The locpream package: Local preambles within LaTeX documents*

Michael D. Adams
https://michaeldadams.org

Abstract

The `locpream` package allows the use of preambles that are local to specific parts of a document. This is useful for using symbols from packages without loading those packages into the main document. For example, `locpream` allows the simultaneous use of symbols from packages that conflict with each other or otherwise cannot be loaded together.

Contents

*This document corresponds to locpream v0.1.0, dated 2019/01/05.

1 Introduction

The `locpream` package allows the use of preambles that are local to specific parts of a document. For example, you may want to use symbols from packages that cannot be loaded at the same time. By compiling them in separate documents and then inserting their compiled results into a master document, `locpream` allows you to use these symbols without conflicts.

This document serves as both the documentation and test suite for the `locpream` package, so some examples in this document are for testing not just documentation.

Warning

This package is pre-alpha, and its interface may change without notice.

This document is organized as follows.

- Section 1 (Introduction) contains:
  - an introduction to using this package (Section 1.1: Quick Start),
  - an explanation of the notation used in this document (Section 1.2: Notation), and
  - a comparison of this package to other, similar packages (Section 1.3: Related Packages).
- Section 2 (High-level Commands) contains documentation for the high-level commands and environments of this package, namely:
  - `\localpreamble`→ P. 7,
  - `localpreambleenv`→ P. 7,
  - `\newlocalpreamble`→ P. 8, and
  - `\newlocalpreambleenv`→ P. 8.
- Section 3 (Low-level Commands) contains documentation for the low-level commands of this package, namely,
  - `\LocalPreambleWrite`→ P. 8,

    * `\LocalPreambleCompile` → P. 8,
    * `\LocalPreambleRead` → P. 9, and
    * `\LocalPreambleCode` → P. 9.

– Section 4 (Options) contains documentation for the options that can be passed to the commands and environments of this package.

– Section 5 (Issues and Workarounds) contains common issues and how to fix or work around them.

– Section 6 (Changelog) contains a list of the notable changes for each version of this package.

1.1 Quick Start

Suppose you want to use the `\textbeta` symbol from the `textgreek` package, but you do not want to load the `textgreek` package into your document. Maybe it is incompatible with some other package that you use or maybe LaTeX has run out of space for declaring new fonts. You can render `\textbeta` using the `\localpreamble` → P. 7 command as follows.

```
An atom undergoing
\localpreamble[preamble={\usepackage{textgreek}}]{\textbeta-decay}
can emit an electron.
```

An atom undergoing β-decay can emit an electron.

If you prefer, you can also use the environment `localpreambleenv` → P. 7 as follows.

```
An atom undergoing
\begin{localpreambleenv}[preamble={\usepackage{textgreek}}]
  \textbeta-decay
\end{localpreambleenv}
~can emit an electron.
```

An atom undergoing β-decay can emit an electron.

The only difference between `\localpreamble` → P. 7 and `localpreambleenv` → P. 7 is that one is a command and the other is an environment.

If you want to format the LaTeX code as "display" math, use the `math=display` option as follows.

```
The solution to the two-dimensional integral
\localpreamble[preamble={\usepackage{amsmath}}, math=display]
  {\iint xy\,dx\,dy}
involves $x^2$ and $y^2$.
```

The solution to the two-dimensional integral

$$\iint xy\,dx\,dy$$

involves x^2 and y^2.

Note that even though these examples are compiled as separate LaTeX documents, they are automatically properly spaced relative to the surrounding text. In the resulting PDF, they even behave properly with regard to search, text selection, and copy-and-paste.

For example, in the following, it is impossible to tell the difference between the "y" generated from `\localpreamble` → P. 7 and the one that is not. (This even handles the descender on the "y".)

```
x\localpreamble{y}z xyz
```

xyz xyz

However, there is a limitation to this when it comes to kerning and ligatures. For example, in the following, due to kerning, "T" and "e" have a different amount of space between them when using `\localpreamble` → P. 7 and when not, and putting the second "f" in `\localpreamble` → P. 7 breaks up the "ffi" ligature.

```
Teffi T\localpreamble{e}f\localpreamble{f}i
```

Teffi Teffi

1.2 Notation

For testing purposes, this documentation surrounds most examples with < and >, which are customized to render as ⁞ and ⁞. These are used as gauges/registration marks. They make it easy to see whether there is extra space to the left or right of a symbol and whether parts of the symbol extend below the baseline. The bottom of the bottom bar is at the baseline. The top of the top, middle and bottom bars are at the font size (10 points), the top of an "M" (7 points), and the top of an "x" (4.5 points), respectively. This is demonstrated in the following example.

```
<\rule{2pt}{10pt}> <\rule{2pt}{7pt}> <\rule{2pt}{4.5pt}> <M> <x> <>
```

1.3 Related Packages

There are a number of packages that provide similar functionality to that of `locpream`. The main distinguishing features of `locpream` are that it allows locally specified preambles and that it does exact spacing and sizing.

1.3.1 Externalization

A number of packages allow specified parts of a document to be "externalized" by compiling those parts as a separate LaTeX documents and then including the result in the master document. However, none of these allow those document parts to have a local preamble.

`tikz` Contains an `external` library, which caches the results of compiling each `tikzpicture` in order to improve later compilation times. Has no support for those pictures having different preambles from the main document.

`pgfplots` Contains an `external` library similar to the `tikz external` library.

`preview` Allows extracting certain environments from LaTeX sources as graphics. Intended for rendering "preview" versions of parts of a document and used as part of `preview-latex` and AUCTeX. Does not support custom preambles or exact spacing.

1.3.2 Miniature Documents

`minidocument` Provides the `minidocument` environment, which allows an entire miniature document to be embedded including a separate `\documentclass` and preamble. However, assumes documents are entire pages and does not support exact spacing.

1.3.3 Ignoring Preambles

Several packages allow sub-documents to be included in a master document. These sub-documents are full-fledged LaTeX documents that can be compiled on their own, but when they are included in the master document, either their preamble or specified parts are ignored. These packages are geared towards helping authors manage large documents that otherwise take a long time to compile (e.g., a book with each chapter as a sub-document). When compiling the master document, these packages do not compile the sub-documents separately. Thus, they do not support local preambles when compiling the master document.

`docmute` Redefines `\input` and `\include` to ignore everything between the initial `\documentclass` and `\begin{document}`.

`includex` Defines versions of `\include` that ignore certain parts of the included document. Not only can it ignore everything between `\documentclass` and `\begin{document}`, but it also allows ignoring anything outside specified environments. The documentation for this package says that it is currently unsupported.

`newclude` Adds various features to the LaTeX inclusion system. One of those features is the ability to ignore anything outside specified environments. The documentation for this package says that it is intended to subsume the features of `includex`, but it also marks these features as in development.

1.3.4 Copying Preambles from Master Files

Another approach to sub-documents is to have those sub-documents automatically copy the preamble of the master file. As with the packages in Section 1.3.3, these packages are geared towards helping authors manage large documents that otherwise take a long time to compile (e.g., a book with each chapter as a sub-document). When compiling the master document, these packages do not compile the sub-documents separately. Thus, they do not support local preambles when compiling the master document.

`subfiles` Defines a document class for sub-documents that automatically copies the preamble of the master document when the sub-document is compiled, but when the master document is compiled, everything outside the `document` environment is ignored.

`childdoc` Defines commands to be put in sub-documents instead of the standard `\documentclass` and preamble. When the sub-document is separately compiled, this automatically copies the preamble of the master document, but when the master document is compiled, the commands are ignored.

1.3.5 Including Entire Pages

Some packages are designed to allow completely separate documents to be combined into one document (e.g., combining multiple articles into a proceedings). Each document has its own preamble, but documents are included a whole page at a time.

`combine` Allows assembling a group of individual LaTeX documents into a single document. Also includes the `combinet` and `combnat` package to allow documents to share things like the table of contents.

`subdocs` Allows combining documents where each sub-document is a complete, normal LaTeX document that is typeset separately. Shares the `.aux` files between documents so thing like the table of contents can be kept in common.

1.3.6 Listing Code and Displaying the Results

`tcolorbox`, `example`, `examplep`, `latexdemo`, and `showexpl` These packages all provide environments that display their content as source code next to the result of rendering that code. None of these involve separate compilation or allow for specifying a local preamble.

1.3.7 Minimal Page Layouts

`standalone` Provides the document class `standalone` that produces a page with minimal layout that is sized to fit its contents. Does not support automatic extraction of standalone documents, and has no support for exact spacing or including the results of compiling a standalone document.

2 High-level Commands

2.1 `\localpreamble` and `localpreambleenv`

The main commands of this package are `\localpreamble` and `localpreambleenv`.

`\localpreamble[`⟨*options*⟩`]{`⟨*code*⟩`}`

Externally compiles the LaTeX code in ⟨*code*⟩ and then includes the result with exact spacing. This allows ⟨*code*⟩ to have its own preamble. An example of this is the following.

```
<\localpreamble[preamble={\usepackage{amsmath}}, math=inline]
   {\iint xy\,dx\,dy}>
```

$$\iint xy\,dx\,dy$$

`\begin{localpreambleenv}[`⟨*options*⟩`]`
⟨*environment content*⟩
`\end{localpreambleenv}`

The same as `\localpreamble`. The only difference between the two is that `\localpreamble` is a command and `localpreambleenv` is an environment. An example of this is the following.

```
<\begin{localpreambleenv}[preamble={\usepackage{amsmath}}, math=inline]
   \iint xy\,dx\,dy
 \end{localpreambleenv}>
```

$\iint xy\,dx\,dy$

2.2 `\newlocalpreamble` and `\newlocalpreambleenv`

It is sometimes useful to define versions of the `\localpreamble` → P. 7 command and the `localpreambleenv` → P. 7 environment that have customized default values for their options. For example, you might want to define versions that load the `amsmath` package by default. These can be created with `\newlocalpreamble` and `\newlocalpreambleenv`.

`\newlocalpreamble[`⟨*options*⟩`]{`⟨*command name*⟩`}`

With `\newlocalpreamble`, one can define a version of `\localpreamble` → P. 7 that has customized default values for its options as seen in the following example.

```
<\newlocalpreamble[preamble={\usepackage{amsmath}}, math=inline]{\ams}>
<\ams{\iint xy\,dx\,dy}>
<\ams[math=false]{$\iint xyzw\,dx\,dy$}>
```

$\iint xy\,dx\,dy$ $\iint xyzw\,dx\,dy$

`\newlocalpreambleenv[`⟨*options*⟩`]{`⟨*command name*⟩`}`

With `\newlocalpreambleenv`, one can define a version of `localpreambleenv` → P. 7 that has customized default values for its options as seen in the following example.

```
<\newlocalpreambleenv[preamble={\usepackage{amsmath}}, math=inline]
   {amsenv}>
<\begin{amsenv}\iint xy\,dx\,dy\end{amsenv}>
<\begin{amsenv}[math=false]$\iint xyzw\,dx\,dy$\end{amsenv}>
```

$\iint xy\,dx\,dy$ $\iint xyzw\,dx\,dy$

3 Low-level Commands

`\LocalPreambleWrite[`⟨*options*⟩`]{`⟨*body*⟩`}`

`\LocalPreambleCompile[`⟨*options*⟩`]`

\LocalPreambleRead[⟨*options*⟩]

Both the \localpreamble → P. 7 command and the localpreambleenv → P. 7 environment contain three phases:

1. writing the intermediate LaTeX file,
2. compiling the intermediate LaTeX file, and
3. reading the file (typically a PDF) that results from that compilation.

These are handled by \LocalPreambleWrite, \LocalPreambleCompile, and \LocalPreambleRead, respectively. For example, instead of using the command \localpreamble → P. 7, you could explicitly call each of these as in the following.

```
<\LocalPreambleWrite[file=locpream-locpream-separate,
                     preamble={\usepackage{amsmath}}]
   {$\iint xy\,dx\,dy$}>
<\LocalPreambleCompile[file=locpream-locpream-separate]>
<\LocalPreambleRead[file=locpream-locpream-separate]>
```

$$\iint xy\,dx\,dy$$

Taking explicit control of these is particularly useful if you want to cache compilation results. See the \LocalPreambleCode command for an example of this.

\LocalPreambleCode{⟨*dimension file*⟩}{⟨*preamble*⟩}{⟨*before savebox*⟩}{⟨*body*⟩}
{⟨*after savebox*⟩}{⟨*before usebox*⟩}{⟨*after usebox*⟩}

This command expands to the code used in the intermediate LaTeX file. It is useful if you want to store the LaTeX code to be compiled in a separate file and reuse the compiled results between compilations of the master LaTeX file.

The ⟨*dimension file*⟩ is the full filename of the dimension file to be generated. The ⟨*body*⟩ is the LaTeX code to be compiled. The ⟨*preamble*⟩, ⟨*before savebox*⟩, ⟨*after savebox*⟩, ⟨*before usebox*⟩, and ⟨*after usebox*⟩ are the same as the corresponding options in Section 4.

For example, you might write the following file.

locpream-standalone-simple.tex

```
\RequirePackage{locpream.code}
\LocalPreambleCode
  {locpream-standalone-simple.dim} % dimension file
  {\documentclass{article}\usepackage{amsmath}} % preamble
  {} % before savebox
  {$\iint xy\,dx\,dy$} % body
  {} % after savebox
  {} % before usebox
  {} % after usebox
```

Then, in your master file you can compile and read that file with the following commands.

```
<\LocalPreambleCompile[file=locpream-standalone-simple]>
<\LocalPreambleRead[file=locpream-standalone-simple]>
```

$$\iint xy\,dx\,dy$$

Be careful if you rename such a file, as you will need to change the ⟨*dimension file*⟩ argument to match. Otherwise, you will get an error along the lines of:

`In \LocalPreambleRead, input dimension file does not exist`

Also note that `\LocalPreambleCode` is defined in the `locpream.code` package. This package is imported by the main `locpream` package, so you do not need to import it separately. However, `locpream.code` is designed be minimal and has no dependencies. Thus in the previous example, using the command `\RequirePackage{locpream.code}` instead of the command `\RequirePackage{locpream}` minimizes the compilation time. When there are a large number of standalone files, this difference can amount to a significant amount of time.

If you want to reuse compiled results between compilations of the master LaTeX file, you will want to manually run the following command.

```
pdflatex -shell-escape locpream-standalone-simple.tex
```

Then you would omit the call to `\LocalPreambleCompile` → P. 8 and just call `\LocalPreambleRead` → P. 9 as in the following.

```
<\LocalPreambleRead[file=locpream-standalone-simple]>
```

$$\iint xy\,dx\,dy$$

4 Options

Options that are passed to commands are parsed using the `keyval` package. Their syntax is `[`⟨*key*$_1$⟩`=`⟨*value*$_1$⟩`,` ⟨*key*$_2$⟩`=`⟨*value*$_2$⟩`,` $\cdots$`,` ⟨*key*$_n$⟩`=`⟨*value*$_n$⟩`]`, where white space around '`,`' and '`=`' is ignored.

`\locpreamkeys{`⟨*options*⟩`}`

You can set the default values for options with the `\locpreamkeys` command. For example, if want to default to use `mypdflatex` instead of `pdflatex` to compile LaTeX code, you could use the following command.

```
\locpreamkeys{latex=mylatex}
```

TEST

You can also specify this by passing the option when the `locpream` package is loaded as seen in the following example.

```
\usepackage[latex=mylatex]{locpream}
```

TEST

4.1 Commonly Used Options

`documentclass`=⟨*class*⟩ (initially `article`)

This option specifies the document class used (by way of `\documentclass`) by the intermediate LaTeX file that is generated for each piece of code with a local preamble. For example, the following uses the `proc` class, which (unlike `article`) contains the `\pagename` macro.

```
<\localpreamble[documentclass=proc]{\pagename}>
```

If the value of this key is blank, a `\documentclass` declaration is not added to the intermediate file. In this case, you will likely want to put a `\documentclass` declaration in the `preamble` option as in the following example. (This could also be achieved with `documentclass=prog`, so doing it this way is gratuitous and solely for the sake of example.)

```
<\localpreamble[documentclass={}, preamble={\documentclass{proc}}]
   {\pagename}>
```

`documentclass/options`=⟨*options*⟩ (initially empty)

This option specifies options to pass to the document class. For example, the following specifies passing the `12pt` option to `article`, which changes the default font to be 12 points tall.

```
<\localpreamble[documentclass/options={12pt}]{M}>
```

`preamble`=⟨*code*⟩ (initially empty)

This options specifies LaTeX code to be put in the preamble of the intermediate LaTeX file. For example, you might want to load packages as in the following.

```
<\localpreamble[preamble={\usepackage{amsmath}}]{$\iint xy\,dx\,dy$}>
```

$\iint xy\,dx\,dy$

`math=false`, `inline`, or `display` (initially `false`)

This option controls whether the code in the body of `\localpreamble` → P. 7 or `localpreambleenv` → P. 7 is treated as math, and if so, whether it is inline math or display math. The following examples demonstrate the possible values for this option.

```
<\localpreamble[preamble={\usepackage{amsmath}}, math=false]
   {$\iint xy\,dx\,dy$}>
```

$\iint xy\,dx\,dy$

```
<\localpreamble[preamble={\usepackage{amsmath}}, math=inline]
   {\iint xy\,dx\,dy}>
```

$\iint xy\,dx\,dy$

```
<\localpreamble[preamble={\usepackage{amsmath}}, math=display]
   {\iint xy\,dx\,dy}>
```

$$\iint xy\,dx\,dy$$

Note that, as seen in the following example, `math=display` is equivalent to the incantation `\[\localpreamble{$\displaystyle`⟨*code*⟩`$}\]`.

```
<\[\localpreamble[preamble={\usepackage{amsmath}}]
     {$\displaystyle\iint xy\,dx\,dy$}\]>
```

$$\iint xy\,dx\,dy$$

`margin/top`=⟨*length*⟩ (initially `1in`)

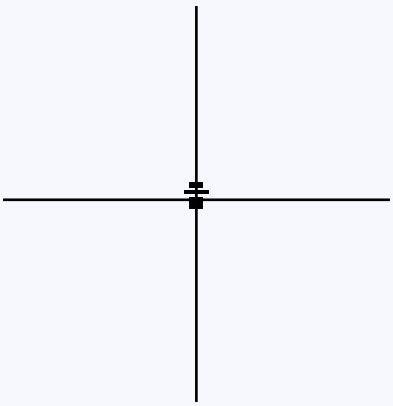
`margin/bottom`=⟨*length*⟩ (initially `1in`)

`margin/left`=⟨*length*⟩ (initially `1in`)

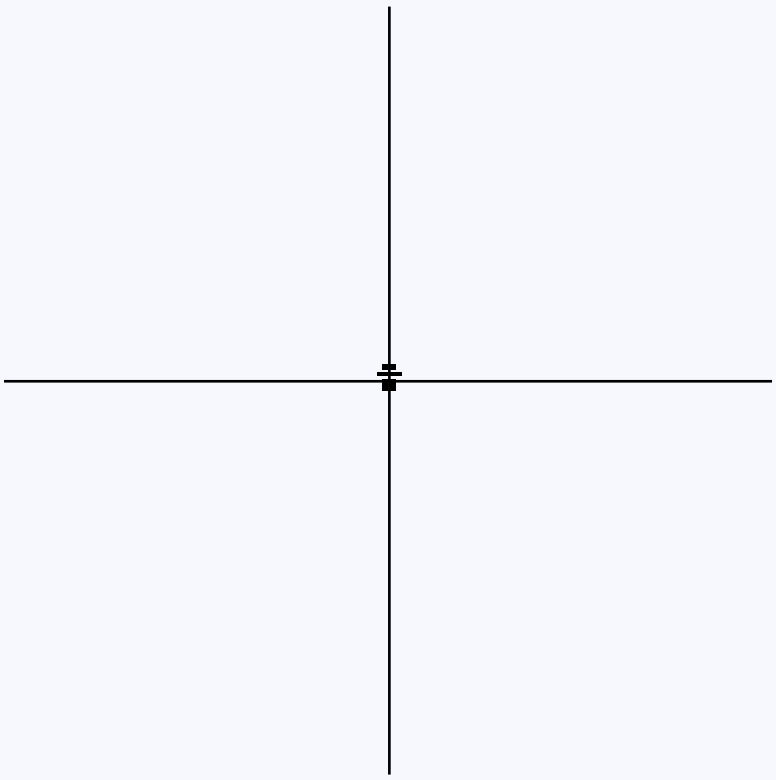
`margin/right`=⟨*length*⟩ (initially `1in`)

These options specify the margin to place around the LaTeX code being separately compiled. This is useful if the LaTeX code draws outside its bounding box. If there is not enough margin to contain the drawn portions, the result may be clipped. For example, compare the two following examples. The 2-inch rules are clipped when the default 1-inch margins are used but are not clipped when 3-inch margins are used.

```
\vspace{2in}
\hspace{2in}
<\begin{localpreambleenv}
   \rlap{\rule[3pt]{2in}{1pt}}%
   \llap{\rule[3pt]{2in}{1pt}}%
   \smash{\rule[-2in]{1pt}{4in}}%
 \end{localpreambleenv}>
\vspace{2.2in}
```



```
\vspace{2in}
\hspace{2in}
<\begin{localpreambleenv}[
   margin/top=3in, margin/bottom=3in,
   margin/left=3in, margin/right=3in]
   \rlap{\rule[3pt]{2in}{1pt}}%
   \llap{\rule[3pt]{2in}{1pt}}%
   \smash{\rule[-2in]{1pt}{4in}}%
 \end{localpreambleenv}>
\vspace{2.2in}
```



`debug=true` or `false` (initially `false`)

Whether to print tracing information to standard out. This is helpful in determining exactly what part of a command failed.

For example, consider the following call to `\localpreamble` → P. 7.

```
<\localpreamble[preamble={\usepackage{amsmath}}]{$\iint xy\,dx\,dy$}>
```

$\iint xy\,dx\,dy$

When the `debug` option is `true`, lines like the following will be printed to the standard output.

```
**** Begin \LocalPreambleWrite on {locpream-locpream-21}
       with {$\iint xy\,dx\,dy$}
**** End \LocalPreambleWrite on {locpream-locpream-21}
**** Begin \LocalPreambleCompile on {locpream-locpream-21}
**** End \LocalPreambleCompile on {locpream-locpream-21}
**** Begin \LocalPreambleRead on {locpream-locpream-21}
**** End \LocalPreambleRead on {locpream-locpream-21}
```

4.2 Uncommonly Used Options

`before`=⟨*code*⟩ (initially `\stepcounter{locpream@number}`)

`after`=⟨*code*⟩ (initially empty)

These options specify LaTeX code to run before or after the rest of the code in a `\localpreamble` → P. 7 command or a `localpreambleenv` → P. 7 environment. This is particularly useful for incrementing any counters used in the `file` → P. 19 option. The following is an example of these in action.

```
\newcounter{foo}%
\thefoo
<\localpreamble[file=locpream-locpream-before,
                before={\stepcounter{foo}}]{}>%
\thefoo
<\localpreamble[file=locpream-locpream-after,
                after={\stepcounter{foo}}]{}>%
\thefoo
```

0 1 2

`before write`=⟨*code*⟩ (initially empty)

`after write`=⟨*code*⟩ (initially empty)

`before compile`=⟨*code*⟩ (initially empty)

`after compile`=⟨*code*⟩ (initially empty)

`before read`=⟨*code*⟩ (initially empty)

`after read`=⟨*code*⟩ (initially empty)

These options specify code to run before or after `\LocalPreambleWrite` → P. 8, `\LocalPreambleCompile` → P. 8, or `\LocalPreambleRead` → P. 9. The following is an example of these in action.

```
\newcounter{baz}%
\thebaz
<\localpreamble[file=locpream-locpream-before-read,
                before read={\stepcounter{baz}}]{}>%
\thebaz
<\localpreamble[file=locpream-locpream-after-read,
                after read={\stepcounter{baz}}]{}>%
\thebaz
<\localpreamble[file=locpream-locpream-before-compile,
                before compile={\stepcounter{baz}}]{}>%
\thebaz
<\localpreamble[file=locpream-locpream-after-compile,
                after compile={\stepcounter{baz}}]{}>%
\thebaz
<\localpreamble[file=locpream-locpream-before-write,
                before write={\stepcounter{baz}}]{}>%
\thebaz
<\localpreamble[file=locpream-locpream-after-write,
                after write={\stepcounter{baz}}]{}>%
\thebaz
```

0 1 2 3 4 5 6

`before savebox`=⟨*code*⟩ (initially empty)

`after savebox`=⟨*code*⟩ (initially empty)

These options specify code to be put before or after the `\savebox` that is used in the intermediate LaTeX file. These options correspond to the ⟨*before savebox*⟩ and ⟨*after savebox*⟩ arguments of `\LocalPreambleCode` → P. 9.

These options are rarely needed.

The following example demonstrates the use of these options, though since there are other ways to accomplish this effect, doing it in this way is gratuitous and solely for the sake of example. Note that we have to set the margins to small or zero lengths to prevent the gray area from overlapping the rest of the page.

```
<\localpreamble[preamble={\usepackage{xcolor}},
                before savebox={\newcommand{\p}[1]{(##1)}},
                after savebox={\pagecolor{gray!50}},
                margin/top=1pt, margin/bottom=1pt,
                margin/left=0pt, margin/right=0pt]
  {\p{x}}>
```

(x)

`before usebox`=⟨*code*⟩ (initially empty)

`after usebox`=⟨*code*⟩ (initially empty)

These options specify code to be put before or after the `\usebox` that is used in the intermediate LaTeX file. These options correspond to the ⟨*before usebox*⟩ and ⟨*after usebox*⟩ arguments of `\LocalPreambleCode`→ P. 9.

These options are rarely needed.

The following example demonstrates the use of these options, though since there are other ways to accomplish this effect, doing it this way is gratuitous and solely for the sake of example. Note that we have to set the margins to small or zero lengths to prevent the gray area from overlapping the rest of the page.

```
<\localpreamble[preamble={\usepackage{xcolor}},
                before usebox={\pagecolor{gray!50}},
                margin/top=1pt, margin/bottom=1pt,
                margin/left=0pt, margin/right=0pt]
   {ABC}>
<\localpreamble[preamble={\usepackage{xcolor}},
                after usebox={\pagecolor{gray!50}},
                margin/top=1pt, margin/bottom=1pt,
                margin/left=0pt, margin/right=0pt]
   {DEF}>
```

ABC DEF

`latex`=⟨*program name*⟩ (initially empty)

This option specifies the program used to compile the intermediate LaTeX file. Blank means to autodetect between `pdflatex`, `xelatex`, or `lualatex` to match whatever the master document is being compiled with. For example, if you want to force certain code to run under `pdflatex` or `lualatex`, you can do the following.

```
<\localpreamble[latex=pdflatex]{\pdfescapehex{ABC}}>
<\localpreamble[latex=lualatex]{\directlua{tex.print("ABC")}}>
```

414243 ABC

`latex/options`=⟨*code*⟩ (initially `-halt-on-error -interaction=batchmode`)

This option specifies what command-line options to pass to LaTeX when compiling the intermediate LaTeX file.

Note that if you change this, you will almost certainly want to include the `-halt-on-error` and `-interaction=batchmode` options in whatever you change it to.

For example, the `ifplatform` package needs the `-shell-escape` option in order to give precise platform information as seen in the following.

```
<\localpreamble[preamble={\usepackage{ifplatform}}]
   {\platformname}>
<\localpreamble[preamble={\usepackage{ifplatform}},
                latex/options={-shell-escape -halt-on-error
                               -interaction=batchmode}]
   {\platformname}>
```

*NIX Linux

Note that if the documentation (i.e., the document you are reading) is compiled on Windows, then the two calls to `\localpreamble` → P. 7 in this example produce the same results as each other, but on any other platform they are different.

TEST

`file=`⟨*basename*⟩ (initially `\jobname-locpream-\arabic{locpream@number}`)

This option is used by the default values of `file/tex file` → P. 20, `file/dim file` → P. 20, and `file/out file` → P. 20 to specify the basename of the intermediate files that are generated. For example, the following uses `locpream-locpream-file` as the basename.

```
<\localpreamble[file=locpream-locpream-file,
                preamble={\usepackage{amsmath}}]
   {$\iint xy\,dx\,dy$}>
```

$\iint xy\,dx\,dy$

Be careful not to use the same filename for two different pieces of code as this can lead to unexpected results.

`file/tex extension=`⟨*extension*⟩ (initially `.tex`)

`file/dim extension=`⟨*extension*⟩ (initially `.dim`)

`file/out extension=`⟨*extension*⟩ (initially `.pdf`)

These options are used by the default values of `file/tex file` → P. 20, `file/dim file` → P. 20, and `file/out file` → P. 20 to specify the filename extensions of the intermediate LaTeX, dimension, and compiled files, respectively.

These options are rarely needed.

An example of using them is the following.

```
\DeclareGraphicsRule{.mypdf}{pdf}{.mypdf}{}%
<\localpreamble[
   file/tex extension=.mytex,
   file/dim extension=.mydim,
   file/out extension=.mypdf,
   file=locpream-locpream-extension,
   before read={\ShellEscape{mv locpream-locpream-extension.pdf
                                locpream-locpream-extension.mypdf}}]
   {ABC}>
```

`file/tex file`=⟨*filename*⟩ (initially `\locpream@file\csname locpream@file/tex extension\endcsname`)

`file/dim file`=⟨*filename*⟩ (initially `\locpream@file\csname locpream@file/dim extension\endcsname`)

`file/out file`=⟨*filename*⟩ (initially `\locpream@file\csname locpream@file/out extension\endcsname`)

These options specify the filenames of the intermediate LaTeX, dimension, and compiled files, respectively. The default values for these concatenate `file`→P. 19 with `file/tex extension`→P. 19, `file/dim extension`→P. 19, and `file/out extension`→P. 19, respectively.

These options are rarely needed. For most uses, it is better to use `file`→P. 19.

An example of using these options is the following.

```
\DeclareGraphicsRule{.mypdf}{pdf}{.mypdf}{}%
<\localpreamble[
   file/tex file=locpream-locpream-full-file.mytex,
   file/dim file=locpream-locpream-full-file.mydim,
   file/out file=locpream-locpream-full-file.mypdf,
   before read={\ShellEscape{mv locpream-locpream-full-file.pdf
                                locpream-locpream-full-file.mypdf}}]
   {XYZ}>
```

Be careful not to use the same filename for two different pieces of code as this can lead to unexpected results.

`includegraphics/options`=⟨*key-value sequence*⟩ (initially empty)

This option specifies options to be passed to the `\includegraphics` command that is used to read into the master document the result of compiling the LaTeX code. For example, the following uses `angle` to rotate the image read by `\includegraphics`.

```
<\localpreamble[includegraphics/options={angle=45}]{M}>
```

5 Issues and Workarounds

5.1 Document Not Updating

A common issue is changing the LaTeX code in a `\localpreamble` → P. 7 or `localpreambleenv` → P. 7 but those changes not being reflected in the master document after re-compiling the master document. The cause of this is that `locpream` has no way to detect whether compiling the intermediate LaTeX file succeeded or failed. If compilation of the intermediate LaTeX file fails, the dimension file and the resulting PDF file from a previous compilation will not be overwritten.

To fix this, delete the PDF and dimension files. Then failure of the compilation of the intermediate LaTeX file will cause a file-not-found error when the PDF and dimension file are read. You can then use this file-not-found error to let you know when you have fixed that LaTeX code.

5.2 Hash Symbols and Command Arguments

Using hash symbols, such as when referencing a command argument (e.g., `#1`), can lead to problems. For example, the following would lead to an error.

```
<\localpreamble[preamble={\newcommand{\p}[1]{(#1)}}]{\p{x}}>
```

The solution to this is to use double hashes (e.g, `##1` instead of `#1`) as demonstrated in the following.

```
<\localpreamble[preamble={\newcommand{\p}[1]{(##1)}}]{\p{x}}>
```

```
<\begin{localpreambleenv}[preamble={\newcommand{\p}[1]{(##1)}}]
   \p{x}
 \end{localpreambleenv}>
```

This even applies in the body of the command `\localpreamble` → P. 7 and the environment `localpreambleenv` → P. 7 as seen in the following.

```
<\localpreamble{\newcommand{\p}[1]{(##1)}\p{x}}>
```

(x)

```
<\begin{localpreambleenv}
   \newcommand{\p}[1]{(##1)}%
   \p{x}
 \end{localpreambleenv}>
```

(x)

This also applies to standalone files as seen in the following.

locpream-standalone-hash.tex

```
\RequirePackage{locpream.code}
\LocalPreambleCode
  {locpream-standalone-hash.dim} % dimension file
  {\documentclass{article}\newcommand{\p}[1]{(##1)}} % preamble
  {} % before savebox
  {\p{x}} % body
  {} % after savebox
  {} % before usebox
  {} % after usebox
```

```
<\LocalPreambleCompile[file=locpream-standalone-hash]>
<\LocalPreambleRead[file=locpream-standalone-hash]>
```

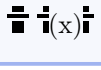
(x)

Finally, `\newlocalpreamble` → P. 8 and `\newlocalpreambleenv` → P. 8 require *four* hashes due to an extra level of indirection occurring in them as demonstrated in the following examples.

```
<\newlocalpreamble[preamble={\newcommand{\p}[1]{(####1)}}]{\paren}>
<\paren{\p{x}}>
```

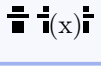
(x)

```
<\newlocalpreambleenv[preamble={\newcommand{\p}[1]{(####1)}}]{paren}>
<\begin{paren}\p{x}\end{paren}>
```

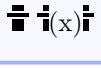


However, in their bodies this does not apply, and only two hashes should be used as demonstrated in the following.

```
<\newlocalpreamble{\paren}>
<\paren{\newcommand{\parens}[1]{(##1)}\parens{x}}>
```



```
<\newlocalpreambleenv{paren}>
<\begin{paren}
   \newcommand{\parens}[1]{(##1)}%
   \parens{x}
 \end{paren}>
```



5.3 Category Codes

Some commands change the category codes of characters. These pose a problem for use with commands from the `locpream` package as the arguments to commands are parsed before those category codes have changed. The way to work around this is to use the `\scantokens` macro to cause parts of those arguments to be re-parsed.

For example, the `\DeclareFontShape` macro redefines category codes for characters used in its argument. Thus, to use it one must insert a call to `\scantokens` as in the following.

```
<\localpreamble[preamble={
                  \usepackage{pifont}
                  \DeclareFontFamily{U}{msa}{}
                  \scantokens{
                    \DeclareFontShape
                      {U}{msa}{m}{n}
                      {<-6>msam5<6-8>msam7<8->msam10}{}\relax}}]
   {\Pisymbol{msa}{15}}>
```

Note that the `\relax` before the end of the argument to `\scantokens` ensures that `\scantokens` does not insert an extra space at the end. See `https:`

//tex.stackexchange.com/questions/117906/use-of-everyeof-and-endlinechar-with-scantokens for details.

This trick also works when using a standalone file as in the following.

locpream-standalone-catcode.tex

```
\RequirePackage{locpream.code}
\LocalPreambleCode
  {locpream-standalone-catcode.dim} % dimension file
  % preamble
  {\documentclass{article}
   \usepackage{pifont}
   \DeclareFontFamily{U}{msa}{}
   \scantokens{
     \DeclareFontShape
       {U}{msa}{m}{n}
       {<-6>msam5<6-8>msam7<8->msam10}{}\relax}}
  {} % before savebox
  {\Pisymbol{msa}{15}} % body
  {} % after savebox
  {} % before usebox
  {} % after usebox
```

```
<\LocalPreambleCompile[file=locpream-standalone-catcode]>
<\LocalPreambleRead[file=locpream-standalone-catcode]>
```

6 Changelog

All notable changes to this project will be documented here.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

0.1.0 Michael D. Adams (2019-01-05)

- Initial version

Index