The `preview` Package for LaTeX
Version 0.7.8

David Kastrup*

2003/01/20

1 Introduction

The main purpose of this package is the extraction of certain environments (most notably displayed formulas) for use in different contexts. While the erstwhile application has been the embedding of those preview fragments into Emacs source buffers under the AUC TeX editing environment, other applications are easily imaginable.

In particular it should be noted that producing EPS files with Dvips and its derivatives using the `-E` option is not currently well-supported by LaTeX. People make do by fiddling around with `\thispagestyle{empty}` and hoping for the best (namely, that the specified contents will indeed fit on single pages), and then trying to guess the baseline of the resulting code and stuff, but this is at best dissatisfactory. The preview package provides an easy way to ensure that exactly one page per request gets shipped, with a well-defined baseline and no page decorations. Thus you can safely use

```
dvips -E -i
```

and get a single EPS file with shrink-wrapped bounding box for every generated image of a LaTeX run.

If your ultimate goal is to produce a set of files in a different format that can be produced by GhostScript, take a look at the `tightpage` option of the preview package. This will embed the page dimensions into the PostScript code, obliterating the need to use the `-E -i` options to Dvips. You can then produce all image files with a single run of GhostScript from a single PostScript file for all images at once. The `tightpage` option requires setting the `dvips` option as well.

Various options exist that will pass TeX dimensions and other information about the respective shipped out material (including descender size) into the log file, where external applications might make use of it.

The possibility for generating a whole set of graphics with a single run of LaTeX, Dvips, and GhostScript increases both speed and robustness of applications. It is

---

[*] `dakas@users.sourceforge.net`

to be hoped that applications like LaTeX2HTML will be able to make use of this package in future.

2 Package options

The package is included with the customary

```
\usepackage[⟨options⟩]{preview}
```

You should usually load this package as the last one, since it redefines several things that other packages may also provide.

The following options are available:

`active` is the most essential option. If this option is not specified, the `preview` package will be inactive and the document will be typeset as if the `preview` package were not loaded, except that all declarations and environments defined by the package are still legal but have no effect. This allows defining previewing characteristics in your document, and only activating them by calling LaTeX as

```
latex '\PassOptionsToPackage{active}{preview}
\input{⟨filename⟩}'
```

`noconfig` Usually the file `prdefault.cfg` gets loaded whenever the `preview` package gets activated. `prdefault.cfg` is supposed to contain definitions that can cater for otherwise bad results, for example, if a certain document class would otherwise lead to trouble. It also can be used to override any settings made in this package, since it is loaded at the very end of it. In addition, there may be configuration files specific for certain `preview` options like `auctex` which have more immediate needs. The `noconfig` option suppresses loading of those option files, too.

`psfixbb` Dvips determines the bounding boxes from the material in the DVI file it understands. Lots of PostScript specials are not part of that. Since the TeX boxes do not make it into the DVI file, but merely characters, rules and specials do, Dvips might include far too small areas. The option `psfixbb` will include `/dev/null` as a graphic file in the ultimate upper left and lower right corner of the previewed box. This will make Dvips generate an appropriate bounding box.

`dvips` If this option is specified as a class option or to other packages, several packages pass things like page size information to Dvips, or cause crop marks or draft messages written on pages. This seriously hampers the usability of previews. If this option is specified, the changes will be undone if possible.

`displaymath` will make all displayed math environments subject to preview processing. This will typically be the most desired option.

`floats` will make all float objects subject to preview processing. If you want to be more selective about what floats to pass through to a preview, you should instead use the `\PreviewSnarfEnvironment` command on the floats you want to have previewed.

`textmath` will make all text math subject to previews. Since math mode is used throughly inside of LaTeX even for other purposes, this works by redefining `\(`, `\)` and `$`. Only occurences of these text math delimiters in later loaded packages and in the main document will thus be affected.

`graphics` will subject all `\includegraphics` commands to a preview.

`sections` will subject all section headers to a preview.

`delayed` will delay all activations and redefinitions the `preview` package makes until `\begin{document}`. The purpose of this is to cater for documents which should be subjected to the `preview` package without having been prepared for it. You can process such documents with

```
latex '\RequirePackage[active,delayed,⟨options⟩]{preview}
\input{⟨filename⟩}'
```

This relaxes the requirement to be loading the `preview` package as last package.

⟨*driver*⟩ loads a special driver file `pr`⟨*driver*⟩`.def`. The remaining options are implemented through the use of driver files.

`auctex` This driver will produce fake error messages at the start and end of every preview environment that enable the Emacs package Preview-LaTeX in connection with AUC TeX to pinpoint the exact source location where the previews have originated. Unfortunately, there is no other reliable means of passing the current TeX input position *in* a line to external programs. In order to make the parsing more robust, this option also switches off quite a few diagnostics that could be misinterpreted.

You should not specify this option manually, since it will only be needed by automated runs that want to parse the pseudo error messages. Those runs will then use `\PassOptionsToPackage` in order to effect the desired behaviour. In addition, `prauctex.cfg` will get loaded unless inhibited by the `noconfig` option. This caters for the most frequently encountered problematic commands.

`showlabels` During the editing process, some people like to see the label names in their equations, figures and the like. Now if you are using Emacs for editing, and in particular Preview-LaTeX, I'd strongly recommend that you check out the RefTeX package which pretty much obliterates the need for this kind of functionality. If you still want it, standard LaTeX provides it with the `showkeys` package, and there is also the less encompassing `showlabels` package. Unfortunately, since those go to some pain not to change the page layout and spacing, they also don't change `preview`'s idea of the TeX dimensions of the involved boxes. So if you are using `preview` for determing bounding boxes, those packages are mostly useless. The option `showlabels` offers a substitute for them.

`tightpage` It is not uncommon to want to use the results of `preview` as graphic images for some other application. One possibility is to generate a flurry of EPS files with

```
dvips -E -i -Pwww -o ⟨outputfile⟩.000 ⟨inputfile⟩
```

However, in case those are to be processed further into graphic image files by GhostScript, this process is inefficient since all of those files need to be processed one by one. In addition, it is necessary to extract the bounding box comments from the EPS files and convert them into page dimension parameters for GhostScript in order to avoid full-page graphics. This is not even possible if you wanted to use GhostScript in a *single* run for generating the files from a single PostScript file, since Dvips will in that case leave no bounding box information anywhere.

The solution is to use the `tightpage` option together with the `dvips` option so that additional PostScript code gets written into the produced file that will set the device dimensions at the start of each output page. That way a single command line like

```
gs -sDEVICE=png16m -dTextAlphaBits=4 -r300
-dGraphicsAlphaBits=4 -dSAFER -q -dNOPAUSE
-sOutputFile=⟨outputfile⟩%d.png ⟨inputfile⟩.ps
```

will be able to produce tight graphics from a single PostScript file generated with Dvips *without* use of the options `-E -i`, in a single run. If you need this in a batch environment where you don't want to use `preview`'s automatic extraction facilities, no problem: just don't use any of the special options, and wrap everything to be previewed into `preview` environments.

If the pages under the `tightpage` option are just too tight, you can adjust by setting the length `\PreviewBorder` to a different value by using `\setlength`. The default value is `0.50001bp`, which is half of a usual PostScript point, rounded up. If you go below this value, the resulting page size may drop below `1bp`, and GhostScript does not seem to like that. If you need finer control, you can adjust the bounding box dimensions individually by changing the macro `\PreviewBbAdjust` with the help of `\renewcommand`. Its default value is

```
\newcommand \PreviewBbAdjust {-\PreviewBbAdjust
-\PreviewBbAdjust \PreviewBbAdjust \PreviewBbAdjust}
```

This adjusts the left, lower, right and upper borders by the given amount. The macro must contain 4 TeX dimensions after another, and you may not omit the units if you specify them explicitly instead of by register. PostScript points have the unit `bp`.

`lyx` This option is for the sake of LyX developers. It will output a few diagnostics relevant for the sake of LyX' preview functionality (at the time of writing, just implemented for math insets, in the CVS version of LyX that will eventually be released as 1.3.0).

`counters` This writes out diagnostics at the start and the end of previews. Only the counters changed since the last output get written, and if no counters changed, nothing get written at all. The list consists of counter name and value, both enclosed in `{}` braces, followed by a space. The last such pair is followed by a colon (`:`) if it is at the start of the preview snippet, and by a period (`.`) if it is at the end. The order of different diagnostics like this being issued depends on the order of the specification of the options when calling the package.

Systems like Preview-LaTeX use this for keeping counters accurate when single previews are regenerated.

`footnotes` This makes footnotes render as previews, and only as their footnote symbol. A convenient editing feature inside of Emacs.

The following options are just for debugging purposes of the package and similar to the corresponding TeX commands they allude to:

`tracingall` causes lots of diagnostic output to appear in the log file during the preview collecting phases of TeX's operation. In contrast to the similarly named TeX command, it will not switch to `\errorstopmode`, nor will it change the setting of `\tracingonline`.

`showbox` This option will show the contents of the boxes shipped out to the DVI files. It also sets `\showboxbreadth` and `\showboxdepth` to their maximum values at the end of loading this package, but you may reset them if you don't like that.

3 Provided Commands

`preview` The `preview` environment causes its contents to be set as a single preview image. Insertions like figures and footnotes (except those included in minipages) will typically lead to error messages or be lost. In case the `preview` package has not been activated, the contents of this environment will be typeset normally.

`nopreview` The `nopreview` environment will cause its contents not to undergo any special treatment by the `preview` package. When `preview` is active, the contents will be discarded like all main text that does not trigger the `preview` hooks. When `preview` is not active, the contents will be typeset just like the main text.

Note that both of these environments typeset things as usual when preview is not active. If you need something typeset conditionally, use the `\ifPreview` conditional for it.

`\PreviewMacro` If you want to make a macro like `\includegraphics` (actually, this is what is done by the `graphics` option to `preview`) produce a preview image, you put a declaration like

```
\PreviewMacro[*[[!]{\includegraphics}
```

or, more readable,

```
\PreviewMacro[{*[][]{}}]{\includegraphics}
```

into your preamble. The optional argument to `\PreviewMacro` specifies the arguments `\includegraphics` accepts, since this is necessary information for properly ending the preview box. Note that if you are using the more readable form, you have to enclose the argument in a `[{` and `}]` pair. The inner braces are necessary to stop any included `[]` pairs from prematurely ending the optional argument, and to make a single `{}` denoting an optional argument not get stripped away by TEX's argument parsing.

The letters simply mean

`*` indicates an optional `*` modifier, as in `\includegraphics*`.

`[` indicates an optional argument in brackets. This syntax is somewhat baroque, but brief.

`[]` also indicates an optional argument in brackets. Be sure to have encluded the entire optional argument specification in an additional pair of braces as described above.

`!` indicates a mandatory argument.

`{}` indicates the same. Again, be sure to have that additional level of braces around the whole argument specification.

`?`***⟨delimiter⟩***`{`***⟨true case⟩***`}{`***⟨false case⟩***`}` is a conditional. The next character is checked against being equal to ⟨*delimiter*⟩. If it is, the specification ⟨*true case*⟩ is used for the further parsing, otherwise ⟨*false case*⟩ will be employed. In neither case is something consumed from the input, so `{`⟨*true case*⟩`}` will still have to deal with the upcoming delimiter.

`@{`***⟨literal sequence⟩***`}` will insert the given sequence literally into the executed call of the command.

There is a second optional argument in brackets that can be used to declare any default action to be taken instead. This is mostly for the sake of macros that influence numbering: you would want to keep their effects in that respect. The default action should use `#1` for referring to the original (not the patched) command with the parsed options appended. Not specifying a second optional argument here is equivalent to specifying `[#1]`.

`\PreviewMacro*` A similar invocation `\PreviewMacro*` simply throws the macro and all of its arguments declared in the manner above away. This is mostly useful for having things like `\footnote` not do their magic on their arguments. More often than not, you don't want to declare any arguments to scan to `\PreviewMacro*` since you would want the remaining arguments to be treated as usual text and typeset in that manner instead of being thrown away. An exception might be, say, sort keys for `\cite`.

A second optional argument in brackets can be used to declare any default action to be taken instead. This is for the sake of macros that influence numbering: you would want to keep their effects in that respect. The default action might use `#1` for referring to the original (not the patched) command with the parsed options appended. Not specifying a second optional argument here is equivalent to specifying `[]` since the command usually gets thrown away.

As an example for using this argument, you might want to specify

```
\PreviewMacro*\footnote[{[]}][#1{}]
```

This will replace a footnote by an empty footnote, but taking any optional parameter into account, since an optional paramter changes the numbering scheme. That way the real argument for the footnote remains for processing by Preview-LaTeX (the actual definition is more complicated in order not to change the numbering in case of optional arguments being present).

`\PreviewEnvironment`
`\PreviewEnvironment*`
The macro `\PreviewEnvironment` works just as `\PreviewMacro` does, only for environments. And the same goes for `\PreviewEnvironment*` as compared to `\PreviewMacro*`.

`\PreviewSnarfEnvironment`
This macro does not typeset the original environment inside of a preview box, but instead typesets just the contents of the original environment inside of the preview box, leaving nothing for the original environment. This has to be used for figures, for example, since they would

1. produce insertion material that cannot be extracted to the preview properly,
2. complain with an error message about not being in outer par mode.

`\PreviewOpen`
`\PreviewClose`
Those Macros form a matched preview pair. This is for macros that behave similar as `\begin` and `\end` of an environment. It is essential for the operation of `\PreviewOpen` that the macro treated with it will open an additional group even when the preview falls inside of another preview or inside of a `nopreview` environment. Similarly, the macro treated with `\reviewClose` will close an environment even when inactive.

`\ifPreview`
In case you need to know whether `preview` is active, you can use the conditional `\ifPreview` together with `\else` and `\fi`.

4 The Implementation

Here we go: the start is somewhat obtuse since we figure out version number and date from RCS strings. This should really be done at docstrip time instead. Takers?

```
⟨*style⟩
```

```
⟨*!active⟩
\NeedsTeXFormat{LaTeX2e} \def\reserved@a #1#2$#3:
#4${\edef#1{\reserved@c #2#4 $}} \def\reserved@c #1 #2${#1}
\reserved@a\reserved@b $Name: rel-0-7-8 $ \ifx\reserved@b\@empty
\reserved@a\reserved@b CVS-$Revision: 1.80 $ \else \begingroup
\lccode`-=`.  \def\next rel-{} \edef\next{\lowercase{\endgroup
    \def\noexpand\reserved@b{\expandafter\next\reserved@b}}} \next \fi
\reserved@a\next $Date: 2003/01/20 00:09:00 $
\edef\next{\noexpand\ProvidesPackage{preview}%
  [\next\space preview-latex \reserved@b]}
\next
```

Since many parts here will not be needed as long as the package is inactive, we will include them enclosed with `<*active>` and `</active>` guards. That way, we can append all of this stuff at a place where it does not get loaded if not necessary.

`\ifPreview` Setting the `\ifPreview` command should not be done by the user, so we don't use `\newif` here. As a consequence, there are no `\Previewtrue` and `\Previewfalse` commands.

```
\let\ifPreview\iffalse
⟨/!active⟩
```

`\ifpr@outer` We don't allow previews inside of previews. The macro `\ifpr@outer` can be used for checking whether we are outside of any preview code.

```
⟨*active⟩
\newif\ifpr@outer
\pr@outertrue
⟨/active⟩
```

`\preview@delay` The usual meaning of `\preview@delay` is to just echo its argument in normal `preview` operation. If `preview` is inactive, it swallows its argument. If the `delayed` option is active, the contents will be passed to the `\AtBeginDocument` hook.

`\pr@advise` The core macro for modifying commands is `\pr@advise`. You pass it the original command name as first argument and what should be executed before the saved original command as second argument.

`\pr@advise@ship` The most often used macro for modifying commands is `\pr@advise@ship`. It receives three arguments. The first is the macro to modify, the second specifies some actions to be done inside of a box to be created before the original macro gets executed, the third one specifies actions after the original macro got executed.

`\pr@loadcfg` The macro `\pr@loadcfg` is used for loading in configuration files, unless disabled by the `noconfig` option.

```
⟨*!active⟩
\let\preview@delay=\@gobble
\let\pr@advise=\@gobbletwo
\def\pr@advise@ship#1#2#3{}
\def\pr@loadcfg#1{\InputIfFileExists{#1.cfg}{}{}}
\DeclareOption{noconfig}{\let\pr@loadcfg=\@gobble}
```

`\pr@addto@front` This adds code globally to the front of a macro.

```
\def\pr@addto@front#1#2{%
  \toks@{#2}\toks@\expandafter{\the\expandafter\toks@#1}%
  \xdef#1{\the\toks@}}
```

These commands get more interesting when `preview` is active:

```
\DeclareOption{active}{%
  \let\ifPreview\iftrue
  \def\pr@advise#1{%
    \expandafter\pr@adviseii\csname pr@\string#1\endcsname#1}%
  \def\pr@advise@ship#1#2#3{\pr@advise#1{\pr@protect@ship{#2}{#3}}}%
  \let\preview@delay\@firstofone}
```

`\pr@adviseii` Now `\pr@advise` needs its helper macro. In order to avoid recursive definitions, we advise only macros that are not yet advised. Or, more exactly, we throw away the old advice and only take the new one.

```
\def\pr@adviseii#1#2#3{\preview@delay{%
  \ifx#1\relax \let#1#2\fi
  \toks@{#3#1}\edef#2{\the\toks@}}}
```

The `delayed` option is easy to implement: this is *not* done with `\let` since at the course of document processing, LaTeX redefines `\AtBeginDocument` and we want to follow that redefinition.

```
\DeclareOption{delayed}{%
  \ifPreview \def\preview@delay{\AtBeginDocument}\fi
}
```

`\ifpr@fixbb` Another conditional. `\ifpr@fixbb` tells us whether we want to surround the typeset materials with invisible rules so that Dvips gets the bounding boxes right for, say, pure PostScript inclusions.

If you are installing this on an operating system different from the one `preview` has been developed on, you might want to redefine `\pr@markerbox` in your `prdefault.cfg` file to use a file known to be empty, like `/dev/null` is under Unix. Make this redefinition depend on `\ifpr@fixbb` since only then `\pr@markerbox` will be defined.

```
\newif\ifpr@fixbb
\pr@fixbbfalse
\DeclareOption{psfixbb}{\ifPreview%
  \pr@fixbbtrue
  \newbox\pr@markerbox
  \setbox\pr@markerbox\hbox{\special{psfile=/dev/null}\fi}%
}
```

The `dvips` option redefines the `bop-hook` to reset the page size.

```
\DeclareOption{dvips}{%
  \preview@delay{\AtBeginDvi{%
    \special{!userdict begin/bop-hook{/isls false def%
    /vsize 792 def/hsize 612 def}def end}}}}
⟨/!active⟩
```

4.1 The internals

Those are only needed if `preview` is active.

```
⟨*active⟩
```

`\pr@snippet`

`\pr@snippet` is the current snippet number. We need a separate counter to `\c@page` since several other commands might fiddle with the page number.

```
\newcount\pr@snippet
\global\pr@snippet=1
```

`\pr@protect`

This macro gets one argument which is unpacked and executed in typesetting situations where we are not yet inside of a preview.

```
\def\pr@protect{\ifx\protect\@typeset@protect
  \ifpr@outer \expandafter\expandafter\expandafter
     \@secondoftwo\fi\fi\@gobble}
```

`\pr@protect@ship`

Now for the above mentioned `\pr@protect@ship`. This gets three arguments. The first is what to do at the beginning of the preview, the second what to do at the end, the third is the macro where we stored the original definition.

In case we are not in a typesetting situation, `\pr@protect@ship` leaves the stored macro to fend for its own. No better or worse protection than the original. And we only do anything different when `\ifpr@outer` turns out to be true.

```
\def\pr@protect@ship{\pr@protect{\@firstoftwo\pr@startbox}%
   \@gobbletwo}
```

`\pr@insert`

We don't want insertions to end up on our lists. So we disable them right now by replacing them with the following:

```
\def\pr@insert{\afterassignment\pr@insertii\count@}
\def\pr@insertii{\setbox\pr@box\vbox}
```

`\pr@box`
`\pr@startbox`

Previews will be stored in `\box\pr@box`. `\pr@startbox` gets two arguments: code to execute immediately before the following stuff, code to execute afterwards. You have to cater for `\pr@endbox` being called at the right time yourself. We will use a `\vsplit` on the box later in order to remove any leading glues, penalties and similar stuff. For this reason we start off the box with an optimal break point.

```
\newbox\pr@box
\def\pr@startbox#1#2{%
  \ifpr@outer
    \toks@{#2}%
    \edef\pr@cleanup{\the\toks@}%
    \setbox\pr@box\vbox\bgroup
    \break
    \pr@outerfalse\@arrayparboxrestore
    \let\insert\pr@insert
    \expandafter\expandafter\expandafter
    \pr@ship@start
    \expandafter\@firstofone
  \else
```

```
     \expandafter \@gobble
  \fi{#1}}
```

`\pr@endbox` Cleaning up also is straightforward. If we have to watch the bounding TeX box, we want to remove spurious skips. We also want to unwrap a possible single line paragraph, so that the box is not full line length. We use `\vsplit` to clean up leading glue and stuff, and we make some attempt of removing trailing ones. After that, we wrap up the box including possible material from `\AtBeginDvi`. If the `psfixbb` option is active, we adorn the upper left and lower right corners with copies of `\pr@markerbox`. The first few lines cater for LaTeX hiding things like like the code for `\paragraph` in `\everypar`.

```
\def\pr@endbox{%
   \let\reserved@a\relax
   \ifvmode \edef\reserved@a{\the\everypar}%
      \ifx\reserved@a\@empty\else
            \dimen@\prevdepth
            \noindent\par
            \setbox\z@\lastbox\unskip\unpenalty
            \prevdepth\dimen@
            \setbox\z@\hbox\bgroup\penalty-\maxdimen\unhbox\z@
              \ifnum\lastpenalty=-\maxdimen\egroup
              \else\egroup\box\z@ \fi\fi\fi
   \ifhmode \par\unskip\setbox\z@\lastbox
     \nointerlineskip\hbox{\unhbox\z@\/}%
   \else \unskip\unpenalty\unskip \fi
   \egroup
   \setbox\pr@box\vbox{%
       \baselineskip\z@skip \lineskip\z@skip \lineskiplimit\z@
       \@begindvi
       \nointerlineskip
       \splittopskip\z@skip\setbox\z@\vsplit\pr@box to\z@
       \unvbox\z@
       \nointerlineskip
       %\color@setgroup
       \box\pr@box
       %\color@endgroup
     }%
```

`\pr@ship@end` At this point, `\pr@ship@end` gets called. You must not under any circumstances change `\box\pr@box` in any way that would add typeset material at the front of it, except for PostScript header specials, since the front of `\box\pr@box` may contains stuff from `\AtBeginDvi`. `\pr@ship@end` contains two types of code additions: stuff that adds to `\box\pr@box`, like the `labels` option does, and stuff that measures out things or otherwise takes a look at the finished `\box\pr@box`, like the `auctex` or `showbox` option do. The former should use `\r@addto@front` for adding to this hook, the latter use `\@addto@macro` for adding at the end of this hook.

Note that we shift the output box up by its height via `\voffset`. This has three reasons: first we make sure that no package-inflicted non-zero value of `\voffset` or `\hoffset` will have any influence on the positioning of our box. Second we shift the box such that its basepoint will exactly be at the (1in,1in) mark defined by TeX. That way we can properly take ascenders into account. And the third reason is that TeX treats a `\hbox` and a `\vbox` differently with regard to the treating of its depth.

```
  \pr@ship@end
  {\let\protect\noexpand
  \voffset=-\ht\pr@box
  \hoffset=\z@
  \c@page=\pr@snippet
  \pr@shipout
  \ifpr@fixbb\hbox{%
    \dimen@\wd\pr@box
    \@tempdima\ht\pr@box
    \@tempdimb\dp\pr@box
    \box\pr@box
    \llap{\raise\@tempdima\copy\pr@markerbox\kern\dimen@}%
    \lower\@tempdimb\copy\pr@markerbox}%
  \else \box\pr@box \fi}%
  \global\advance\pr@snippet\@ne
  \pr@cleanup
}
```

Oh, and we kill off the usual meaning of `\shipout` in case somebody makes a special output routine. The following is rather ugly, but should do the trick most of the time since `\shipout` is most often called in a local group by `\output`.

`\shipout`

```
\let\pr@shipout=\shipout
\def\shipout{\deadcycles\z@\setbox\z@\box\voidb@x\setbox\z@}
```

4.2 Parsing commands

`\pr@parseit`
`\pr@endparse`
`\pr@callafter`

The following stuff is for parsing the arguments of commands we want to somehow surround with stuff. Usage is

> `\pr@callafter`⟨*aftertoken*⟩⟨*parsestring*⟩`\pr@endparse`
> ⟨*macro*⟩⟨*parameters*⟩

⟨*aftertoken*⟩ is stored away and gets executed once parsing completes, with its first argument being the parsed material. ⟨*parsestring*⟩ would be, for example for the `\includegraphics` macro, `*[[!`, an optional `*` argument followed by two optional arguments enclosed in `[]`, followed by one mandatory argument.

For the sake of a somewhat more intuitive syntax, we now support also the syntax `{*[]{}}` in the optional argument. Since TeX strips redundant braces, we have to write `[{{}}]` in this syntax for a single mandatory argument. Hard to avoid. We use an unusual character for ending the parsing. The implementation is rather trivial.

```
\def\pr@parseit#1{\csname pr@parse#1\endcsname}
\let\pr@endparse=\@percentchar
\def\next#1{%
\def\pr@callafter{%
  \afterassignment\pr@parseit
  \let#1= }}
\expandafter\next\csname pr@parse\pr@endparse\endcsname
```

`\pr@parse*`

Straightforward, same mechanism LaTeX itself employs.

```
\expandafter\def\csname pr@parse*\endcsname#1\pr@endparse#2{%
  \@ifstar{\pr@parseit#1\pr@endparse{#2*}}%
          {\pr@parseit#1\pr@endparse{#2}}}
```

`\pr@parse[`
`\pr@brace`

Copies optional parameters in brackets if present. The additional level of braces is necessary to ensure that braces the user might have put to hide a `]` bracket in an optional argument don't get lost. There will be no harm if such braces were not there at the start.

```
\expandafter\def\csname pr@parse[\endcsname#1\pr@endparse#2{%
  \@ifnextchar[{\pr@bracket#1\pr@endparse{#2}}%
               {\pr@parseit#1\pr@endparse{#2}}}
\def\pr@bracket#1\pr@endparse#2[#3]{\pr@parseit#1\pr@endparse{#2[{#3}]}}
```

`\pr@parse]`

This is basically a do-nothing, so that we may use the syntax `{*[][]!}` in the optional argument instead of the more concise but ugly `*[[!` which confuses the brace matchers of editors.

```
\expandafter\let\csname pr@parse]\endcsname=\pr@parseit
```

`\pr@parse`
`\pr@parse!`

Mandatory arguments are perhaps easiest to parse.

```
\def\pr@parse#1\pr@endparse#2#3{%
  \pr@parseit#1\pr@endparse{#2{#3}}}
\expandafter\let\csname pr@parse!\endcsname=\pr@parse
```

`\pr@parse?`
`\pr@parsecond`

This does an explicit call of `\@ifnextchar` and forks into the given two alternatives as a result.

```
\expandafter\def\csname pr@parse?\endcsname#1#2\pr@endparse#3{%
  \begingroup\toks@{#2\pr@endparse{#3}}
  \@ifnextchar#1{\pr@parsecond\@firstoftwo}%
               {\pr@parsecond\@secondoftwo}}
\def\pr@parsecond#1{\edef\next{%
  \endgroup\noexpand\expandafter
  \noexpand\pr@parseit\noexpand#1\the\toks@}\next}
```

`\pr@parse@`

This makes it possible to insert literal material into the argument list.

```
\def\pr@parse@#1#2\pr@endparse#3{%
  \pr@parseit #2\pr@endparse{#3#1}}
⟨/active⟩
```

4.3 Selection options

The `displaymath` option. The `equation` environments in AMSLaTeX already do too much before our hook gets to interfere, so we hook earlier. Some juggling is involved to ensure we get the original `\everydisplay` tokens only once and where appropriate.

The incredible hack with `\dt@ptrue` is necessary for working around bug 'amslatex/3425'.

```
⟨*!active⟩
\begingroup
\catcode`\*=11
\@firstofone{\endgroup
\DeclareOption{displaymath}{%
  \preview@delay{\toks@{%
      \pr@startbox{\noindent$$%
        \aftergroup\pr@endbox\@gobbletwo}{$$}\@firstofone}%
    \everydisplay\expandafter{\the\expandafter\toks@
      \expandafter{\the\everydisplay}}}%
  \pr@advise@ship\equation{\begingroup\aftergroup\pr@endbox
    \def\dt@ptrue{\m@ne=\m@ne}\noindent}
    {\endgroup}%
  \pr@advise@ship\equation*{\begingroup\aftergroup\pr@endbox
    \def\dt@ptrue{\m@ne=\m@ne}\noindent}
    {\endgroup}%
  \PreviewOpen[][\def\dt@ptrue{\m@ne=\m@ne}\noindent#1]\[%
  \PreviewClose\]%
  \PreviewEnvironment[][\noindent#1]{eqnarray}%
  \PreviewEnvironment[][\noindent#1]{eqnarray*}%
  \PreviewEnvironment{displaymath}%
}}
```

The `textmath` option. Some folderol in order to define the active `$` math mode delimiter. \pr@textmathcheck is used for checking whether we have a single `$` or double `$$`. In the latter case, we enter display math (this sort of display math is not allowed inside of LaTeX because of inconsistent spacing, but surprisingly many people use it nevertheless). Strictly speaking, this is incorrect, since not every `$$` actually means display math. For example, `\hbox{$$}` will because of restricted horizontal mode rather yield an empty text math formula. Since our implementation moved the sequence inside of a `\vbox`, the interpretation will change. People should just not enter rubbish like that.

```
\begingroup
\def\next#1#2{%
  \endgroup
  \DeclareOption{textmath}{%
    \preview@delay{\ifx#1\@undefined \let#1=$%$
      \fi\catcode`\$=\active}%
    \pr@advise@ship\(\pr@endaftergroup{}% \)
    \pr@advise@ship#1{\@firstoftwo{\let#1=#2%
        \futurelet\reserved@a\pr@textmathcheck}}{}}%
  \def\pr@textmathcheck{\expandafter\pr@endaftergroup
    \ifx\reserved@a#1{#2#2}\expandafter\@gobbletwo\fi#2}}
\lccode`\~=`\$
\lowercase{\expandafter\next\expandafter~}%
  \csname pr@\string$\endcsname
⟨/!active⟩
```

`\pr@endaftergroup` This justs ends the box after the group opened by `#1` is closed again.

```
⟨*active⟩
\def\pr@endaftergroup#1{#1\aftergroup\pr@endbox}
⟨/active⟩
```

The `graphics` option.

```
⟨*!active⟩
\DeclareOption{graphics}{%
  \PreviewMacro[*[[!]{\includegraphics}%]]
}
```

The `floats` option.

```
\DeclareOption{floats}{%
  \PreviewSnarfEnvironment[![]{@float}%]
  \PreviewSnarfEnvironment[![]{@dblfloat}%]
}
```

The `sections` option.

```
\DeclareOption{sections}{%
  \PreviewMacro[!!!!!!*[!]{\@startsection}%]
}
```

We now interpret any further options as driver files we load. Note that these driver files are loaded even when `preview` is not active. The reason is that they might define commands (like `\PreviewCommand`) that should be available even in case of an inactive package. Large parts of the `preview` package will not have been loaded in this case: you have to cater for that.

```
\DeclareOption*
  {\InputIfFileExists{pr\CurrentOption.def}{}{\OptionNotUsed}}
```

4.4 Preview attaching commands

`\PreviewMacro` As explained above. Detect possible `*` and call appropriate macro.

```
\def\PreviewMacro{\@ifstar\pr@starmacro\pr@macro}
```

The version without `*` is now rather straightforward.

`\pr@macro`
`\pr@domacro`
`\pr@macroii`
`\pr@endmacro`

```
\def\pr@domacro#1#2{%
  \def\next##1{#2}%
  \pr@callafter\next#1\pr@endparse}
\newcommand*\pr@macro[1][]{%
  \def\next[##1]##2{%
```

```
   \pr@advise@ship{##2}{\pr@domacro{#1}{##1\pr@endbox}}{}}%
   \@ifnextchar[\next\pr@macroii}
\def\pr@macroii{\next[##1]}
\def\pr@endmacro#1{#1\pr@endbox}
```

`PreviewMacro*`
`\pr@protect@domacro`
`\pr@starmacro`

The version with * has to parse the arguments, then throw them away. Some internal macros first, then the interface call.

```
\def\pr@protect@domacro#1#2{\pr@protect{%
    \def\next##1{#2}%
    \pr@callafter\next#1\pr@endparse}}
\newcommand*\pr@starmacro[1][]{\def\next[##1]##2{%
    \pr@advise##2{\pr@protect@domacro{#1}{##1}}}%
    \@ifnextchar[\next{\next[]}}
```

`\PreviewOpen`

As explained above. Detect possible * and call appropriate macro.

```
\def\PreviewOpen{\@ifstar\pr@starmacro\pr@open}
```

The version without * is now rather straightforward.

`\pr@open`

```
\newcommand*\pr@open[1][]{%
   \def\next[##1]##2{%
   \pr@advise##2{\begingroup
     \pr@protect@ship{\pr@domacro{#1}{\begingroup\aftergroup\pr@endbox##1}}%
                     {\endgroup}}}%
   \@ifnextchar[\next\pr@macroii}
```

`\PreviewClose`

As explained above. Detect possible * and call appropriate macro.

```
\def\PreviewClose{\@ifstar\pr@starmacro\pr@close}
```

The version without * is now rather straightforward.

`\pr@close`

```
\newcommand*\pr@close[1][]{%
  \def\next[##1]##2{%
   \pr@advise{##2}{\pr@domacro{#1}{##1\endgroup}}}%
   \@ifnextchar[\next\pr@macroii}
```

`\PreviewEnvironment`

Actually, this ignores any syntax argument. But don't tell anybody. But for the * form, it respects (actually ignores) any argument! Of course, we'll need to deactivate `\end{`⟨*environment*⟩`}` as well.

```
\def\PreviewEnvironment{\@ifstar\pr@starenv\pr@env}
\newcommand*\pr@starenv[1][]{\def\next##1##2{\pr@starmacro[{#1}][{##2}]##1}%
  \begingroup\pr@starenvii}
\newcommand*\pr@starenvii[2][]{\endgroup
  \expandafter\next\csname#2\endcsname{#1}%
  \expandafter\pr@starmacro\csname end#2\endcsname}
\newcommand*\pr@env[1][]{%
   \def\next[##1]##2{%
   \expandafter\pr@advise@ship\csname##2\endcsname{\pr@domacro{#1}%
```

```
      {\begingroup\aftergroup\pr@endbox##1}}{\endgroup}}%
  \@ifnextchar[\next\pr@macroii %]
}
```

`\PreviewSnarfEnvironment` This is a nuisance since we have to advise *both* the environment and its end.

```
\newcommand*{\PreviewSnarfEnvironment}[2][]{%
  \expandafter\pr@advise
   \csname #2\endcsname{\pr@snarfafter#1\pr@endparse}%
 \expandafter\pr@advise
   \csname end#2\endcsname{\endgroup}}
⟨/!active⟩
```

`\pr@snarfafter` `\pr@startsnarf` Ok, this looks complicated, but we have to start a group in order to be able to hook `\pr@endbox` into the game only when `\ifpr@outer` has triggered the start. And we need to get our start messages out before parsing the arguments.

```
⟨*active⟩
\def\pr@snarfafter{\ifpr@outer
     \pr@ship@start
     \let\pr@ship@start\@empty
   \fi
  \pr@callafter\pr@startsnarf}
\def\pr@startsnarf#1{#1\begingroup
   \pr@startbox{\begingroup\aftergroup\pr@endbox}{\endgroup}%
   \ignorespaces}
⟨/active⟩
```

`\pr@ship@start` `\pr@ship@end` The hooks `\pr@ship@start` and `\pr@ship@end` can be added to by option files by the help of the `\g@addto@macro` command from LaTeX, and by the `\pr@addto@front` command from `preview.sty` itself. They are called just before starting to process some preview, and just after it. Here is the policy for adding to them: `\pr@ship@start` is called inside of the vbox `\pr@box` before typeset material gets produced. It is, however, preceded by a break command that is intended for usage in `\vsplit`, so that any following glue might disappear. In case you want to add any material on the list, you have to precede it with `\unpenalty` and have to follow it with `\break`. You have make sure that under no circumstances any other legal breakpoints appear before that, and your material should contribute no nonzero dimensions to the page. For the policies of the `\pr@ship@end` hook, see the description on page 11.

```
⟨*!active⟩
\let\pr@ship@start\@empty
\let\pr@ship@end\@empty
```

`preview` `nopreview` First we write the definitions of these environments when `preview` is inactive. We will redefine them if `preview` gets activated.

```
\newenvironment{preview}{\ignorespaces}{\ifhmode\unskip\fi}
\newenvironment{nopreview}{\ignorespaces}{\ifhmode\unskip\fi}
```

We now process the options and finish in case we are not active.

```
\ProcessOptions\relax
\ifPreview\else\expandafter\endinput\fi
⟨/!active⟩
```

Now for the redefinition of the `preview` and `endpreview` environments:

```
⟨*active⟩
\renewenvironment{preview}{\begingroup
   \pr@startbox{\begingroup\aftergroup\pr@endbox}%
               {\endgroup}%
   \ignorespaces}%
   {\ifhmode\unskip\fi\endgroup}
\renewenvironment{nopreview}{\pr@outerfalse\ignorespaces}%
  {\ifhmode\unskip\fi}
```

Try to keep LaTeX from overwriting its information files:

```
\nofiles
```

Let the output routine throw everything gathered regularly away. Start with all float boxes, continue with output box, pack everything afloat from `\@currlist` back into `\@freelist`.

```
\output{\def\@elt#1{\global\setbox#1=\box\voidb@x}%
  \@currlist
  \@elt{255}%
  \let\@elt\relax
  \xdef\@freelist{\@currlist\@freelist}%
  \global\let\@currlist\@empty
  \deadcycles\z@}
```

`\pr@typeinfos` Then we have some document info that style files might want to output.

```
\def\pr@typeinfos{\typeout{Preview: Fontsize \f@size pt}%
  \ifnum\mag=\@m\else\typeout{Preview: Magnification \number\mag}\fi}
\AtBeginDocument{\pr@typeinfos}
```

And at the end we load the default configuration file, so that it may override settings from this package:

```
\pr@loadcfg{prdefault}
⟨/active⟩
⟨/style⟩
```

5 The option files

5.1 The `auctex` option

The AUC TeX option will cause error messages to spew. We want them on the terminal, but we don't want LaTeX to stop its automated run. We delay `\nonstopmode` in case the user has any pseudo-interactive folderol like reading in of file names in his preamble. Because we are so good-hearted, we will not break this as long as the document has not started, but after that we need the error message mechanism operative.

So here is the contents of the `prauctex.def` file:

```
⟨auctex⟩\ifPreview\else\expandafter\endinput\fi
⟨auctex⟩\preview@delay{\nonstopmode}
```

Ok, here comes creative error message formatting. It turns out a sizable portion of the runtime is spent in I/O. Making the error messages short is an advantage. It is not possible to convince TeX to make shorter error messages than this: TeX always wants to include context. This is about the shortest æsthetic one we can muster.

```
⟨auctex⟩\begingroup
⟨auctex⟩\lccode`\~=`\-
⟨auctex⟩\lccode`\{=`\<
⟨auctex⟩\lccode`\}=`\>
⟨auctex⟩\lowercase{\endgroup
⟨auctex⟩  \def\pr@msgi{{~}}}
⟨auctex⟩\def\pr@msgii{Preview:
⟨auctex⟩   Snippet \number\pr@snippet\space}
⟨auctex⟩\begingroup
⟨auctex⟩\catcode`\-=13
⟨auctex⟩\catcode`\<=13
⟨auctex⟩\@firstofone{\endgroup
⟨auctex⟩\def\pr@msg#1{{%
⟨auctex⟩   \let<\pr@msgi
⟨auctex⟩   \def-{\pr@msgii#1}%
⟨auctex⟩   \errhelp{Not a real error.}%
⟨auctex⟩   \errmessage<}}}
⟨auctex⟩\g@addto@macro\pr@ship@start{\pr@msg{started}}
⟨auctex⟩\g@addto@macro\pr@ship@end{\pr@msg{ended.%
⟨auctex⟩  (\number\ht\pr@box+\number\dp\pr@box x\number\wd\pr@box)}}
```

This looks pretty baffling, but it produces something short and semi-graphical, namely `<-><->`. That is a macro `<` that expands into `<->`, where `<` and `>` are the braces around an `\errmessage` argument and `-` is a macro expanding to the full text of the error message. Cough cough. You did not really want to know, did you?

Since over/underfull boxes are about the messiest things to parse, we disable them by setting the appropriate badness limits and making the variables point to junk. We also disable other stuff. While we set `\showboxbreadth` and `\showboxdepth` to indicate as little diagnostic output as possible, we keep them operative, so that the user retains the option of debugging using this stuff. The other variables concerning the generation of warnings and daignostics, however, are more often set by commonly employed packages and macros such as `\sloppy`. So we kill them off for good.

```
⟨auctex⟩\hbadness=\maxdimen
⟨auctex⟩\newcount\hbadness
⟨auctex⟩\vbadness=\maxdimen
⟨auctex⟩\let\vbadness=\hbadness
⟨auctex⟩\hfuzz=\maxdimen
⟨auctex⟩\newdimen\hfuzz
```

```
⟨auctex⟩\vfuzz=\maxdimen
⟨auctex⟩\let\vfuzz=\hfuzz
⟨auctex⟩\showboxdepth=-1
⟨auctex⟩\showboxbreadth=-1
```

Ok, now we load a possible configuration file.

```
⟨auctex⟩\pr@loadcfg{prauctex}
```

And here we cater for several frequently used commands in `prauctex.cfg`:

```
⟨auccfg⟩\PreviewMacro*[?[{@{[]}}{}][#1{}]\footnote
⟨auccfg⟩\PreviewMacro*[?[{@{[]}}{}]\item
⟨auccfg⟩\PreviewMacro*\emph
⟨auccfg⟩\PreviewMacro*\textrm
⟨auccfg⟩\PreviewMacro*\textit
⟨auccfg⟩\PreviewMacro*\textsc
⟨auccfg⟩\PreviewMacro*\textsf
⟨auccfg⟩\PreviewMacro*\textsl
⟨auccfg⟩\PreviewMacro*\texttt
⟨auccfg⟩\PreviewMacro*\textcolor
⟨auccfg⟩\PreviewMacro*\mbox
⟨auccfg⟩\PreviewMacro*[][#1{}]\author
⟨auccfg⟩\PreviewMacro*[][#1{}]\title
⟨auccfg⟩\PreviewMacro*\and
⟨auccfg⟩\PreviewMacro*\thanks
⟨auccfg⟩\PreviewMacro*[][#1{}]\caption
⟨auccfg⟩\preview@delay{\@ifundefined{pr@\string\@startsection}{%
⟨auccfg⟩   \PreviewMacro*[!!!!!!*]\@startsection}{}}
⟨auccfg⟩\PreviewMacro*\index
```

5.2 The `lyx` option

The following is the option providing LyX with info for its preview implementation.

```
⟨lyx⟩\ifPreview\else\expandafter\endinput\fi
⟨lyx⟩\pr@loadcfg{prlyx}
⟨lyx⟩\g@addto@macro\pr@ship@end{\typeout{Preview:
⟨lyx⟩  Snippet \number\pr@snippet\space
⟨lyx⟩  \number\ht\pr@box\space \number\dp\pr@box \space\number\wd\pr@box}}
```

5.3 The `counters` option

This outputs a checkpoint. We do this by saving all counter registers in backup macros starting with `\pr@c@` in their name. A checkpoint first writes out all changed counters (previously unchecked counters are not written out unless different from zero), then saves all involved counter values. LaTeX tracks its counters in the global variable `\cl@ckpt`.

```
⟨counters⟩\ifPreview\else\expandafter\endinput\fi
⟨counters⟩\def\pr@eltprint#1{\expandafter\@gobble\ifnum\value{#1}=0%
⟨counters⟩  \csname pr@c@#1\endcsname\else\relax
⟨counters⟩  \space{#1}{\arabic{#1}}\fi}
```

```
⟨counters⟩\def\pr@eltdef#1{\expandafter\xdef
⟨counters⟩  \csname pr@c@#1\endcsname{\arabic{#1}}}
⟨counters⟩\def\pr@ckpt#1{{\let\@elt\pr@eltprint\edef\next{\cl@@ckpt}%
⟨counters⟩  \ifx\next\@empty\else\typeout{Preview: Counters\next#1}%
⟨counters⟩  \let\@elt\pr@eltdef\cl@@ckpt\fi}}
⟨counters⟩\g@addto@macro\pr@ship@start{\pr@ckpt:}
⟨counters⟩\g@addto@macro\pr@ship@end{\pr@ckpt.}
```

5.4 Debugging options

Those are for debugging the operation of `preview`, and thus are mostly of interest for people that want to use `preview` for their own purposes. Since debugging output is potentially confusing to the error message parsing from AUC TeX, you should not turn on `\tracingonline` or switch from `\nonstopmode` unless you are certain your package will never be used with Preview-LaTeX.

The `showbox` option will generate diagnostic output for every produced box. It does not delay the resetting of the `\showboxbreadth` and `\showboxdepth` parameters so that you can still change them after the loading of the package. It does, however, move them to the end of the package loading, so that they will not be affected by the `auctex` option.

```
⟨showbox⟩\ifPreview\else\expandafter\endinput\fi
⟨showbox⟩\AtEndOfPackage{%
⟨showbox⟩  \showboxbreadth\maxdimen
⟨showbox⟩  \showboxdepth\maxdimen}
⟨showbox⟩\g@addto@macro\pr@ship@end{\showbox\pr@box}
```

The `tracingall` option is for the really heavy diagnostic stuff. For the reasons mentioned above, we do not want to change the setting of the interaction mode, nor of the `tracingonline` flag. If the user wants them different, he should set them outside of the preview boxes.

```
⟨tracingall⟩\ifPreview\else\expandafter\endinput\fi
⟨tracingall⟩\pr@addto@front\pr@ship@start{\let\tracingonline\count@
⟨tracingall⟩  \let\errorstopmode\@empty\tracingall}
```

5.5 Supporting conversions

It is not uncommon to want to use the results of `preview` as images. One possibility is to generate a flurry of EPS files with

`dvips -E -i -Ppdf -o` ⟨*outputfile*⟩`.000` ⟨*inputfile*⟩

However, in case those are to be processed further into graphic image files by GhostScript, this process is inefficient. One cannot use GhostScript in a single run for generating the files, however, since one needs to set the page size (or full size pages will be produced). The `tightpage` option will set the page dimensions at the start of each PostScript page so that the output will be sized appropriately.

That way, a single pass of Dvips followed by a single pass of GhostScript will be sufficient for generating all images.

You need to use to use the `dvips` option along with this option, or you'll get PostScript errors.

`\PreviewBorder`
`\PreviewBbAdjust`

We start this off with the user tunable parameters which get defined even in the case of an inactive package, so that redefinitions and assignments to them will always work:

```
⟨tightpage⟩\newdimen\PreviewBorder
⟨tightpage⟩\PreviewBorder=0.50001bp
⟨tightpage⟩\def\PreviewBbAdjust{-\PreviewBorder -\PreviewBorder
⟨tightpage⟩  \PreviewBorder \PreviewBorder}
```

Here is stuff used for parsing this:

```
⟨tightpage⟩\ifPreview\else\expandafter\endinput\fi
⟨tightpage⟩\def\pr@nextbb{\edef\next{\next\space\number\dimen@}%
⟨tightpage⟩  \advance\count@\m@ne\ifnum\count@>\z@
⟨tightpage⟩  \afterassignment\pr@nextbb\dimen@\fi}
```

And here is the stuff that we fudge into our hook. Of course, we have to do it in a box, and we start this box off with our special. There is one small consideration here: it might come before any `\AtBeginDvi` stuff containing header specials. It turns out Dvips rearranges this amicably: header code specials get transferred to the appropriate header section, anyhow, so this ensures that we come right after the bop section. We insert the 7 numbers here: the 4 bounding box adjustments, and the 3 TeX box dimensions. In case the box adjustments have changed since the last time, we write them out to the console.

```
⟨tightpage⟩\global\let\pr@bbadjust\@empty
⟨tightpage⟩\pr@addto@front\pr@ship@end{\begingroup
⟨tightpage⟩  \let\next\@gobble
⟨tightpage⟩  \count@4\afterassignment\pr@nextbb
⟨tightpage⟩  \dimen@\PreviewBbAdjust
⟨tightpage⟩  \ifx\pr@bbadjust\next
⟨tightpage⟩  \else \global\let\pr@bbadjust\next
⟨tightpage⟩  \typeout{Preview: Tightpage \pr@bbadjust}%
⟨tightpage⟩  \fi\endgroup}
⟨tightpage⟩\g@addto@macro\pr@ship@end{\setbox\pr@box\hbox{%
⟨tightpage⟩  \special{ps::\pr@bbadjust\space\number\ht\pr@box\space
⟨tightpage⟩  \number\dp\pr@box\space\number\wd\pr@box}\box\pr@box}}
```

Ok, here comes the beef. First we fish the 7 numbers from the file with `token` and convert them from TeX `sp` to PostScript points.

```
⟨tightpage⟩\preview@delay{\AtBeginDvi{%
⟨tightpage⟩  \special{!userdict begin/bop-hook{%
⟨tightpage⟩      7{currentfile token not{stop}if
⟨tightpage⟩        65781.76 div DVImag mul}repeat
```

Next we produce the horizontal part of the bounding box as

$$(1\text{in}, 1\text{in}) + \big(\min(\texttt{\textbackslash wd\textbackslash pr@box}, 0), \max(\texttt{\textbackslash wd\textbackslash pr@box}, 0)\big)$$

and roll it to the bottom of the stack:

```
⟨tightpage⟩      72 add 72 2 copy gt{exch}if 4 2 roll
```

Next is the vertical part of the bounding box. Depth counts in negatively, and we again take min and max of possible extents in the vertical direction, limited by 0. 720 corresponds to 10 in and is the famous 1 in distance away from the edge of letterpaper.

```
⟨tightpage⟩      neg 2 copy lt{exch}if dup 0 gt{pop 0 exch}%
⟨tightpage⟩      {exch dup 0 lt{pop 0}if}ifelse 720 add exch 720 add
⟨tightpage⟩      3 1 roll
```

Ok, we now have the bounding box on the stack in the proper order llx, lly, urx, ury. We add the adjustments:

```
⟨tightpage⟩    4{5 -1 roll add 4 1 roll}repeat
```

The page size is calculated as the appropriate differences, the page offset consists of the coordinates of the lower left corner, with those coordinates negated that would be reckoned positive in the device coordinate system.

```
⟨tightpage⟩      <</PageSize[5 -1 roll 6 index sub 5 -1 roll 5 index sub]%
⟨tightpage⟩        /PageOffset[7 -2 roll [1 1 dtransform exch]%
⟨tightpage⟩        {0 ge{neg}if exch}forall]>>setpagedevice%
```

So we now bind the old definition of `bop-hook` into our new definition and finish it.

```
⟨tightpage⟩      //bop-hook exec}bind def end}}}%
```

5.6 The `showlabels` option

During the editing process, some people like to see the label names in their equations, figures and the like. Now if you are using Emacs for editing, and in particular Preview-LaTeX, I'd strongly recommend that you check out the RefTeX package which pretty much obliterates the need for this kind of functionality. If you still want it, standard LaTeX provides it with the `showkeys` package, and there is also the less encompassing `showlabels` package. Unfortunately, since those go to some pain not to change the page layout and spacing, they also don't change `preview`'s idea of the TeX dimensions of the involved boxes.

So those packages are mostly useless. So we present here an alternative hack that will get the labels through.

`\pr@labelbox` This works by collecting them into a separate box which we then tack to the right of the previews.

```
⟨showlabels⟩\ifPreview\else\expandafter\endinput\fi
⟨showlabels⟩\newbox\pr@labelbox
```

`\pr@label` We follow up with our own definition of the `\label` macro which will be active only in previews. The original definition is stored in `\pr@@label`. `\pr@lastlabel` contains the last typeset label in order to avoid duplication in certain environments, and we keep the stuff in `\pr@labelbox`.

```
⟨showlabels⟩\def\pr@label#1{\pr@@label{#1}%
```

Ok, now we generate the box, by placing the label below any existing stuff.

```
⟨showlabels⟩    \ifpr@setbox\z@{#1}%
⟨showlabels⟩      \global\setbox\pr@labelbox\vbox{\unvbox\pr@labelbox
⟨showlabels⟩       \box\z@}\egroup\fi}
```

`\ifpr@setbox` \ifpr@setbox receives two arguments, #1 is the box into which to set a label, #2 is the label text itself. If a label needs to be set (if it is not a duplicate in the current box, and is nonempty, and we are in the course of typesetting and so on), we are left in a true conditional and an open group with the preset box. If nothing should be set, no group is opened, and we get into skipping to the closing of the conditional. Since \ifpr@setbox is a macro, you should not place the call to it into conditional text, since it will not pair up with \fi until being expanded.

We have some trickery involved here. \romannumeral\z@ expands to empty, and will also remove everything between the two of them that also expands to empty, like a chain of \fi.

```
⟨showlabels⟩\def\ifpr@setbox#1#2{%
⟨showlabels⟩  \romannumeral%
⟨showlabels⟩  \ifx\protect\@typeset@protect\ifpr@outer\else
```

Ignore empty labels. . .

```
⟨showlabels⟩    \z@\bgroup
⟨showlabels⟩    \protected@edef\next{#2}\@onelevel@sanitize\next
⟨showlabels⟩    \ifx\next\@empty\egroup\romannumeral\else
```

and labels equal to the last one.

```
⟨showlabels⟩    \ifx\next\pr@lastlabel\egroup\romannumeral\else
⟨showlabels⟩    \global\let\pr@lastlabel\next
⟨showlabels⟩    \setbox#1\pr@boxlabel\pr@lastlabel
⟨showlabels⟩    \expandafter\expandafter\romannumeral\fi\fi\fi\fi
⟨showlabels⟩    \z@\iffalse\iftrue\fi}
```

`\pr@boxlabel` Now the actual typesetting of a label box is done. We use a small typewriter font inside of a framed box (the default frame/box separating distance is a bit large).

```
⟨showlabels⟩\def\pr@boxlabel#1{\hbox{\normalfont
⟨showlabels⟩    \footnotesize\ttfamily\fboxsep0.4ex\relax\fbox{#1}}}
```

`\pr@maketag` And here is a version for amsmath equations. They look better when the label is right beside the tag, so we place it there, but augment \box\pr@labelbox with an appropriate placeholder.

```
⟨showlabels⟩\def\pr@maketag#1{\pr@@maketag{#1}%
⟨showlabels⟩  \ifpr@setbox\z@{\df@label}%
⟨showlabels⟩       \global\setbox\pr@labelbox\vbox{%
⟨showlabels⟩          \hrule\@width\wd\z@\@height\z@
⟨showlabels⟩          \unvbox\pr@labelbox}%
```

Set the width of the box to empty so that the label placement gets not disturbed, then append it.

```
⟨showlabels⟩         \wd\z@\z@\box\z@ \egroup\fi}
```

`\pr@lastlabel` Ok, here is how we activate this: we clear out box and label info

```
⟨showlabels⟩\g@addto@macro\pr@ship@start{%
⟨showlabels⟩  \global\setbox\pr@labelbox\box\voidb@x
⟨showlabels⟩  \xdef\pr@lastlabel{}%
```

The definitions above are global because we might be in any amount of nesting. We then reassign the appropriate labelling macros:

```
⟨showlabels⟩  \let\pr@@label\label \let\label\pr@label
⟨showlabels⟩  \let\pr@@maketag\maketag@@@ \let\maketag@@@\pr@maketag
⟨showlabels⟩}
```

Now all we have to do is to add the stuff to the box in question.

```
⟨showlabels⟩\pr@addto@front\pr@ship@end{%
⟨showlabels⟩   \ifvoid\pr@labelbox
⟨showlabels⟩   \else \setbox\pr@box\hbox{%
⟨showlabels⟩         \box\pr@box\,\box\pr@labelbox}%
⟨showlabels⟩   \fi}
```

5.7 The `footnotes` option

This is rather simplistic right now. It overrides the default footnote action (which is to disable footnotes altogether for better visibility).

```
⟨footnotes⟩\PreviewMacro[[!]\footnote %]
```

6 Various driver files

The installer, in case it is missing. If it is to be used via `make`, we don't specify an installation path, since

```
make install
```

is supposed to cater for the installation itself.

```
⟨installer⟩ \input docstrip
⟨installer & make⟩ \askforoverwritefalse
⟨installer⟩ \generate{
⟨installer⟩    \file{preview.drv}{\from{preview.dtx}{driver}}
⟨installer&!make⟩    \usedir{tex/latex/preview}
⟨installer⟩    \file{preview.sty}{\from{preview.dtx}{style}
⟨installer⟩                      \from{preview.dtx}{style,active}}
⟨installer⟩    \file{prauctex.def}{\from{preview.dtx}{auctex}}
⟨installer⟩    \file{prauctex.cfg}{\from{preview.dtx}{auccfg}}
⟨installer⟩    \file{prshowbox.def}{\from{preview.dtx}{showbox}}
⟨installer⟩    \file{prshowlabels.def}{\from{preview.dtx}{showlabels}}
⟨installer⟩    \file{prtracingall.def}{\from{preview.dtx}{tracingall}}
⟨installer⟩    \file{prtightpage.def}{\from{preview.dtx}{tightpage}}
⟨installer⟩    \file{prlyx.def}{\from{preview.dtx}{lyx}}
⟨installer⟩    \file{prcounters.def}{\from{preview.dtx}{counters}}
⟨installer⟩    \file{prfootnotes.def}{\from{preview.dtx}{footnotes}}
⟨installer⟩ }
⟨installer⟩ \endbatchfile
```

And here comes the documentation driver.

```
⟨driver⟩ \documentclass{ltxdoc}
⟨driver⟩ \newcommand\previewlatex{Preview-\LaTeX}
⟨driver⟩ \begin{document}
⟨driver⟩ \DocInput{preview.dtx}
⟨driver⟩ \end{document}
```