraphics
            [\unexpanded\expandafter{\XKV@rm}]
            {\pstool@filestub}^^J%
        \if@pstool@pdfcrop@\else
            \noexpand\end{preview}^^J%
        \fi
        ^^J%
        \unexpanded{\vfill^^J^^J\makeatletter^^J\immediate\write\@mainaux}{\pstool@auxma
        \unexpanded{\makeatother^^J\end{document}}^^J%
    }%
    \immediate\closeout\pstool@out
}

\def\pstool@cleanup{%
    \@for\@ii:=\pstool@rm@files\do{%
        \pstool@rm{\pstool@path}{\pstool@jobname\@ii}%
    }%
}

\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:

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

Parse optional input definitions:

```
\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 (or other) macros

```
\newcommand\pstool@@psfragfig[4]{%
    % Find the .eps file to use.
    \IfFileExists{#2-psfrag.eps}{%
        \edef\pstool@eps{#2-psfrag}%
        \IfFileExists{#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}%
        \IfFileExists{#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}%
          }{%
            \PackageWarning{pstool}{%
                No file "#2.tex" or "#2-psfrag.tex" can be found
                that may contain macros for "\pstool@eps.eps"%
            }%
        }%
    }%
    % 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.)
    \ifx\pstool@eps\@undefined\else
        \edef\@tempa{%
            \unexpanded{\pstool#3[#1]}{\pstool@eps}{%
                \ifx\pstool@macrofile\@empty\else
                    \unexpanded{\csname @input\endcsname}{\pstool@macrofile}%
                \fi
                \ifx\pstool@tex\@empty\else
                    \unexpanded{\csname @input\endcsname}{\pstool@tex}%
                \fi
                \unexpanded{#4}%
            }%
        }\@tempa
```

```
    \fi
}
```

—The End—