

# The `minted` package: Highlighted source code in $\LaTeX$

Geoffrey M. Poore  
`gpoore@gmail.com`  
`github.com/gpoore/minted`

Originally created and maintained (2009–2013) by  
Konrad Rudolph

v3.4.0 from 2024/11/17

## Abstract

`minted` provides syntax highlighting using the `Pygments` library. It also provides options for customizing the highlighted source code output, including features implemented in Python such as selecting snippets of code with regular expressions.

The original development of `minted` version 3 was funded by a `TEX Development Fund grant` from the `TEX Users Group` in 2023.

## License

`LaTeX Project Public License (LPPL)` version 1.3c.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Installation</b>	<b>4</b>
2.1	Package manager . . . . .	4
2.2	Manual installation . . . . .	5
2.2.1	Option 1 (recommended): Install latexminted within Python installation . . . . .	5
2.2.2	Option 2: Install latexminted within T <sub>E</sub> X installation . . . . .	5
<b>3</b>	<b>Migrating from minted version 2</b>	<b>6</b>
<b>4</b>	<b>Basic usage</b>	<b>7</b>
4.1	The latexminted Python executable and shell escape . . . . .	7
4.2	A minimal complete example . . . . .	8
4.3	Formatting source code . . . . .	9
4.4	Using different styles . . . . .	10
4.5	Supported languages . . . . .	10
<b>5</b>	<b>Floating listings</b>	<b>10</b>
<b>6</b>	<b>Configuration</b>	<b>12</b>
6.1	minted config file .latexminted_config . . . . .	12
6.2	macOS compatibility . . . . .	14
<b>7</b>	<b>Options</b>	<b>14</b>
7.1	Package options . . . . .	14
7.2	Setting options for commands and environments . . . . .	17
7.3	Command and environment options . . . . .	18
<b>8</b>	<b>Defining shortcuts</b>	<b>32</b>
<b>9</b>	<b>FAQ and Troubleshooting</b>	<b>34</b>
<b>10</b>	<b>Acknowledgements</b>	<b>37</b>
<b>11</b>	<b>Implementation</b>	<b>37</b>
11.1	Required packages . . . . .	37
11.2	Exception handling . . . . .	37
11.3	Python executable and minimum supported version . . . . .	38
11.4	Timestamp . . . . .	38
11.5	Jobname MD5 and derived file names . . . . .	39
11.6	Package options . . . . .	41
11.6.1	Package option definitions . . . . .	41
11.6.2	Package options that are no longer supported or deprecated . . . . .	44
11.6.3	Package option processing . . . . .	45
11.7	Util . . . . .	45
11.7.1	Check whether a string matches the regex $^{\wedge}[0-9A-Za-z_-]+\$$ . . . . .	46
11.8	State . . . . .	47
11.9	Calling minted executable . . . . .	47
11.10	Config detection . . . . .	49

11.11	Options . . . . .	52
11.11.1	Option processing . . . . .	52
11.11.2	Option handlers . . . . .	59
11.11.3	Option definitions . . . . .	60
11.12	Caching, styles, and highlighting . . . . .	63
11.12.1	State . . . . .	63
11.12.2	Cache management . . . . .	64
11.12.3	Style definitions . . . . .	66
11.12.4	Lexer-specific line numbering . . . . .	70
11.12.5	Highlighting code . . . . .	71
11.13	Public API . . . . .	76
11.14	Command shortcuts . . . . .	81
11.15	Float support . . . . .	83

# 1 Introduction

minted provides syntax highlighting using the `Pygments` library. The general strategy is to wrap code in a command or environment that captures it verbatim, like this:

```
\begin{minted}<language>
<code>
\end{minted}
```

Then the code is passed to Python, highlighted with Pygments, and passed back to  $\LaTeX$  for inclusion in the document. Here is an example with Ruby code, showing the  $\LaTeX$  source and then the highlighted output:

<pre>\begin{minted}{ruby} class Foo   def init     pi = Math::PI     @var = "Pi = #{pi}..."   end end \end{minted}</pre>	<pre>class Foo   def init     pi = Math::PI     @var = "Pi = #{pi}..."   end end</pre>
--	--

Because minted uses Pygments and other Python software, it can provide more highlighting features than are practical in syntax highlighting packages like `listings` that are implemented purely in  $\LaTeX$ . In the past, this reliance on external software brought several disadvantages, including a requirement for separately installing Pygments. As of minted version 3, all Python software including Pygments is bundled with the  $\LaTeX$  package when it is installed with a  $\TeX$  package manager, and no dependencies must be installed separately.

## 2 Installation

### 2.1 Package manager

Installation will typically be simpler and faster using your  $\TeX$  distribution's package manager. Start your package manager's graphical user interface, or use the relevant command below:

- TeX Live: `tlmgr install minted`
- MiKTeX: `mpm --admin --install=minted`

When the minted package is installed, it includes the `latexminted` Python executable and all required Python libraries including Pygments. For these to function correctly, Python 3.8+ must be installed and on `PATH` when the `latexminted` executable runs.

Note that if you plan to use Pygments plugin packages, you will need to install the `latexminted` Python package and dependencies including Pygments within a Python installation. The Python libraries installed by a  $\TeX$  package manager within a  $\TeX$  installation are not compatible with plugin packages. After installing `latexminted` within a Python installation, make sure that its `latexminted` executable has precedence on `PATH`.

The minted package has the  $\LaTeX$  package dependencies listed below. Depending on your  $\TeX$  distribution and configuration, these may be installed automatically when minted is installed.

- catchfile
- etoolbox
- float
- fvextra
- latex2pydata
- newfloat
- pdftexcmds
- pgfkeys
- pgfopts
- shellesc
- xcolor

## 2.2 Manual installation

minted source files are available at [github.com/gpoore/minted](https://github.com/gpoore/minted). There is also [ctan.org/pkg/minted](https://ctan.org/pkg/minted).

Install `minted.sty` (and `minted2.sty` and `minted1.sty` if desired) within your  $\TeX$  installation. For TeX Live, it may be best to put style files under `TEXMFLOCAL`, which can be located by running `kpsewhich --var-value TEXMFLOCAL`. For example, you might put the files in `<texlive>/<year>/texmf-local/tex/latex/local/minted`. For further details, consult your  $\TeX$  distribution's documentation, or an online guide such as [en.wikibooks.org/wiki/LaTeX/Installing\\_Extra\\_Packages](https://en.wikibooks.org/wiki/LaTeX/Installing_Extra_Packages) or [texfaq.org](https://texfaq.org). After installing the `.sty` files, make  $\TeX$  aware of the new files by running `texhash` or `mktexlsr` (TeX Live), or `initexmf --update-fndb` (MiKTeX).

Next, install the Python side of the package. Python 3.8+ is required. There are two options: Install the `latexminted` package and dependencies within a Python installation (typically easier, and required for compatibility with Pygments plugin packages), or install them within your  $\TeX$  installation.

Note that if you are only using the `minted2` package for backward compatibility with `minted` version 2, you do not need `latexminted`. `minted2` only requires the `Pygments` package, which can be installed with something like `pip install pygments`, `conda install anaconda::pygments`, or `brew install pygments`, depending on your operating system and Python distribution. You may need to modify the command depending on system versus user installation and depending on virtual environments.

### 2.2.1 Option 1 (recommended): Install latexminted within Python installation

If your Python distribution is compatible with [The Python Package Index \(PyPI\)](https://pypi.org), this can be accomplished by running `pip install latexminted`. This will install `latexminted` plus all dependencies including `Pygments`. You may need to modify the command depending on whether you want a system or user (`--user`) installation, depending on whether you are using virtual environments, and depending on whether something like `pip3` is needed instead of `pip`.

If you cannot or do not wish to use PyPI via `pip`, install the following packages manually or from other sources.

- `latexminted`: <https://pypi.org/project/latexminted/>
- `latexrestricted`: <https://pypi.org/project/latexrestricted/>
- `latex2pydata`: <https://pypi.org/project/latex2pydata/>
- `Pygments`: <https://pypi.org/project/Pygments/>

### 2.2.2 Option 2: Install latexminted within $\TeX$ installation

This approach is more involved and essentially replicates the process that is performed automatically when using a  $\TeX$  package manager.

Install the `latexminted.py` executable within your  $\TeX$  installation. (It is part of the minted  $\LaTeX$  package, separate from the latexminted Python package.) This should typically be within a `scripts` directory. When TeX Live installs minted with its package manager, this is something like `<texlive>/<year>/texmf-dist/scripts/minted`.

Download Python wheels (`*.whl`) for the following Python packages, and place them in the same location as `latexminted.py`.

- latexminted: <https://pypi.org/project/latexminted/>
- latexrestricted: <https://pypi.org/project/latexrestricted/>
- latex2pydata: <https://pypi.org/project/latex2pydata/>
- Pygments: <https://pypi.org/project/Pygments/>

Under non-Windows operating systems, create a symlink called `latexminted` in the  $\TeX$  binary directory or another appropriate location that points to `latexminted.py`. When TeX Live installs minted with its package manager, this is something like `<texlive>/<year>/bin/<architecture>`.

Under Windows, a launcher executable for `latexminted.py` needs to be created. When TeX Live installs minted with its package manager, it creates a copy of `runscript.exe` named `latexminted.exe` within the  $\TeX$  binary directory, which is something like `<texlive>/<year>/bin/windows`.

### 3 Migrating from minted version 2

minted version 3 is a complete rewrite from version 2.9. A brief summary of changes is provided below. For full details, see `CHANGELOG_MINTED_LATEX_PACKAGE.md`.

#### Backward compatibility

The new `minted2` package provides the features of minted version 2.9, the final release before version 3. No additional version 2 releases are planned; no changes to the `minted2` package are expected.

#### New features and changes

- Version 3 uses a new minted-specific Python executable called `latexminted` to perform syntax highlighting. This executable is specifically designed to meet the security requirements for restricted shell escape programs. Once it has passed a security review and is accepted by  $\TeX$  distributions, it will be possible to highlight code without `-shell-escape` and its attendant security vulnerabilities.

Syntax highlighting is still performed with Pygments, but the `pygmentize` executable included with Pygments is no longer used.

When minted is installed with a  $\TeX$  package manager, the new `latexminted` executable and all Python libraries including Pygments are installed within the  $\TeX$  installation. A separate step to install Pygments is no longer necessary.

- Temporary files are no longer created unless code needs to be highlighted. There is a new naming scheme for temporary files and for cache files.
- New package options: `debug` (additional debug info during compilation), `highlightmode` (modify when code is highlighted for faster compilation),

`placeholder` (insert a placeholder instead of code), and `verbatim` (insert verbatim approximation of code).

- Renamed package options `langlinenos` to `lexerlinenos` and `inputlanglinenos` to `inputlexerlinenos`. The old names are still supported.
- `bgcolor` now uses the new `bgcolor` option from `fvextra` v1.8. Because `bgcolor` now introduces no additional whitespace or padding, existing documents may require some modification. Added new option `bgcolorpadding` for modifying padding in background color regions. Added new option `bgcolorvphantom` for setting height of background color in inline contexts. When more sophisticated background colors are needed, `tcolorbox` or a similar package should be used.
- The default cache directory name is now `_minted`. All files within a directory now share the same cache, instead of having separate per-document caches. Document-specific caching as in `minted` version 2 can be restored using the package option `cachedir`.
- `\newminted` now creates an environment that takes an optional argument consisting of options, instead of taking no argument.
- File encoding changes: The new `latexminted` executable assumes that  $\LaTeX$  output files are UTF-8, and saves highlighted code as UTF-8. That is,  $\LaTeX$  should be configured so that everything is UTF-8. The encoding option now defaults to UTF-8. It is only used in decoding files for `\inputminted` and commands based on it. The `outencoding` option is no longer supported.
- Added new options for including ranges of code based on literal string delimiters or regular expressions: `rangestartstring`, `rangestartafterstring`, `rangestopstring`, `rangestopbeforestring`, `rangeregex`.
- There is now support for custom lexers in standalone Python files. See the documentation for the new `.latexminted_config` configuration files for details.
- Several package options are no longer supported and result in errors or warnings. The package options `finalizcache`, `outputdir`, and `kpsewhich` are no longer needed given new `minted` version 3 capabilities. The package options `draft` and `final` no longer have any effect and will soon be removed altogether. The new package options `placeholder` and `verbatim` are available in cases where using highlighted code should be completely avoided.

## 4 Basic usage

### 4.1 The `latexminted` Python executable and shell escape

The `minted` package operates by passing code to the `latexminted` Python executable, which performs syntax highlighting and then returns the highlighted code in  $\LaTeX$  format.

`latexminted` is designed to be compatible with the security requirements for restricted shell escape. For up-to-date installations of TeX Live 2024+, the `-shell-escape` option is no longer required. The `latexminted` Python executable has been added to TeX Live's list of trusted executables.

For versions of TeX Live before 2024 and for MiKTeX, `latexminted` requires special permission to run. This can be accomplished by running  $\TeX$  with the `-shell-escape` option (TeX Live) or the `-enable-write18` option (MiKTeX). Note that using `-shell-escape` or `-enable-write18` allows  $\TeX$  to run potentially arbitrary commands on your system. These should only be used when necessary, with documents from trusted sources. An alternative is to manually designate `latexminted` as a trusted executable.

- TeX Live: Copy the variable `shell_escape_commands` from the distribution `texmf.cnf` (something like `<texlive>/<yr>/texmf-dist/web2c/texmf.cnf`) into the user `texmf.cnf` (something like `<texlive>/<yr>/texmf.cnf`), and then add `latexminted` to the `shell_escape_commands` list. The location of the `texmf.cnf` files can be determined by running `kpsewhich -all texmf.cnf`. Note that under Windows, this only works when `latexminted` is installed within a TeX Live installation; it is not compatible with `latexminted` being installed in a Python installation.
- MiKTeX: Add a line `AllowedShellCommands [] = latexminted` to the existing list of allowed commands in `miktex.ini`. You may want to modify the user-scoped configuration instead of the system-wide configuration. See the [MiKTeX documentation](#) for more details, particularly `initexmf --edit-config-file` and `initexmf --set-config-value`.

For the `latexminted` Python executable to correctly inherit security settings from  $\TeX$ , there are requirements for system configuration when multiple  $\TeX$  installations are present.

- With MiKTeX on systems with multiple MiKTeX installations, the desired MiKTeX installation must be the first MiKTeX installation on `PATH`.
- With TeX Live on Windows systems with multiple TeX Live installations, the desired TeX Live installation must be the first TeX Live installation on `PATH`.

See the [`latexrestricted`](#) documentation for details.

## 4.2 A minimal complete example

The following file `minimal.tex` shows the basic usage of `minted`.

```
\documentclass{article}

\usepackage{minted}
\usepackage[svgnames]{xcolor}

\begin{document}
\begin{minted}[bgcolor=Beige, bgcolorpadding=0.5em]{c}
int main() {
    printf("hello, world");
    return 0;
}
\end{minted}
\end{document}
```



This document can be compiled by running “`pdflatex -shell-escape minimal`” to produce the following output in `minimal.pdf`:

```
int main() {
    printf("hello, world");
    return 0;
}
```

### 4.3 Formatting source code

`minted` The `minted` environment highlights a block of code:

<pre>\begin{minted}{python} def boring(args = None):     pass \end{minted}</pre>	<pre>def boring(args = None):     pass</pre>
--	--

The environment accepts a number of optional arguments in key=value notation. These are described in section 7.2.

To use `minted` with a language that is not supported by Pygments, or simply to disable highlighting, set the language to `text`: `\begin{minted}{text}`.

`\mint` For a single line of source code, you can use `\mint` as a shorthand for `minted`:

<pre>\mint{python}/import this/</pre>	<pre>import this</pre>
---------------------------------------	------------------------

This typesets a single line of code using a command rather than an environment, so it saves a little typing, but its output is equivalent to that of the `minted` environment.

The code is delimited by a pair of identical characters, similar to how `\verb` works. The complete syntax is `\mint [options] {language}delimcodedelim`, where the code delimiter can be almost any punctuation character. The *code* may also be delimited with paired curly braces `{}`, so long as *code* itself does not contain unpaired curly braces.

Note that the `\mint` command **is not for inline use**. Rather, it is a shortcut for `minted` when only a single line of code is present. The `\mintinline` command is provided for inline use.

`\mintinline` Code can be typeset inline:

<pre>\mintinline{py}{print("Hello!")}</pre>	<pre>print("Hello!")</pre>
---	----------------------------

The syntax is `\mintinline [options] {language}delimcodedelim`. The delimiters can be a single repeated character, just like for `\verb`. They can also be a pair of curly braces, `{}`. Curly braces are required when `\mintinline` is used in a movable argument, such as in a `\section`.

Unlike `\verb`, `\mintinline` can usually be used inside other commands. The main exception is when the code contains the percent `%` or hash `#` characters, or unpaired curly braces. For example, `\mintinline` typically works in `\footnote` and `\section!` Note that some document classes or packages, such as `memoir`, redefine `\section` or have options that modify it in ways that are incompatible with `\mintinline`. If you use `\mintinline` inside `\section` or otherwise in a movable argument, you should

experiment to make sure it is compatible with your document configuration. You may also want to consider `fvextra's \Verb` or `\EscVerb` as an alternative.

The code typesetting for `\mintinline` is based on `fvextra's \Verb`. See the [fvextra documentation on \Verb](#) for additional details about functionality and limitations.

`\inputminted` Finally, there's the `\inputminted` command to input external files. Its syntax is `\inputminted[<options>]{<language>}{<filename>}`.

#### 4.4 Using different styles

`\usemintedstyle`  
`\setminted`

Instead of using the default highlighting style you may choose another style provided by Pygments. There are two equivalent ways to do this:

```
\usemintedstyle{name}  
\setminted{style=name}
```

The `\setminted` approach has the advantage that other minted options are accepted as well; `\usemintedstyle` is restricted to style modifications. The full syntax is `\usemintedstyle[<language>]{<style>}` and `\setminted[<language>]{<key=value>}`. The style may be set for the document as a whole (no language specified), or only for a particular language. Note that the style may also be set via the optional argument for each command and environment.

Highlighting styles with examples are at [pygments.org/styles](http://pygments.org/styles). It is possible to preview your code with different styles using the online demo at [pygments.org/demo](http://pygments.org/demo). Available styles can also be listed by running the command `pygmentize -L styles`.

It is also possible to create your own styles. See the instructions on the [Pygments website](#). `minted` only supports style names that match the regular expression `^[0-9A-Za-z_-]+$`.

#### 4.5 Supported languages

Pygments supports hundreds of different programming languages, template languages, and other markup languages. The list of currently supported languages is at [pygments.org/docs/lexers/](http://pygments.org/docs/lexers/). You can also run `pygmentize -L lexers`.

### 5 Floating listings

`listing` `minted` provides a `listing` environment that can be used to wrap code blocks. This puts the code in a floating box similar to a figure or table, with default placement `tbp`. You can also provide a `\caption` and a `\label`:

```

\begin{listing}[H]
\mint{cl}/(car (cons 1 '(2)))/
\caption{Example of a listing.}
\label{lst:example}
\end{listing}

```

Listing `\ref{lst:example}` contains an example of a listing.

---

```
(car (cons 1 '(2)))
```

Listing 1: Example of a listing.

Listing 1 contains an example of a listing.

The default `listing` placement can be modified easily. When the package option `newfloat=false` (default), the `float` package is used to create the `listing` environment. Placement can be modified by redefining `\fps@listing`. For example,

```

\makeatletter
\renewcommand{\fps@listing}{htp}
\makeatother

```

When `newfloat=true`, the more powerful `newfloat` package is used to create the `listing` environment. In that case, `newfloat` commands are available to customize `listing`:

```
\SetupFloatingEnvironment{listing}{placement=htp}
```

`\listoflistings` The `\listoflistings` macro will insert a list of all (floated) listings in the document:

<code>\listoflistings</code>	<b>List of Listings</b>
	1 Example of a listing. . . 11

## Customizing the listing environment

By default, the `listing` environment is created using the `float` package. In that case, the `\listingscaption` and `\listoflistingscaption` macros described below may be used to customize the caption and list of listings. If `minted` is loaded with the `newfloat` option, then the `listing` environment will be created with the more powerful `newfloat` package instead. `newfloat` is part of `caption`, which provides many options for customizing captions.

When `newfloat` is used to create the `listing` environment, customization should be achieved using `newfloat`'s `\SetupFloatingEnvironment` command. For example, the string "Listing" in the caption could be changed to "Program code" using

```
\SetupFloatingEnvironment{listing}{name=Program code}
```

And “List of Listings” could be changed to “List of Program Code” with

```
\SetupFloatingEnvironment{listing}{listname=List of Program Code}
```

Refer to the newfloat and caption documentation for additional information.

`\listingscaption` This allows the string “Listing” in a listing’s caption to be customized. It only applies when package option `newfloat=false`. For example:

```
\renewcommand{\listingscaption}{Program code}
```

`\listoflistingscaption` This allows the caption of the listings list, “List of Listings,” to be customized. It only applies when package option `newfloat=false`. For example:

```
\renewcommand{\listoflistingscaption}{List of Program Code}
```

## 6 Configuration

### 6.1 minted config file `.latexminted_config`

Several minted settings with security implications can be customized with a config file `.latexminted_config`. This config file is loaded by the `latexminted` Python executable when it runs.

The `latexminted` Python executable looks for `.latexminted_config` files in the following locations:

- User home directory, as found by Python’s `pathlib.Path.home()`.
- `TEXMFHOME`. With MiKTeX on systems with multiple MiKTeX installations, this will be the `TEXMFHOME` from the first MiKTeX installation on `PATH`. With TeX Live on Windows systems with multiple TeX Live installations, this will be the `TEXMFHOME` from the first TeX Live installation on `PATH`. In all other cases, `TEXMFHOME` will correspond to the currently active TeX installation. See the `latexrestricted` documentation for details. `latexrestricted` is used by the `latexminted` Python executable to retrieve the value of `TEXMFHOME`.
- The current TeX working directory. Note that `enable_cwd_config` must be set `true` in the `.latexminted_config` in the user home directory or in the `TEXMFHOME` directory to enable this; `.latexminted_config` in the current TeX working directory is not enabled by default for security reasons. Even when a config file in the current TeX working directory is enabled, it cannot be used to modify certain security-related settings.

Overall configuration is derived by merging all config files, with later files in the list above having precedence over earlier files. Boolean and string values are overwritten by later config files. Collection values (currently only sets derived from lists) are merged with earlier values.

The `.latexminted_config` file may be in Python literal format (dicts and lists of strings and bools), JSON, or TOML (requires Python 3.11+). It must be encoded as UTF-8.

## Config settings

**security:** `dict[str, str | bool]` These settings relate to latexminted security. They can only be set in `.latexminted_config` in the user home directory or in `TEXMFHOME`. They cannot be set in `.latexminted_config` in the current  $\TeX$  working directory.

**enable\_cwd\_config:** `bool = False` Load a `.latexminted_config` file from the current  $\TeX$  working directory if it exists. This is disabled by default because the config file can enable `custom_lexers`, which is equivalent to arbitrary code execution.

**file\_path\_analysis:** `"resolve" | "string" = "resolve"` This specifies how latexminted determines whether files are readable and writable. Relative file paths are always treated as being relative to the current  $\TeX$  working directory.

With `resolve`, any symlinks in file paths are resolved with the file system before paths are compared with permitted  $\LaTeX$  read/write locations. Arbitrary relative paths including `..` are allowed so long as the final location is permitted.

With `string`, paths are analyzed as strings in comparing them with permitted  $\LaTeX$  read/write locations. This follows the approach taken in  $\TeX$ 's file system security. Paths cannot contain `..` to access a parent directory, even if the parent directory is a valid location. Because symlinks are not resolved with the file system, it is possible to access locations outside permitted  $\LaTeX$  read/write locations, if the permitted locations contain symlinks to elsewhere.

**permitted\_pathext\_file\_extensions:** `list[str]` As a security measure under Windows,  $\LaTeX$  cannot write files with file extensions in `PATHEXT`, such as `.bat` and `.exe`. This setting allows latexminted to write files with the specified file extensions, overriding  $\LaTeX$  security. File extensions should be in the form `".<ext>"`, for example, `".bat"`. This setting is used in extracting source code from  $\LaTeX$  documents and saving it in standalone source files.

When these file extensions are enabled for writing, as a security measure latexminted will only allow them to be created in **subdirectories** of the current  $\TeX$  working directory, `TEXMFOUTPUT`, and `TEXMF_OUTPUT_DIRECTORY`. These files cannot be created directly under the  $\TeX$  working directory, `TEXMFOUTPUT`, and `TEXMF_OUTPUT_DIRECTORY` because those locations are more likely to be used as a working directory in a shell, and thus writing executable files in those locations would increase the risk of accidental code execution.

**custom\_lexers:** `dict[str, str | list[str]]` This is a mapping of custom lexer file names to SHA256 hashes. Only custom lexers with these file names and the corresponding hashes are permitted. Lists of hashes are allowed to permit multiple versions of a lexer with a given file name. All other custom lexers are prohibited, because loading custom lexers is equivalent to arbitrary code execution. For example:

```
"custom_lexers": {
```

```
"mylexer.py": "<sha256>"
}
```

By default, it is assumed that custom lexer files implement a class `CustomLexer`. This can be modified by including the lexer class name with the file name, separated by a colon, when the lexer is used. For example:

```
\inputminted{./<path>/mylexer.py:LexerClass}{<file>}
```

Note that `custom_lexers` only applies to custom lexers in standalone Python files. Lexers that are installed within Python as plugin packages work automatically with Pygments and do not need to be enabled separately. However, in that case it is necessary to install `latexminted` and Pygments within a Python installation. When  $\TeX$  package managers install `latexminted` and Pygments within a  $\TeX$  installation, these are not compatible with Pygments plugin packages.

## 6.2 macOS compatibility

If you are using `minted` with some versions/configurations of macOS, and are using caching with a large number of code blocks (> 256), you may receive a Python error during syntax highlighting that looks like this:

```
OSError: [Errno 24] Too many open files:
```

This is due to the way files are handled by the operating system, combined with the way that caching works. To resolve this, you may use one of the following commands to increase the number of files that may be used:

- `launchctl limit maxfiles`
- `ulimit -n`

## 7 Options

### 7.1 Package options

`chapter` To control how  $\LaTeX$  counts the listing floats, you can pass either the `chapter` or `section` option when loading the `minted` package. For example, the following will cause listings to be counted by chapter:

```
\usepackage[chapter]{minted}
```

`cache=(boolean)` `(default: true)` `minted` works by saving code to a temporary file, highlighting it with Pygments, and then passing the result back to  $\LaTeX$  for inclusion in the document. This process can become quite slow if there are several chunks of code to highlight. To avoid this, the package provides a `cache` option. This is on by default.

The `cache` option creates a directory `_minted` in the document's root directory (this may be customized with the `cachedir` option). Files of highlighted code are stored in this directory, so that the code will not have to be highlighted again in the future. Cache files that are no longer used are automatically deleted. In most cases, caching will significantly speed up document compilation.

`cachedir=(directory)` This allows the directory in which cache files are stored to be customized. Paths should use forward slashes, even under Windows. Special characters must be escaped with `\string` or `\detokenize`.  
(default: `_minted`)

Note that the cache directory is relative to `-output-directory` or equivalently the `TEXMF_OUTPUT_DIRECTORY` environment variable, if that is set.

`debug=(boolean)` Provide additional information for aid in debugging. This keeps temp files that are used in generating highlighted code and also writes additional information to the log.  
(default: `false`)

`frozenscache=(boolean)` Use a frozen (static) cache. When `frozenscache=true`, Python and Pygments are not required, and any external files accessed through `\inputminted` are no longer necessary. If a cache file is missing, an error will be reported and there will be no attempt to generate the missing cache file.  
(default: `false`)

When using `frozenscache` with `-output-directory`, the `cachedir` package option should be used to specify a full relative path to the cache (for example, `cachedir=./<output_directory>/_minted`).

`highlightmode=(string)` Determines when code is highlighted. This only has an effect when `cache=true`.  
(default: `fastfirst`)

The default is `fastfirst`. If a cache for the document exists, then code is highlighted immediately. If a cache for the document does not exist, then typeset a placeholder instead of code and highlight all code at the end of the document. This will require a second compile before code is typeset, but because all code is highlighted at once, there is less overhead and the total time required can be significantly less for documents that include many code snippets.

The alternatives are `fast` (always highlight at end of document, requiring a second compile) and `immediate` (always highlight immediately, so no second compile is needed).

Temporary files with the following file extensions are automatically detected and processed correctly, regardless of `highlightmode`: `.listing`, `.out`, `.outfile`, `.output`, `.temp`, `.tempfile`, `.tmp`, `.verb`, and `.vrb`. For temp files with other file extensions, `highlightmode=immediate` is needed if the files are overwritten or deleted during compilation. `fastfirst` can work in such cases, but it will give an error message about modified or missing files during the first compile, and then will work correctly during subsequent compiles when it switches to `immediate` mode.

When code is highlighted at the end of the document with `fast` or `fastfirst`, any error and warning messages will refer to a location at the end of the document rather than the original code location, since highlighting occurred at the end of the document. In this case, messages are supplemented with original  $\LaTeX$  source file names and line numbers to aid in debugging.

`inputlexerlinenos=(boolean)` This enables `lexerlinenos` and causes it to apply to `\inputminted` (and custom commands based on it) in addition to `minted` environments and `\mint` commands (and custom environments/commands based on them).  
(default: `false`)

The regular `lexerlinenos` option treats all code within a document's `.tex` files as having one set of line numbering per language, and then treats each inputted source file as having its own separate numbering. `inputlexerlinenos` defines a single numbering per lexer, regardless of where code originates.

`lexerlinenos=(boolean)` This allows all `minted` environments and `\mint` commands (and custom environments/commands based on them) for a given lexer (language) to share line numbering when `firstnumber=last`, so that each subsequent command/environment has line numbering that continues from the previous one. This does not apply to `\inputminted` (and custom commands based on it); see the package option `inputlexerlinenos` for that.  
(default: `false`)

minted uses the fancyvrb package behind the scenes for the code typesetting. fancyvrb provides an option `firstnumber` that allows the starting line number of an environment to be specified. For convenience, there is an option `firstnumber=last` that allows line numbering to pick up where it left off. The `lexerlinenos` option makes `firstnumber` work for each lexer (language) individually with all `minted` and `\mint` usages. For example, consider the code and output below.

```
\begin{minted}[linenos]{python}
def f(x):
    return x**2
\end{minted}

\begin{minted}[linenos]{ruby}
def func
    puts "message"
end
\end{minted}

\begin{minted}[linenos, firstnumber=last]{python}
def g(x):
    return 2*x
\end{minted}
```

---

```
1 def f(x):
2     return x**2

1 def func
2     puts "message"
3 end

3 def g(x):
4     return 2*x
```

Without the `lexerlinenos` option, the line numbering in the second Python environment would not pick up where the first Python environment left off. Rather, it would pick up with the Ruby line numbering.

- `newfloat=(boolean)`  
(default: false) By default, the `listing` environment is created using the `float` package. The `newfloat` option creates the environment using `newfloat` instead. This provides better integration with the `caption` package.
- `placeholder=(boolean)`  
(default: false) Instead of typesetting code, insert a placeholder. This is enabled automatically when working with PGF/TikZ externalization.
- `section` To control how  $\LaTeX$  counts the `listing` floats, you can pass either the `section` or `chapter` option when loading the `minted` package.
- `verbatim=(boolean)`  
(default: false) Instead of highlighting code, attempt to typeset it verbatim without using the `latexminted` Python executable. This is not guaranteed to be an accurate representation of the code, since some features such as `autogobble` require Python.



## 7.2 Setting options for commands and environments

All minted highlighting commands and environment accept the same set of options. Options are specified as a comma-separated list of key=value pairs. For example, we can specify that the lines should be numbered:

<pre>\begin{minted}[linenos=true]{c++} #include &lt;iostream&gt; int main() {     std::cout &lt;&lt; "Hello "                 &lt;&lt; "world"                 &lt;&lt; std::endl; } \end{minted}</pre>	<pre>1 #include &lt;iostream&gt; 2 int main() { 3     std::cout &lt;&lt; "Hello " 4               &lt;&lt; "world" 5               &lt;&lt; std::endl; 6 }</pre>
---	--

An option value of true may also be omitted entirely (including the “=”).  
`\mint` accepts the same options:

<pre>\mint[linenos]{perl} \$x~/foo/ </pre>	<pre>1 \$x~/foo/</pre>
--	------------------------

Here’s another example: we want to use the  $\LaTeX$  math mode inside comments:

<pre>\begin{minted}[mathescape]{py} # Returns <math>\sum_{i=1}^n i</math> def sum_from_one_to(n):     r = range(1, n + 1)     return sum(r) \end{minted}</pre>	<pre># Returns <math>\sum_{i=1}^n i</math> def sum_from_one_to(n):     r = range(1, n + 1)     return sum(r)</pre>
--	--

To make your  $\LaTeX$  code more readable you might want to indent the code inside a minted environment. The option `gobble` removes a specified number of characters from the output. There is also an `autogobble` option that automatically removes indentation (dedents code).

<pre>\begin{minted}[showspaces]{py}     def boring(args = None):         pass \end{minted}  versus  \begin{minted}[gobble=4, showspaces]{py}     def boring(args = None):         pass \end{minted}</pre>	<pre>def_boring(args=_None):     pass  versus  def_boring(args=_None):     pass</pre>
---	---

`\setminted` You may wish to set options for the document as a whole, or for an entire lexer (language). This is possible via `\setminted[<lexer>]{<key=value,...>}`. Lexer-specific options override document-wide options. Individual command and environment options override lexer-specific options.

`\setmintedinline` You may wish to set separate options for `\mintinline`, either for the document as a whole or for a specific lexer (language). This is possible via `\setmintedinline`. The syntax is `\setmintedinline[<lexer>]{<key=value,...>}`. Lexer-specific options override document-wide options. Individual command options override lexer-specific options. All settings specified with `\setmintedinline` override those set with `\setminted`. That is, inline settings always have a higher precedence than general settings.

### 7.3 Command and environment options

Following is a full list of available options. Several options are simply passed on to Pygments, fancyvrb, and fvextra for processing. In those cases, more details may be in the documentation for those software packages.

`autogobble` (boolean) (default: `false`)  
Remove (gobble) all common leading whitespace from code. Essentially a version of `gobble` that automatically determines what should be removed. Good for code that originally is not indented, but is manually indented after being pasted into a  $\LaTeX$  document.

<pre>...text. \begin{minted}[autogobble]{py}     def f(x):         return x**2 \end{minted}</pre>	<pre>...text. def f(x):     return x**2</pre>
---	---

When `autogobble` and `gobble` are used together, the effect is cumulative. First `autogobble` removes all common indentation, and then `gobble` is applied.

`autogobble` and `gobble` operate on code before the highlighting process begins (before lexing), treating the code purely as text. Meanwhile, `gobblefilter` operates on the token stream generated by a lexer. If the removed characters are simply indentation coming from how the code was entered within  $\LaTeX$ , then `autogobble` and `gobble` should typically be preferred. If the removed characters are syntactically significant, then `gobblefilter` may be better. Which approach is preferable may also depend on the implementation details of the lexer.

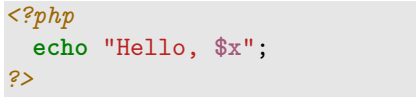
`baselinestretch` (dimension) (default: `<document default>`)  
Value to use for `baselinestretch` inside the listing.

`beameroverlays` (boolean) (default: `false`)  
Give the `<` and `>` characters their normal text meanings when used with `escapeinside` and `texcomments`, so that beamer overlays of the form `\only<1>{...}` will work.

`bgcolor` (string) (default: `none`)  
Background color behind commands and environments. This is only a basic, lightweight implementation of background colors using `\colorbox`. For more control of background colors, consider `tcolorbox` or a similar package, or a custom background color implementation.

`bgcolor` prevents line breaks for `\mintinline`. If you want to use `\setminted` to set background colors, and only want background colors on `minted` and `\mint`, you may use `\setmintedinline{bgcolor=none}` to turn off the coloring for inline commands.

The value of this option must *not* be a color command. Instead, it must be a color *name*, given as a string, of a previously-defined color:

<pre>\definecolor{bg}{rgb}{.9, .9, .9} \begin{minted}[bgcolor=bg]{php} &lt;?php     echo "Hello, \$x"; ?&gt; \end{minted}</pre>	
---	--

As an alternative to `bgcolor`, `tcolorbox` provides a built-in framing environment with `minted` support. Simply use `\tcbuselibrary{minted}` in the preamble, and then put code within a `tcblisting` environment:

```
\begin{tcblisting}{<tcb options>,
    minted language=<language>,
    minted style=<style>,
    minted options={<option list>} }
<code>
\end{tcblisting}
```

`tcolorbox` provides other commands and environments for fine-tuning listing appearance and for working with external code files.

- `bgcolorpadding` (length) (default: none)  
 Padding when `bgcolor` is set. For inline commands and for environments based on `BVerbatim`, this sets `\fboxsep` for the `\colorbox` that is used to create the background color. For environments based on `Verbatim`, `fancyvrb`'s frame options are used instead, particularly `framesep` and `fillcolor`. See the `fvextra` documentation for implementation details.
- `bgcolorvphantom` (macro) (default: `\vphantom{"Apgjy}`)  
`\vphantom` or similar macro such as `\strut` that is inserted at the beginning of each line of code using `bgcolor`. This allows the height of the background for each line of code to be customized. This is primarily useful for customizing the background for `\mintinline` and other inline code. It will typically have no effect on `minted` environments and other block code unless it is set to a size larger than `\strut`.
- `breakafter` (string) (default: `<none>`)  
 Break lines after specified characters, not just at spaces, when `breaklines=true`.  
 For example, `breakafter=-/` would allow breaks after any hyphens or slashes. Special characters given to `breakafter` should be backslash-escaped (usually `#`, `{`, `}`, `%`, `[`, `]`, and the comma `,`; the backslash `\` may be obtained via `\\`).  
 For an alternative, see `breakbefore`. When `breakbefore` and `breakafter` are used for the same character, `breakbeforeinrun` and `breakafterinrun` must both have the same setting.

```

\begin{minted}[breaklines, breakafter=d]{python}
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeverFitOnOneLine'
\end{minted}

-----

some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCould
↳ NeverFitOnOneLine'

```

`breakafterinrun` (boolean) (default: false)  
 When `breakafter` is used, insert breaks within runs of identical characters. If false, treat sequences of identical characters as a unit that cannot contain breaks. When `breakbefore` and `breakafter` are used for the same character, `breakbeforeinrun` and `breakafterinrun` must both have the same setting.

`breakaftersymbolpost` (string) (default: *none*)  
 The symbol inserted post-break for breaks inserted by `breakafter`.

`breakaftersymbolpre` (string) (default:  $\lfloor$ ,  $\rfloor$ )  
 The symbol inserted pre-break for breaks inserted by `breakafter`.

`breakanywhere` (boolean) (default: false)  
 Break lines anywhere, not just at spaces, when `breaklines=true`.

```

\begin{minted}[breaklines, breakanywhere]{python}
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeverFitOnOneLine'
\end{minted}

-----

some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNever
↳ FitOnOneLine'

```

`breakanywhereinlinestretch` (length) (default: *none*)  
 Stretch glue to insert at potential `breakanywhere` break locations in inline contexts, to give better line widths and avoid overfull `\hbox`. This allows the spacing between adjacent non-space characters to stretch, so it should not be used when column alignment is important. For typical line lengths, values between 0.01em and 0.02em should be sufficient to provide a cumulative stretch per line that is equal to or greater than the width of one character.

This is typically not needed in cases where an overfull `\hbox` only overflows by tiny amount, perhaps a fraction of a pt. In those cases, the overfull `\hbox` could be ignored, `\hfuzz` could be set to 1pt or 2pt to suppress tiny overfull `\hbox` warnings, or `breakanywheresymbolpre` might be redefined to adjust spacing.

`breakanywheresymbolpost` (string) (default: *none*)  
 The symbol inserted post-break for breaks inserted by `breakanywhere`.

`breakanywheresymbolpre` (string) (default:  $\lfloor$ ,  $\rfloor$ )  
 The symbol inserted pre-break for breaks inserted by `breakanywhere`.

`breakautoindent` (boolean) (default: `true`)  
When a line is broken, automatically indent the continuation lines to the indentation level of the first line. When `breakautoindent` and `breakindent` are used together, the indentations add. This indentation is combined with `breaksymbolindentleft` to give the total actual left indentation. Does not apply to `\mintinline`.

`breakbefore` (string) (default: `<none>`)  
Break lines before specified characters, not just at spaces, when `breaklines=true`.

For example, `breakbefore=A` would allow breaks before capital A's. Special characters given to `breakbefore` should be backslash-escaped (usually `#`, `{`, `}`, `%`, `[`, `]`, and the comma `,`; the backslash `\` may be obtained via `\\`).

For an alternative, see `breakafter`. When `breakbefore` and `breakafter` are used for the same character, `breakbeforeinrun` and `breakafterinrun` must both have the same setting.

```
\begin{minted}[breaklines, breakbefore=A]{python}
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeverFitOnOneLine'
\end{minted}

_____

some_string = 'SomeTextThatGoesOn,
- AndOnForSoLongThatItCouldNeverFitOnOneLine'
```

`breakbeforeinrun` (boolean) (default: `false`)  
When `breakbefore` is used, insert breaks within runs of identical characters. If `false`, treat sequences of identical characters as a unit that cannot contain breaks. When `breakbefore` and `breakafter` are used for the same character, `breakbeforeinrun` and `breakafterinrun` must both have the same setting.

`breakbeforesymbolpost` (string) (default: `<none>`)  
The symbol inserted post-break for breaks inserted by `breakbefore`.

`breakbeforesymbolpre` (string) (default: `\, \footnotesize\ensuremath{\_rflfloor}, \_j`)  
The symbol inserted pre-break for breaks inserted by `breakbefore`.

`breakbytoken` (boolean) (default: `false`)  
Only break lines at locations that are not within tokens; prevent tokens from being split by line breaks. By default, `breaklines` causes line breaking at the space nearest the margin. While this minimizes the number of line breaks that are necessary, it can be inconvenient if a break occurs in the middle of a string or similar token.

This does not allow line breaks between immediately adjacent tokens; for that, see `breakbytokenanywhere`.

A complete list of Pygments tokens is available at [pygments.org/docs/tokens](https://pygments.org/docs/tokens). If the breaks provided by `breakbytoken` occur in unexpected locations, it may indicate a bug or shortcoming in the Pygments lexer for the language.

`breakbytokenanywhere` (boolean) (default: `false`)  
Like `breakbytoken`, but also allows line breaks between immediately adjacent tokens, not just between tokens that are separated by spaces. Using `breakbytokenanywhere` with `breakanywhere` is redundant.

`breakindent` (dimension) (default: `(breakindentnchars)`)  
When a line is broken, indent the continuation lines by this amount.

When `breakautoindent` and `breakindent` are used together, the indentations add. This indentation is combined with `breaksymbolindentleft` to give the total actual left indentation.

Does not apply to `\mintinline`.

`breakindentnchars` (integer) (default: 0)  
This allows `breakindent` to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).

`breaklines` (boolean) (default: `false`)  
Automatically break long lines in minted environments and `\mint` commands, and wrap longer lines in `\mintinline`.

By default, automatic breaks occur at space characters. Use `breakanywhere` to enable breaking anywhere; use `breakbytoken`, `breakbytokenanywhere`, `breakbefore`, and `breakafter` for more fine-tuned breaking. Using `escapeinside` to escape to  $\LaTeX$  and then insert a manual break is also an option. For example, use `escapeinside=| |`, and then insert `|\ |` at the appropriate point. (Note that `escapeinside` does not work within strings.)

```
...text.  
\begin{minted}[breaklines]{py}  
def f(x):  
    return 'Some text ' + str(x)  
\end{minted}
```

```
...text.  
def f(x):  
    return 'Some text ' +  
        ↵ str(x)
```

Breaking in `minted` and `\mint` may be customized in several ways. To customize the indentation of broken lines, see `breakindent` and `breakautoindent`. To customize the line continuation symbols, use `breaksymbolleft` and `breaksymbolright`. To customize the separation between the continuation symbols and the code, use `breaksymbolsepleft` and `breaksymbolsepright`. To customize the extra indentation that is supplied to make room for the break symbols, use `breaksymbolindentleft` and `breaksymbolindentright`. Since only the left-hand symbol is used by default, it may also be modified using the alias options `breaksymbol`, `breaksymbolsep`, and `breaksymbolindent`. Note that none of these options applies to `\mintinline`, since they are not relevant in the inline context.

An example using these options to customize the `minted` environment is shown below. This uses the `\carriagereturn` symbol from the `dingbat` package.

```

\begin{minted}[breaklines,
                breakautoindent=false,
                breaksymbolleft=\raisebox{0.8ex}{
                    \small\reflectbox{\carriagereturn}},
                breaksymbolindentleft=0pt,
                breaksymbolsepleft=0pt,
                breaksymbolright=\small\carriagereturn,
                breaksymbolindentright=0pt,
                breaksymbolsepright=0pt]{python}

def f(x):
    return 'Some text ' + str(x) + ' some more text ' +
        str(x) + ' even more text that goes on for a while'
\end{minted}

```

---

```

def f(x):
    return 'Some text ' + str(x) + ' some more text ' +
    str(x) + ' even more text that goes on for a while'

```

Automatic line breaks are limited with Pygments styles that use a colored background behind large chunks of text. This coloring is accomplished with `\colorbox`, which cannot break across lines. It may be possible to create an alternative to `\colorbox` that supports line breaks, perhaps with TikZ, but the author is unaware of a satisfactory solution. The only current alternative is to redefine `\colorbox` so that it does nothing. For example,

```
\AtBeginEnvironment{minted}{\renewcommand{\colorbox}[3] [] {#3}}
```

uses the `etoolbox` package to redefine `\colorbox` within all `minted` environments.

`breaksymbol` (string) (default: `breaksymbolleft`)  
Alias for `breaksymbolleft`.

`breaksymbolindent` (dimension) (default: `<breaksymbolindentleftnchars>`)  
Alias for `breaksymbolindentleft`.

`breaksymbolindentnchars` (integer) (default: `<breaksymbolindentleftnchars>`)  
Alias for `breaksymbolindentleftnchars`.

`breaksymbolindentleft` (dimension) (default: `<breaksymbolindentleftnchars>`)  
The extra left indentation that is provided to make room for `breaksymbolleft`. This indentation is only applied when there is a `breaksymbolleft`.  
Does not apply to `\mintinline`.

`breaksymbolindentleftnchars` (integer) (default: 4)  
This allows `breaksymbolindentleft` to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).

`breaksymbolindentright` (dimension) (default: `<breaksymbolindentrightnchars>`)  
The extra right indentation that is provided to make room for `breaksymbolright`. This indentation is only applied when there is a `breaksymbolright`.

`breaksymbolindentrightnchars`(integer) (default: 4)

This allows `breaksymbolindentright` to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).

`breaksymbolleft` (string) (default: `\tiny\ensuremath{\hookrightarrow}`, `\leftarrow`)  
The symbol used at the beginning (left) of continuation lines when `breaklines=true`. To have no symbol, simply set `breaksymbolleft` to an empty string ("`=`," or "`={}`"). The symbol is wrapped within curly braces `{}` when used, so there is no danger of formatting commands such as `\tiny` "escaping."

The `\hookrightarrow` and `\hookleftarrow` may be further customized by the use of the `\rotatebox` command provided by `graphicx`. Additional arrow-type symbols that may be useful are available in the `dingbat` (`\carriagereturn`) and `mnsymbol` (hook and curve arrows) packages, among others.

Does not apply to `\mintinline`.

`breaksymbolright` (string) (default: `<none>`)  
The symbol used at breaks (right) when `breaklines=true`. Does not appear at the end of the very last segment of a broken line.

`breaksymbolsep` (dimension) (default: `<breaksymbolsepleftnchars>`)  
Alias for `breaksymbolsepleft`.

`breaksymbolsepnchars` (integer) (default: `<breaksymbolsepleftnchars>`)  
Alias for `breaksymbolsepleftnchars`.

`breaksymbolsepleft` (dimension) (default: `<breaksymbolsepleftnchars>`)  
The separation between the `breaksymbolleft` and the adjacent text.

`breaksymbolsepleftnchars` (integer) (default: 2)  
Allows `breaksymbolsepleft` to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).

`breaksymbolsepright` (dimension) (default: `<breaksymbolseprightnchars>`)  
The *minimum* separation between the `breaksymbolright` and the adjacent text. This is the separation between `breaksymbolright` and the furthest extent to which adjacent text could reach. In practice, `\linewidth` will typically not be an exact integer multiple of the character width (assuming a fixed-width font), so the actual separation between the `breaksymbolright` and adjacent text will generally be larger than `breaksymbolsepright`. This ensures that break symbols have the same spacing from the margins on both left and right. If the same spacing from text is desired instead, `breaksymbolsepright` may be adjusted. (See the definition of `\FV@makeLineNumber` in `fvextra` for implementation details.)

`breaksymbolseprightnchars` (integer) (default: 2)  
Allows `breaksymbolsepright` to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).

`codetagify` (single macro or backslash-escaped string) (default: `XXX, TODO, FIXME, BUG, NOTE`)  
Highlight special code tags in comments and docstrings.

The value must be a list of strings, either comma-delimited or space-delimited. The value must be a single macro that gives the desired text when fully expanded, or a string that is interpreted literally except that backslash escapes of ASCII punctuation characters are allowed to give the literal characters ("`\\`" for backslash, "`\#`" for "#", and so on).



`curlyquotes` (boolean) (default: `false`)  
By default, the backtick ``` and typewriter single quotation mark `'` always appear literally, instead of becoming the left and right curly single quotation marks `‘` `’`. This option allows these characters to be replaced by the curly quotation marks when that is desirable.

`encoding` (string) (default: `UTF-8`)  
File encoding used by `\inputminted` and derived commands when reading files.

`envname` (string) (default: `Verbatim`, or `VerbEnv` for inline)  
This is the name of the environment that wraps typeset code. By default, it is `Verbatim` in block contexts and `VerbEnv` in inline contexts (`\setminted` versus `\setmintedinline`). This is compatible with `fancyvrb`'s `BVerbatim`.

There are two requirements for using a custom environment other than `Verbatim` and `BVerbatim` in block contexts:

- For `minted` and `\mint` support, the custom environment must be created with `fancyvrb`'s `\DefineVerbatimEnvironment` or otherwise defined in a manner compatible with `fancyvrb`'s environment implementation conventions.
- For `\inputminted` support, a corresponding `\(envname)Insert` command must be defined, using `fancyvrb`'s `\CustomVerbatimCommand` or otherwise following `fancyvrb` command conventions. For example, using a custom variant of `BVerbatim` involves creating both a custom environment as well as a corresponding variant of `\BVerbatimInput`.

There is currently only limited support for using an environment other than `VerbEnv` in inline contexts. If an environment other than `VerbEnv` is specified, it will be used for highlighted code, but will not be used if code is typeset verbatim instead or if highlighting fails and a verbatim fallback is needed. In both of those cases, `\Verb` is currently used.

Note that `envname` is the name of the environment that wraps typeset code, but it is *not* the name of the environment that literally appears in highlighted code output. Highlighted code output uses the `MintedVerbatim` environment by default, and then `MintedVerbatim` is redefined based on `envname`. This allows a single cache file to be used in multiple contexts. The name of the environment that literally appears in highlighted code output can be modified with `literalenvname`, but there should be few if any situations where that is actually necessary.

`escapeinside` (single macro or backslash-escaped two-character string) (default: `<none>`)  
Escape to  $\TeX$  between the two characters specified. All code between the two characters will be interpreted as  $\TeX$  and typeset accordingly. This allows for additional formatting. The escape characters need not be identical.

The value must be a single macro that gives the desired text when fully expanded, or a string that is interpreted literally except that backslash escapes of ASCII punctuation characters are allowed to give the literal characters (“`\`” for backslash, “`\#`” for “`#`”, and so on). Special  $\TeX$  characters must be escaped when they are used as the escape characters (for example, `escapeinside=\#\%`).

**Escaping does not work inside strings and comments (for comments, there is `texcomments`). Escaping is “fragile” with some lexers.** Due to the way that Pygments implements `escapeinside`, any “escaped”  $\TeX$  code that resembles a string or comment for the current lexer may break `escapeinside`. There is a [Pygments issue](#) for this

case. Additional details and a limited workaround for some scenarios are available on the [minted GitHub site](#).

<pre>\setminted{escapeinside=  } \begin{minted}{py} def f(x):     y = x \colorbox{green}{**} 2     return y \end{minted}</pre>	<pre>def f(x):     y = x ** 2     return y</pre>
--	--

**Note that when math is used inside escapes, any active characters beyond those that are normally active in verbatim can cause problems.** Any package that relies on special active characters in math mode (for example, `icomma`) will produce errors along the lines of “TeX capacity exceeded” and “\leavevmode\kern\z@”. This may be fixed by modifying `\@noligs`, as described at <https://tex.stackexchange.com/questions/223876>.

<code>extrakeywords</code>	(string)	(default: <i>&lt;none&gt;</i> )
<code>extrakeywordsconstant</code>	(string)	(default: <i>&lt;none&gt;</i> )
<code>extrakeywordsdeclaration</code>	(string)	(default: <i>&lt;none&gt;</i> )
<code>extrakeywordsnamespace</code>	(string)	(default: <i>&lt;none&gt;</i> )
<code>extrakeywordspseudo</code>	(string)	(default: <i>&lt;none&gt;</i> )
<code>extrakeywordsreserved</code>	(string)	(default: <i>&lt;none&gt;</i> )
<code>extrakeywordstype</code>	(string)	(default: <i>&lt;none&gt;</i> )

Extra keywords for the current lexer. See the [Pygments documentation](#) for details about the different keyword classes.

Values must be lists of extra keywords, either comma-delimited or space-delimited.

<code>firstline</code>	(integer)	(default: 1)
The first line to be shown. All lines before that line are ignored and do not appear in the output.		
<code>firstnumber</code>	(auto   last   integer)	(default: auto = 1)
Line number of the first line.		
<code>fontencoding</code>	(font encoding)	(default: <i>&lt;doc encoding&gt;</i> )
Set font encoding used for typesetting code. For example, <code>fontencoding=T1</code> .		
<code>fontfamily</code>	(family name)	(default: <code>tt</code> )
The font family to use. <code>tt</code> , <code>courier</code> and <code>helvetica</code> are pre-defined.		
<code>fontseries</code>	(series name)	(default: auto – the same as the current font)
The font series to use.		
<code>fontshape</code>	(font shape)	(default: auto – the same as the current font)
The font shape to use.		
<code>fontsize</code>	(font size)	(default: auto – the same as the current font)
The size of the font to use, as a size command, e.g. <code>\footnotesize</code> .		
<code>formatcom</code>	(command)	(default: <i>&lt;none&gt;</i> )

A format to execute before printing verbatim text.

- `frame` (none | leftline | topline | bottomline | lines | single) (default: none)  
The type of frame to put around the source code listing. For more sophisticated framing, consider `tcolorbox`.
- `framerule` (dimension) (default: 0.4pt)  
Width of the frame.
- `framesep` (dimension) (default: \fboxsep)  
Distance between frame and content.
- `funcnamehighlighting` (boolean) (default: true)  
[For PHP only] If true, highlights built-in function names.
- `gobble` (integer) (default: 0)  
Remove the first  $n$  characters from each input line.  
When `autogobble` and `gobble` are used together, the effect is cumulative. First `autogobble` removes all common indentation, and then `gobble` is applied.  
`autogobble` and `gobble` operate on code before the highlighting process begins (before lexing), treating the code purely as text. Meanwhile, `gobblefilter` operates on the token stream generated by a lexer. If the removed characters are simply indentation coming from how the code was entered within  $\LaTeX$ , then `autogobble` and `gobble` should typically be preferred. If the removed characters are syntactically significant, then `gobblefilter` may be better. Which approach is preferable may also depend on the implementation details of the lexer.
- `gobblefilter` (integer) (default: 0)  
Remove the first  $n$  characters from each input line, using the Pygments `gobble` filter.  
`autogobble` and `gobble` operate on code before the highlighting process begins (before lexing), treating the code purely as text. Meanwhile, `gobblefilter` operates on the token stream generated by a lexer. If the removed characters are simply indentation coming from how the code was entered within  $\LaTeX$ , then `autogobble` and `gobble` should typically be preferred. If the removed characters are syntactically significant, then `gobblefilter` may be better. Which approach is preferable may also depend on the implementation details of the lexer.
- `highlightcolor` (string) (default: LightCyan)  
Set the color used for `highlightlines`, using a predefined color name from `color` or `xcolor`, or a color defined via `\definecolor`.
- `highlightlines` (string) (default: <none>)  
This highlights a single line or a range of lines based on line numbers. For example, `highlightlines={1, 3-4}`. The line numbers refer to the line numbers that would appear if `linenos=true`, etc. They do not refer to original or actual line numbers before adjustment by `firstnumber`.  
The highlighting color can be customized with `highlightcolor`.
- `ignorelexererrors` (boolean) (default: false)  
When lexer errors are shown in highlighted output (default), they are typically displayed as red boxes that surround the relevant text. When lexer errors are ignored, the literal text that caused lexer errors is shown but there is no indication that it caused errors.

```
\begin{minted}{python}
variable = !!!
\end{minted}
```

---

```
variable = !!!
```

```
\begin{minted}[ignorelexererrors=true]{python}
variable = !!!
\end{minted}
```

---

```
variable = !!!
```

`keywordcase` (lower | upper | capitalize | none) (default: none)  
Changes capitalization of keywords.

`label` (string) (default: *empty*)  
Add a label to the top, the bottom or both of the frames around the code. See the `fancyvrb` documentation for more information and examples. *Note:* This does *not* add a `\label` to the current listing. To achieve that, use a floating environment (section 5) instead.

`labelposition` (none | topline | bottomline | all) (default: topline, all, or none)  
Location for the label. Default: `topline` if one label is defined, `all` if two are defined, `none` otherwise. See the `fancyvrb` documentation for more information.

`lastline` (integer) (default: *last line of input*)  
The last line to be shown.

`linenos` (boolean) (default: false)  
Enables line numbers. In order to customize the display style of line numbers, you need to redefine the `\theFancyVerbLine` macro:

```
\renewcommand{\theFancyVerbLine}{
  \sffamily
  \textcolor[rgb]{0.5,0.5,1.0}{
    \scriptsize\oldstylenums{
      \arabic{FancyVerbLine}}}}
\begin{minted}[linenos,
  firstnumber=11]{python}
def all(iterable):
    for i in iterable:
        if not i:
            return False
    return True
\end{minted}
11 def all(iterable):
12     for i in iterable:
13         if not i:
14             return False
15     return True
```

<code>listparameters</code>	(macro)	(default: <code>&lt;empty&gt;</code> )
	fancyvrb option for setting list-related lengths to modify spacing around lines of code. For example, <code>listparameters=\setlength{\topsep}{0pt}</code> will remove space before and after a minted environment.	
<code>literalenvname</code>	(string)	(default: <code>MintedVerbatim</code> )
	This is the name of the environment that literally appears in highlighted code output as a wrapper around the code. It is redefined to be equivalent to <code>envname</code> . There should be few if any situations where modifying <code>literalenvname</code> rather than <code>envname</code> is actually necessary.	
<code>literatecomment</code>	(single macro or backslash-escaped string)	(default: <code>&lt;none&gt;</code> )
	This is for compatibility with literate programming formats, such as the <code>.dtx</code> format commonly used for writing $\LaTeX$ packages. If all lines of code begin with <code>&lt;literatecomment&gt;</code> , then <code>&lt;literatecomment&gt;</code> is removed from the beginning of all lines. For example, for <code>.dtx</code> , <code>literatecomment=\%</code> .	
	The value must be a single macro that gives the desired text when fully expanded, or a string that is interpreted literally except that backslash escapes of ASCII punctuation characters are allowed to give the literal characters (“ <code>\</code> ” for backslash, “ <code>\#</code> ” for “ <code>#</code> ”, and so on).	
<code>mathescape</code>	(boolean)	(default: <code>false</code> )
	Enable $\LaTeX$ math mode inside comments. Usage as in package listings. See the note under <code>escapeinside</code> regarding math and ligatures.	
<code>numberblanklines</code>	(boolean)	(default: <code>true</code> )
	Enables or disables numbering of blank lines.	
<code>numberfirstline</code>	(boolean)	(default: <code>false</code> )
	Always number the first line, regardless of <code>stepnumber</code> .	
<code>numbers</code>	(left   right   both   none)	(default: <code>none</code> )
	Essentially the same as <code>linenos</code> , except the side on which the numbers appear may be specified.	
<code>numbersep</code>	(dimension)	(default: <code>12pt</code> )
	Gap between numbers and start of line.	
<code>obeytabs</code>	(boolean)	(default: <code>false</code> )
	Treat tabs as tabs instead of converting them to spaces—that is, expand them to tab stops determined by <code>tabsize</code> . <b>While this will correctly expand tabs within leading indentation, usually it will not correctly expand tabs that are preceded by anything other than spaces or other tabs. It should be avoided in those case.</b>	
<code>python3</code>	(boolean)	(default: <code>true</code> )
	[For <code>PythonConsoleLexer</code> only] Specifies whether Python 3 highlighting is applied.	
<code>rangeregex</code>	(single macro or backslash-escaped string)	(default: <code>&lt;none&gt;</code> )
	Select code that matches this regular expression.	
	The value must be a single macro that gives the desired text when fully expanded, or a string that is interpreted literally except that backslash escapes of ASCII punctuation characters are allowed to give the literal characters (“ <code>\</code> ” for backslash, “ <code>\#</code> ” for “ <code>#</code> ”, and so on).	

If line numbers are displayed, they are based on the range of code that is selected; code that is discarded in selecting the range is not considered in calculating line numbers.

`rangeregexmatchnumber` (integer) (default: 1)  
If there are multiple non-overlapping matches for `rangeregex`, this determines which is used.

`rangeregexdotall` (boolean) (default: `false`)  
“.” matches any character including the newline.

`rangeregexmultiline` (boolean) (default: `false`)  
“^” and “\$” match at internal newlines, not just at the start/end of the string.

`rangestartafterstring` (single macro or backslash-escaped string) (default: `<none>`)  
Select code starting immediately after this string.

The value must be a single macro that gives the desired text when fully expanded, or a string that is interpreted literally except that backslash escapes of ASCII punctuation characters are allowed to give the literal characters (“\” for backslash, “\#” for “#”, and so on).

If line numbers are displayed, they are based on the range of code that is selected; code that is discarded in selecting the range is not considered in calculating line numbers.

`rangestartstring` (single macro or backslash-escaped string) (default: `<none>`)  
Select code starting with this string.

The value must be a single macro that gives the desired text when fully expanded, or a string that is interpreted literally except that backslash escapes of ASCII punctuation characters are allowed to give the literal characters (“\” for backslash, “\#” for “#”, and so on).

If line numbers are displayed, they are based on the range of code that is selected; code that is discarded in selecting the range is not considered in calculating line numbers.

`rangestopbeforestring` (single macro or backslash-escaped string) (default: `<none>`)  
Select code ending immediately before this string.

The value must be a single macro that gives the desired text when fully expanded, or a string that is interpreted literally except that backslash escapes of ASCII punctuation characters are allowed to give the literal characters (“\” for backslash, “\#” for “#”, and so on).

If line numbers are displayed, they are based on the range of code that is selected; code that is discarded in selecting the range is not considered in calculating line numbers.

`rangestopstring` (single macro or backslash-escaped string) (default: `<none>`)  
Select code ending with this string.

The value must be a single macro that gives the desired text when fully expanded, or a string that is interpreted literally except that backslash escapes of ASCII punctuation characters are allowed to give the literal characters (“\” for backslash, “\#” for “#”, and so on).

If line numbers are displayed, they are based on the range of code that is selected; code that is discarded in selecting the range is not considered in calculating line numbers.

bers.

<code>relabel</code>	(string) fancyvrb option for creating a label that can be used with <code>\pageref</code> .	(default: <code>&lt;none&gt;</code> )
<code>resetmargins</code>	(boolean) Resets the left margin inside other environments.	(default: <code>false</code> )
<code>rulecolor</code>	(color command) The color of the frame.	(default: <code>black</code> )
<code>samepage</code>	(boolean) Forces the whole listing to appear on the same page, even if it doesn't fit.	(default: <code>false</code> )
<code>showspaces</code>	(boolean) Enables visible spaces: <code>visible_spaces</code> .	(default: <code>false</code> )
<code>showtabs</code>	(boolean) Enables visible tabs—only works in combination with <code>obeytabs</code> .	(default: <code>false</code> )
<code>space</code>	(macro) Redefine the visible space character. Note that this is only used if <code>showspaces=true</code> .	(default: <code>\textvisiblespace</code> , <code>␣</code> )
<code>spacecolor</code>	(string) Set the color of visible spaces. By default ( <code>none</code> ), they take the color of their surroundings.	(default: <code>none</code> )
<code>startinline</code>	(boolean) [For PHP only] Specifies that the code starts in PHP mode, i.e., leading <code>&lt;?php</code> is omitted.	(default: <code>false</code> )
<code>stepnumber</code>	(integer) Interval at which line numbers appear.	(default: <code>1</code> )
<code>stepnumberfromfirst</code>	(boolean) By default, when line numbering is used with <code>stepnumber ≠ 1</code> , only line numbers that are a multiple of <code>stepnumber</code> are included. This offsets the line numbering from the first line, so that the first line, and all lines separated from it by a multiple of <code>stepnumber</code> , are numbered.	(default: <code>false</code> )
<code>stepnumberoffsetvalues</code>	(boolean) By default, when line numbering is used with <code>stepnumber ≠ 1</code> , only line numbers that are a multiple of <code>stepnumber</code> are included. Using <code>firstnumber</code> to offset the numbering will change which lines are numbered and which line gets which number, but will not change which <i>numbers</i> appear. This option causes <code>firstnumber</code> to be ignored in determining which line numbers are a multiple of <code>stepnumber</code> . <code>firstnumber</code> is still used in calculating the actual numbers that appear. As a result, the line numbers that appear will be a multiple of <code>stepnumber</code> , plus <code>firstnumber</code> minus 1.	(default: <code>false</code> )
<code>stripall</code>	(boolean) Strip all leading and trailing whitespace from the input.	(default: <code>false</code> )
<code>stripnl</code>	(boolean) Strip leading and trailing newlines from the input.	(default: <code>false</code> )
<code>style</code>	(string)	(default: <code>&lt;default&gt;</code> )

Sets the stylesheet used by Pygments.

<code>tab</code>	(macro)	(default: fancyvrb's <code>\FancyVerbTab</code> , ↲)
	Redefine the visible tab character. Note that this is only used if <code>showtabs=true</code> . <code>\rightarrowfill</code> , <code>→</code> , may be a nice alternative.	
<code>tabcolor</code>	(string)	(default: <code>black</code> )
	Set the color of visible tabs. If <code>tabcolor=none</code> , tabs take the color of their surroundings. This is typically undesirable for tabs that indent multiline comments or strings.	
<code>tabsize</code>	(integer)	(default: 8)
	The number of spaces a tab is equivalent to. If <code>obeytabs</code> is <i>not</i> active, tabs will be converted into this number of spaces. If <code>obeytabs</code> is active, tab stops will be set this number of space characters apart.	
<code>texcl</code>	(boolean)	(default: <code>false</code> )
	Enables $\LaTeX$ code inside comments. Usage as in package listings. See the note under <code>escapeinside</code> regarding math and ligatures.	
<code>vspace</code>	(dimension)	(default: <code>\topsep</code> )
	fancyvrb option for setting the value of the usual vertical list space.	
<code>texcomments</code>	(boolean)	(default: <code>false</code> )
	Enables $\LaTeX$ code inside comments. The newer name for <code>texcl</code> . See the note under <code>escapeinside</code> regarding math and ligatures. <code>texcomments</code> fails with multiline C/C++ preprocessor directives, and may fail in some other circumstances. This is because preprocessor directives are <b>tokenized as Comment.Preproc</b> , so <code>texcomments</code> causes preprocessor directives to be treated as literal $\LaTeX$ code. <b>An issue has been opened</b> at the Pygments site; additional details are also available on the <b>minted GitHub site</b> .	
<code>xleftmargin</code>	(dimension)	(default: 0)
	Indentation to add before the listing.	
<code>xrightmargin</code>	(dimension)	(default: 0)
	Indentation to add after the listing.	

## 8 Defining shortcuts

Large documents with many listings may use the same lexer and the same set of options for most listings. `minted` therefore defines a set of commands that lets you define shortcuts for the highlighting commands and environments. Each shortcut is specific to one lexer.

`\newminted` `\newminted` defines a new alias for the `minted` environment:



<pre>\newminted{cpp}{gobble=2,linenos}  \begin{cppcode}   template &lt;typename T&gt;   T id(T value) {     return value;   } \end{cppcode}</pre>	<pre>1 template &lt;typename T&gt; 2 T id(T value) { 3     return value; 4 }</pre>
---	--

If you want to provide extra options on the fly, or override existing default options, you can do that, too:

<pre>\newminted{cpp}{gobble=2,linenos}  \begin{cppcode}[linenos=false,                  frame=single]   int const answer = 42; \end{cppcode}</pre>	<pre>int const answer = 42;</pre>
--	-----------------------------------

The default name of the environment is `<lexer>code`. If this name clashes with another environment or if you want to choose a different name, you can use an optional argument: `\newminted[<environment name>]{<lexer>}{<options>}`.

Like normal minted environments, environments created with `\newminted` may be used within other environment definitions. Since the minted environments use `fancyvrb` internally, any environment based on them must include the `fancyvrb` command `\VerbatimEnvironment`. This allows `fancyvrb` to determine the name of the environment that is being defined, and correctly find its end. It is best to include this command at the beginning of the definition. For example,

```
\newminted{cpp}{gobble=2,linenos}
\newenvironment{env}{\VerbatimEnvironment\begin{cppcode}}{\end{cppcode}}
```

`\newmint` A shortcut for `\mint` is defined using `\newmint[<macro name>]{<lexer>}{<options>}`. The arguments and usage are identical to `\newminted`. If no `<macro name>` is specified, `<lexer>` is used.

<pre>\newmint{perl}{showspaces} \perl/my \$foo = \$bar;/</pre>	<pre>my \$foo = \$bar;</pre>
--	------------------------------

`\newmintinline` This creates custom versions of `\mintinline`. The syntax is the same as that for `\newmint`: `\newmintinline[<macro name>]{<lexer>}{<options>}`. If a `<macro name>` is not specified, then the created macro is called `<lexer>inline`.

<pre>\newmintinline{perl}{showspaces} \perlinline/my \$foo = \$bar;/</pre>	<pre>my \$foo = \$bar;</pre>
--	------------------------------

`\newmintedfile` This creates custom versions of `\inputminted`. The syntax is `\newmintedfile[<macro name>]{<lexer>}{<options>}`

If no `<macro name>` is given, then the macro is called `<lexer>file`.

## 9 FAQ and Troubleshooting

In some cases, minted may not give the desired result due to other document settings that it cannot control, or due to limitations in  $\LaTeX$  or the PDF format. Common issues are described below, with workarounds or solutions. You may also wish to search [tex.stackexchange.com](https://tex.stackexchange.com) or ask a question there, if you are working with minted in a non-typical context.

- **I can't copy and paste code out of a PDF created with minted. The line numbers also get copied, or whitespace is lost, or something else happens that makes the code incorrect.** There is no known method that always guarantees correct copy and paste for code in a PDF. This does not depend on whether minted is used. You may want to search [tex.stackexchange.com](https://tex.stackexchange.com) to find current approaches (and their limitations). You may also want to consider using `attachfile` or a similar package to bundle source code files as part of your PDF.
- **There are intermittent "I can't write on file" errors.** This can be caused by using minted in a directory that is synchronized with Dropbox or a similar file syncing program. These programs can try to sync minted's temporary files while it still needs to be able to modify them. The solution is to turn off file syncing or use a directory that is not synced.
- **I receive a "Font Warning: Some font shapes were not available" message, or bold or italic seem to be missing.** This is due to a limitation in the font that is currently in use for typesetting code. In some cases, the default font shapes that  $\LaTeX$  substitutes are perfectly adequate, and the warning may be ignored. In other cases, the font substitutions may not clearly indicate bold or italic text, and you will want to switch to a different font. See The  $\LaTeX$  Font Catalogue's section on [Typewriter Fonts](#) for alternatives. If you like the default  $\LaTeX$  fonts, the `lmodern` package is a good place to start. The `beramono` and `courier` packages may also be good options.
- **I receive a "Too many open files" error under macOS when using caching.** See the note on macOS under Section 6.2.
- **Weird things happen when I use the fancybox package.** `fancybox` conflicts with `fancyvrb`, which minted uses internally. When using `fancybox`, make sure that it is loaded before minted (or before `fancyvrb`, if `fancyvrb` is not loaded by minted).
- **When I use minted with KOMA-Script document classes, I get warnings about `\float@addtolist`.** minted uses the `float` package to produce floated listings, but this conflicts with the way KOMA-Script does floats. Load the package `scrhack` to resolve the conflict. Or use minted's `newfloat` package option.
- **Tilde characters ~ are raised, almost like superscripts.** This is a font issue. You need a different font encoding, possibly with a different font. Try `\usepackage[T1]{fontenc}`, perhaps with `\usepackage{lmodern}`, or something similar.
- **I'm getting errors with math, something like `TeX capacity exceeded and \leavevmode\kern\z@`.** This is due to ligatures being disabled within verbatim content. See the note under `escapeinside`.

- **With mathescape and the breqn package (or another special math package), the document never finishes compiling or there are other unexpected results.** Some math packages like breqn give certain characters like the comma special meanings in math mode. These can conflict with minted. In the breqn and comma case, this can be fixed by redefining the comma within minted environments:

```
\AtBeginEnvironment{minted}{\catcode`\,=12\mathcode`\,="613B}
```

Other packages/special characters may need their own modifications.

- **I'm getting errors with Beamer.** Due to how Beamer treats verbatim content, you may need to use either the fragile or fragile=singleslide options for frames that contain minted commands and environments. fragile=singleslide works best, but it disables overlays. fragile works by saving the contents of each frame to a temp file and then reusing them. This approach allows overlays, but will break if you have the string `\end{frame}` at the beginning of a line (for example, in a minted environment). To work around that, you can indent the content of the environment (so that the `\end{frame}` is preceded by one or more spaces) and then use the gobble or autogobble options to remove the indentation.
- **I'm trying to create several new minted commands/environments, and want them all to have the same settings. I'm saving the settings in a macro and then using the macro when defining the commands/environments. But it's failing.** This is due to the way that key-value processing operates. See [this](#) and [this](#) for more information. It is still possible to do what you want; you just need to expand the options macro before passing it to the commands that create the new commands/environments. An example is shown below. The `\expandafter` is the vital part.

```
\def\args{linenos,frame=single,fontsize=\footnotesize,style=bw}

\newcommand{\makenewmintedfiles}[1]{%
  \newmintedfile[inputlatex]{latex}{#1}%
  \newmintedfile[inputc]{c}{#1}%
}

\expandafter\makenewmintedfiles\expandafter{\args}
```

- **I want to use `\mintinline` in a context that normally doesn't allow verbatim content.** The `\mintinline` command will already work in many places that do not allow normal verbatim commands like `\verb`, so make sure to try it first. If it doesn't work, one of the simplest alternatives is to save your code in a box, and then use it later. For example,

```
\newsavebox\mybox
\begin{lrbox}{\mybox}
\mintinline{cpp}{std::cout}
\end{lrbox}

\commandthatdoesnotlikeverbatim{Text \usebox{\mybox}}
```

- **Extended characters do not work inside minted commands and environments, even when the inputenc package is used.** Version 2.0 adds support for extended characters under the pdfTeX engine. But if you need characters that are not supported by inputenc, you should use the XeTeX or LuaTeX engines instead.
- **The polyglossia package is doing undesirable things to code. (For example, adding extra space around colons in French.)** You may need to put your code within `\begin{english}... \end{english}`. This may be done for all minted environments using etoolbox in the preamble:

```
\usepackage{etoolbox}
\BeforeBeginEnvironment{minted}{\begin{english}}
\AfterEndEnvironment{minted}{\end{english}}
```

- **Tabs are being turned into the character sequence `^^I`.** This happens when you use XeLaTeX. You need to use the `-8bit` command-line option so that tabs may be written correctly to temporary files. See <https://tex.stackexchange.com/questions/58732/how-to-output-a-tabulation-into-a-file> for more on XeLaTeX's handling of tab characters.
- **The caption package produces an error when `\captionof` and other commands are used in combination with minted.** Load the caption package with the option `compatibility=false`. Or better yet, use minted's `newfloat` package option, which provides better caption compatibility.
- **I need a listing environment that supports page breaks.** The built-in listing environment is a standard float; it doesn't support page breaks. You will probably want to define a new environment for long floats. For example,

```
\usepackage{caption}
\newenvironment{longlisting}{\captionsetup{type=listing}}{}
```

With the caption package, it is best to use minted's `newfloat` package option. See <https://tex.stackexchange.com/a/53540/10742> for more on listing environments with page breaks.

- **I want to use the command-line option `-output-directory`, or MiKTeX's `-aux-directory`, but am getting errors.** With TeX Live 2024+, this should work automatically. Otherwise, set the environment variable `TEXMF_OUTPUT_DIRECTORY` to the desired location.
- **minted environments have extra vertical space inside tabular.** It is possible to [create a custom environment](#) that eliminates the extra space. However, a general solution that behaves as expected in the presence of adjacent text remains to be found.
- **I'm receiving a warning from `lineno.sty` that "Command `\@parboxrestore` has changed."** This can happen when minted is loaded after `csquotes`. Try loading minted first. If you receive this message when you are not using `csquotes`, you may want to experiment with the order of loading packages and might also open an issue.

- **I'm using texi2pdf, and getting "Cannot stat" errors from tar:** This is due to the way that texi2pdf handles temporary files. minted automatically cleans up its temporary files, but texi2pdf assumes that any temporary file that is ever created will still exist at the end of the run, so it tries to access the files that minted has deleted. It's possible to disable minted's temp file cleanup by adding `\renewcommand{\DeleteFile}[2][{}]` after the `\usepackage{minted}`.

## 10 Acknowledgements

**Konrad Rudolph:** Special thanks to Philipp Stephani and the rest of the guys from `comp.text.tex` and `tex.stackexchange.com`.

**Geoffrey Poore:**

- Thanks to Marco Daniel for the code on `tex.stackexchange.com` that inspired automatic line breaking.
- Thanks to Patrick Vogt for improving TikZ externalization compatibility.
- Thanks to @muzimuzhi for assistance with GitHub issues.
- Thanks to @jfbu for suggestions and discussion regarding support for arbitrary Pygments style names (`#210`, `#294`, `#299`, `#317`), and for debugging assistance.

## 11 Implementation

### 11.1 Required packages

```

1 \RequirePackage{catchfile}
2 \RequirePackage{etoolbox}
3 \RequirePackage{fvextra}[2024/10/16]
4 \RequirePackage{latex2pydata}[2024/10/16]
5 \RequirePackage{pdftexcmds}
6 \RequirePackage{pgfkeys}
7 \RequirePackage{pgfopts}
8 \RequirePackage{shellesc}

```

Make sure that either `color` or `xcolor` is loaded by the beginning of the document.

```

9 \AtEndPreamble{%
10 \IfPackageLoadedTF{color}%
11 {}%
12 {\IfPackageLoadedTF{xcolor}{\RequirePackage{xcolor}}}}

```

### 11.2 Exception handling

```

\minted@error
13 \def\minted@error#1{\PackageError{minted}{#1}{}}

\minted@fatalerror
\batchmode\read -1 to \minted@fatalerror@exitnow forces an immediate
exit with "! Emergency stop [...] cannot \read from terminal in nonstop
modes."
14 \def\minted@fatalerror#1{%

```

```

15 \minted@error{#1}%
16 \batchmode\read -1 to \minted@fatalerror@exitnow}
\minted@warning
17 \def\minted@warning#1{\PackageWarning{minted}{#1}}

```

### 11.3 Python executable and minimum supported version

```

\MintedExecutable
    Name of minted Python executable.
18 \edef\MintedExecutable{\detokenize{latexminted}}
\minted@executable@minversion
19 \edef\minted@executable@minversion{\detokenize{0.2.0}}
\minted@executable@minmajor
\minted@executable@minminor
\minted@executable@minpatch
20 \def\minted@executable@setminversioncomponents{%
21 \expandafter\minted@executable@setminversioncomponents@i
22 \minted@executable@minversion\relax}
23 \begingroup
24 \catcode`\.=12
25 \gdef\minted@executable@setminversioncomponents@i#1.#2.#3\relax{%
26 \def\minted@executable@minmajor{#1}%
27 \def\minted@executable@minminor{#2}%
28 \def\minted@executable@minpatch{#3}}
29 \endgroup
30 \minted@executable@setminversioncomponents
\minted@executable@version
31 \let\minted@executable@version\relax
minted@executable@exists
32 \newbool{minted@executable@exists}
minted@executable@issupported
33 \newbool{minted@executable@issupported}

```

### 11.4 Timestamp

Timestamp for current compile. This could eventually be simplified to use `\c_sys_timestamp_str` for all engines; that macro is in `l3kernel` from 2023-08-29.

The `\outputmode=1` is for `dvilualatex` compatibility.

The `\detokenize` is to prevent any possibility of `\catcode` issues.

```

\minted@timestamp
34 \begingroup
35 \catcode`\-=12
36 \catcode`\+=12
37 \catcode`\:=12
38 \def\minted@creationdatetotimestamp#1{%
39 \expandafter\minted@creationdatetotimestamp@i#1-\relax}
40 \def\minted@creationdatetotimestamp@i#1:#2-#3\relax{%

```

```

41 \minted@creationdatetotimestamp@ii#2+\relax}
42 \def\minted@creationdatetotimestamp@ii#1+#2\relax{%
43 #1}
44 \expandafter\ifx\csname pdftexversion\endcsname\relax
45 \else
46 \xdef\minted@timestamp{\minted@creationdatetotimestamp{\pdfcreationdate}}
47 \fi
48 \expandafter\ifx\csname XeTeXrevision\endcsname\relax
49 \else
50 \xdef\minted@timestamp{\minted@creationdatetotimestamp{\creationdate}}
51 \fi
52 \expandafter\ifx\csname directlua\endcsname\relax
53 \else
54 \begingroup
55 \outputmode=1
56 \xdef\minted@timestamp{\minted@creationdatetotimestamp{\pdffeedback creationdate}}%
57 \endgroup
58 \fi
59 \endgroup
60 \ifcsname minted@timestamp\endcsname
61 \else
62 \begingroup
63 \newcounter{minted@timestamp@hr}
64 \newcounter{minted@timestamp@min}
65 \setcounter{minted@timestamp@min}{\number\time}
66 \loop\unless\ifnum\value{minted@timestamp@min}<60\relax
67 \addtocounter{minted@timestamp@hr}{1}
68 \addtocounter{minted@timestamp@min}{-60}
69 \repeat
70 \xdef\minted@timestamp{%
71 \the\year
72 \ifnum\month<10 0\fi\the\month
73 \ifnum\day<10 0\fi\the\day
74 \ifnum\value{minted@timestamp@hr}<10 0\fi\theminted@timestamp@hr
75 \ifnum\value{minted@timestamp@min}<10 0\fi\theminted@timestamp@min}
76 \endgroup
77 \fi
78 \xdef\minted@timestamp{\detokenize\expandafter{\minted@timestamp}}

```

## 11.5 Jobname MD5 and derived file names

### `\MintedJobnameMdfive`

MD5 hash of `\jobname`. If `\jobname` contains spaces so that  $\LaTeX$  inserts wrapping quotes (single or double) within `\jobname`, these quotes are stripped, so that only the stem (basename without file extension) of the file path is hashed. This makes it simple to calculate the hash externally outside of  $\LaTeX$ .

`\MintedJobnameMdfive` is used for creating temp files rather than `\jobname` to avoid shell escaping issues. Under restricted shell escape, shell commands are quoted and escaped by  $\LaTeX$  itself, so using `\jobname` would work correctly in most cases. However, when full shell escape is enabled, no command escaping is performed by  $\LaTeX$ , so `minted` would have to quote/escape `\jobname` in a platform-specific manner. (See for example `web2c.info` and `texmfmp.c` in the TeX Live source for shell escape implementation details.) It is simpler to avoid escaping issues altogether, including edge

cases in the restricted shell escape scenario, by using an MD5 hash that is guaranteed to consist only of ASCII alphanumeric characters.

```

79 \begingroup
80 \catcode`\#=12
81 \catcode`\'=12
82 \gdef\minted@setjobnamemdfive#1#2\FV@Sentinel{%
83   \ifx#1"\relax
84     \let\minted@next\minted@setjobnamemdfive@dquoted
85   \else\ifx#1'\relax
86     \let\minted@next\minted@setjobnamemdfive@squoted
87   \else
88     \let\minted@next\minted@setjobnamemdfive@uquoted
89   \fi\fi
90   \minted@next#1#2\FV@Sentinel}
91 \gdef\minted@setjobnamemdfive@dquoted#1#2\FV@Sentinel{%
92   \minted@setjobnamemdfive@dquoted@i#2"\FV@Sentinel}
93 \gdef\minted@setjobnamemdfive@dquoted@i#1"#2\FV@Sentinel{%
94   \if\relax\detokenize{#2}\relax
95     \edef\MintedJobnameMdfive{\pdf@mdfivesum{\jobname}}%
96   \else\if\relax\detokenize\expandafter{\@gobble#2}\relax
97     \edef\MintedJobnameMdfive{\pdf@mdfivesum{#1}}%
98   \else
99     \edef\MintedJobnameMdfive{\pdf@mdfivesum{\jobname}}%
100   \fi\fi}
101 \gdef\minted@setjobnamemdfive@squoted#1#2\FV@Sentinel{%
102   \minted@setjobnamemdfive@squoted@i#2'\FV@Sentinel}
103 \gdef\minted@setjobnamemdfive@squoted@i#1'#2\FV@Sentinel{%
104   \if\relax\detokenize{#2}\relax
105     \edef\MintedJobnameMdfive{\pdf@mdfivesum{\jobname}}%
106   \else\if\relax\detokenize\expandafter{\@gobble#2}\relax
107     \edef\MintedJobnameMdfive{\pdf@mdfivesum{#1}}%
108   \else
109     \edef\MintedJobnameMdfive{\pdf@mdfivesum{\jobname}}%
110   \fi\fi}
111 \gdef\minted@setjobnamemdfive@uquoted#1\FV@Sentinel{%
112   \edef\MintedJobnameMdfive{\pdf@mdfivesum{#1}}}
113 \endgroup
114 \expandafter\minted@setjobnamemdfive\jobname\FV@Sentinel

```

#### \MintedCacheIndexFilename

Index file in cache. Used to detect whether cache exists.

```

115 \edef\MintedCacheIndexFilename{%
116   \detokenize_{}\MintedJobnameMdfive\detokenize{.index.minted}}

```

#### \MintedConfigFilename

File containing config info such as Python executable version. Written by the Python side, read by the  $\LaTeX$  side, and then immediately deleted.

```

117 \edef\MintedConfigFilename{%
118   \detokenize_{}\MintedJobnameMdfive\detokenize{.config.minted}}

```

#### \MintedDataFilename

Temp file for data. Written by the  $\LaTeX$  side, read by the Python side. Frequently overwritten, so only cleaned up at the end of the compile.

```

119 \edef\MintedDataFilename{%
120   \detokenize_{}\MintedJobnameMdfive\detokenize{.data.minted}}

```



`\MintedErrlogFilename`

Log file created when the Python side encounters an unexpected error that it is not designed to report to the  $\LaTeX$  side.

```
121 \edef\MintedErrlogFilename{%
122   \detokenize[_]\MintedJobnameMdfive\detokenize{.errlog.minted}}
```

`\MintedMessageFilename`

Messages from the Python side to the  $\LaTeX$  side. Deleted immediately after reading.

```
123 \edef\MintedMessageFilename{%
124   \detokenize[_]\MintedJobnameMdfive\detokenize{.message.minted}}
```

## 11.6 Package options

`\minted@pgfopts`

```
125 \def\minted@pgfopts#1{%
126   \pgfkeys{/minted/pkg/.cd,#1}}
```

### 11.6.1 Package option definitions

`\minted@float@within`

Control the section numbering of the listing float.

```
127 \minted@pgfopts{
128   chapter/.code=\def\minted@float@within{chapter},
129   chapter/.value forbidden,
130   section/.code=\def\minted@float@within{section},
131   section/.value forbidden,
132 }
```

`minted@newfloat`

Use `newfloat` rather than `float` to create a floating listing environment.

```
133 \newbool{minted@newfloat}
134 \minted@pgfopts{
135   newfloat/.is if=minted@newfloat,
136 }
```

`minted@debug`

Keep temp files for aid in debugging.

```
137 \newbool{minted@debug}
138 \minted@pgfopts{
139   debug/.is if=minted@debug,
140 }
```

`minted@cache`

Determine whether highlighted content is cached.

```
141 \newbool{minted@cache}
142 \booltrue{minted@cache}
143 \minted@pgfopts{
144   cache/.is if=minted@cache,
145 }
```

`\minted@cachedir`

Set the directory in which cached content is saved.

```
146 \edef\minted@cachedir{\detokenize[_minted]}
147 \minted@pgfopts{
```

```

148 cachedir/.estore in=\minted@cachedir,
149 }

```

#### `minted@frozenscache`

When a cache file is missing, raise an error instead of attempting to update the cache. This is intended for editing a document with a pre-existing cache in an environment in which `\ShellEscape` support is disabled or the `minted` executable is not available.

```

150 \newbool{minted@frozenscache}
151 \minted@pgfopts{
152   frozenscache/.is if=minted@frozenscache,
153 }

```

#### `minted@lexerlinenos`

Make all `minted` environments and `\mint` commands for a given lexer share cumulative line numbering (if `firstnumber=last`). `langlinenos` is for backward compatibility with `minted v2`.

```

154 \newbool{minted@lexerlinenos}
155 \minted@pgfopts{
156   lexerlinenos/.is if=minted@lexerlinenos,
157   langlinenos/.is if=minted@lexerlinenos,
158 }

```

#### `minted@inputlexerlinenos`

Enable `lexerlinenos` and make it apply to `\inputminted`. `inputlanglinenos` is for backward compatibility with `minted v2`.

```

159 \newbool{minted@inputlexerlinenos}
160 \minted@pgfopts{
161   inputlexerlinenos/.is if=minted@inputlexerlinenos,
162   inputlanglinenos/.is if=minted@inputlexerlinenos,
163 }

```

#### `minted@placeholder`

##### `\minted@insertplaceholder`

Cause all commands and environments to insert a placeholder rather than typesetting code. This functionality is primarily intended for use with PGF/TikZ externalization, when all non-PGF/TikZ features should be disabled.

```

164 \newbool{minted@placeholder}
165 \minted@pgfopts{
166   placeholder/.is if=minted@placeholder,
167 }
168 \gdef\minted@insertplaceholder{%
169   \ifbool{minted@isinline}%
170     {\begingroup
171       \fvset{extra=true}\Verb[formatcom=\color{red}\bfseries]{<MINTED>}%
172       \endgroup}%
173     {\begingroup
174       \par\noindent
175       \fvset{extra=true}\Verb[formatcom=\color{red}\bfseries]{<MINTED>}%
176       \par
177       \endgroup}}%

```

#### `minted@verbatim`

Typeset all code verbatim using `fancyvrb`; do not use Python at all.

```

178 \newbool{minted@verbatim}
179 \minted@pgfopts{

```

```

180 verbatim/.is if=mintedverbatim,
181 }

```

```

\minted@highlightmode@init
\minted@fasthighlightmode@checkstart
\minted@fasthighlightmode@checkend

```

Determine whether highlighting is performed immediately or at the end of the compile. Immediately means more overhead during the compile, but no second compile is required. Highlighting at the end of the compile means a second compile is required, but also makes highlighting much faster since there is only a single `\ShellEscape`.

`\minted@highlightmode@init` is invoked within `\minted@detectconfig` if the Python executable is available and enabled. For the `fastfirst` case, `\minted@highlightmode@init` requires the `\minted@cachepath` that is set within `\minted@detectconfig`.

`\minted@fasthighlightmode@checkend` is invoked at the end of the document within `\minted@clean`.

```

182 \newbool{minted@fasthighlightmode}
183 \newbool{minted@fasthighlightmode@open}
184 \minted@pgfopts{
185   highlightmode/.is choice,
186   highlightmode/fast/.code=
187     \let\minted@highlightmode@init\minted@highlightmode@init@fast,
188   highlightmode/fastfirst/.code=
189     \let\minted@highlightmode@init\minted@highlightmode@init@fastfirst,
190   highlightmode/immediate/.code=
191     \let\minted@highlightmode@init\minted@highlightmode@init@immediate,
192 }
193 \def\minted@highlightmode@init@fast{%
194   \global\booltrue{minted@fasthighlightmode}}
195 \def\minted@highlightmode@init@fastfirst{%
196   \IfFileExists{\minted@cachepath\MintedCacheIndexFilename}%
197     {\global\boolfalse{minted@fasthighlightmode}}
198     {\global\booltrue{minted@fasthighlightmode}}}
199 \def\minted@highlightmode@init@immediate{%
200   \global\boolfalse{minted@fasthighlightmode}}
201 \let\minted@highlightmode@init\minted@highlightmode@init@fastfirst
202 \def\minted@fasthighlightmode@checkstart{%
203   \ifbool{minted@fasthighlightmode}%
204     {\pydatawritelistopen
205       \global\booltrue{minted@fasthighlightmode@open}}%
206     {}%
207   \global\let\minted@fasthighlightmode@checkstart\relax}
208 \def\minted@fasthighlightmode@checkend{%
209   \ifbool{minted@fasthighlightmode@open}%
210     {\pydatasetfilename{\MintedDataFilename}%
211       \pydatawritelistclose
212       \pydataclosefilename{\MintedDataFilename}%
213       \global\boolfalse{minted@fasthighlightmode@open}%
214       \global\boolfalse{minted@fasthighlightmode}%
215       \begingroup
216       \minted@exec@batch
217       \ifx\minted@exec@warning\relax
218       \else
219         \expandafter\minted@exec@warning
220       \fi

```

```

221 \ifx\minted@exec@error\relax
222 \else
223 \expandafter\minted@exec@error
224 \fi
225 \endgroup
226 \global\boolfalse{minted@canexec}}%
227 }%
228 \global\let\minted@fasthighlightmode@checkend\relax}

```

### 11.6.2 Package options that are no longer supported or deprecated

**finalizcache** Old, no longer needed option from minted v2.

```

229 \minted@pgfopts{
230 finalizcache/.code=\minted@error{%
231 Package option "finalizcache" is no longer needed with minted v3+},
232 }

```

**kpsewhich** Old, no longer needed option from minted v2.

```

233 \minted@pgfopts{
234 kpsewhich/.code=\minted@error{%
235 Package option "kpsewhich" is no longer needed with minted v3+},
236 }

```

**outputdir** Old, no longer needed option from minted v2.

The empty `\minted@outputdir` is for backward compatibility with packages that depend on minted v2 internals.

```

237 \minted@pgfopts{
238 outputdir/.code=\minted@error{%
239 Package option "outputdir" is no longer needed with minted v3+;
240 the output directory is automatically detected for TeX Live 2024+,
241 and the environment variable \detokenize{TEXMF_OUTPUT_DIRECTORY}
242 can be set manually in other cases},
243 }
244 \def\minted@outputdir{}

```

**draft** Old, no longer supported option from minted v2. Improvements in caching combined with the new minted v3 package options `placeholder` and `verbatim` provide better alternatives.

```

245 \minted@pgfopts{
246 draft/.code=\minted@warning{%
247 Package option "draft" no longer has any effect with minted v3+},
248 }

```

**final** Old, no longer supported option from minted v2. Improvements in caching combined with the new minted v3 package options `placeholder` and `verbatim` provide better alternatives.

```

249 \minted@pgfopts{
250 final/.code=\minted@warning{%
251 Package option "final" no longer has any effect with minted v3+},
252 }

```

### 11.6.3 Package option processing

```
253 \ProcessPgfOptions{/minted/pkg}
254 \ifbool{minted@cache}{\minted@pgfopts{highlightmode=immediate, cachedir=,}}
255 \ifbool{minted@newfloat}{\RequirePackage{newfloat}}{\RequirePackage{float}}
256 \ifcsname tikzifexternalizing\endcsname
257   \ifx\tikzifexternalizing\relax
258   \else
259     \tikzifexternalizing{\booltrue{minted@placeholder}}{}
260   \fi
261 \fi
```

### 11.7 Util

`\minted@styleprefix`

Prefix for generating Pygments style names.

```
262 \def\minted@styleprefix{PYG}
```

`\minted@tempindex`

Temp index for looping.

```
263 \def\minted@tempindex{0}
```

`\minted@forcsvlist`

Wrapper for etoolbox `\forcsvlist`. Syntax: `\minted@forcsvlist{<handler>}{<listmacro>}`.

```
264 \def\minted@forcsvlist#1#2{%
265   \if\relax\detokenize\expandafter{\@gobble#2}\relax
266   \expandafter\minted@forcsvlist@exp
267   \else
268     \expandafter\minted@forcsvlist@i
269   \fi
270   {#2}{#1}}
271 \def\minted@forcsvlist@exp#1#2{%
272   \expandafter\minted@forcsvlist@i\expandafter{#1}{#2}}
273 \def\minted@forcsvlist@i#1#2{%
274   \forcsvlist{#2}{#1}}
```

`\minted@apptoprovidecs`

```
275 \def\minted@apptoprovidecs#1#2{%
276   \ifcsname#1\endcsname
277   \else
278     \expandafter\def\csname#1\endcsname{}%
279   \fi
280   \expandafter\let\expandafter\minted@tmp\csname#1\endcsname
281   \expandafter\def\expandafter\minted@tmp\expandafter{\minted@tmp#2}%
282   \expandafter\let\csname#1\endcsname\minted@tmp}
```

`\minted@const@pgfkeysnovalue`

```
283 \def\minted@const@pgfkeysnovalue{\pgfkeysnovalue}
```

`\minted@ensureatletter`

```
284 \def\minted@ensureatletter#1{%
285   \edef\minted@tmpatcat{\the\catcode`\@}%
286   \catcode`\@=11\relax
287   #1%
288   \catcode`\@=\minted@tmpatcat\relax}
```

### 11.7.1 Check whether a string matches the regex $^{[0-9A-Za-z_-]+}$

These macros are used to restrict possible names of highlighting styles on the  $\text{\LaTeX}$  side.

$\text{\minted@is}$ <char\_category><codepoint\_decimal>

Create macros used in determining whether a given character is part of a specified set of characters.

```
289 % [0-9]
290 \def\minted@tempindex{48}
291 \loop\unless\ifnum\minted@tempindex>57\relax
292   \expandafter\let\csname minted@isnum\minted@tempindex\endcsname\relax
293   \expandafter\let\csname minted@isalphanum\minted@tempindex\endcsname\relax
294   \expandafter\let
295     \csname minted@isalphanumhyphenunderscore\minted@tempindex\endcsname\relax
296   \edef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}
297 \repeat
298 % [A-Z]
299 \def\minted@tempindex{65}
300 \loop\unless\ifnum\minted@tempindex>90\relax
301   \expandafter\let\csname minted@isalpha\minted@tempindex\endcsname\relax
302   \expandafter\let\csname minted@isalphanum\minted@tempindex\endcsname\relax
303   \expandafter\let
304     \csname minted@isalphanumhyphenunderscore\minted@tempindex\endcsname\relax
305   \edef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}
306 \repeat
307 % [a-z]
308 \def\minted@tempindex{97}
309 \loop\unless\ifnum\minted@tempindex>122\relax
310   \expandafter\let\csname minted@isalpha\minted@tempindex\endcsname\relax
311   \expandafter\let\csname minted@isalphanum\minted@tempindex\endcsname\relax
312   \expandafter\let
313     \csname minted@isalphanumhyphenunderscore\minted@tempindex\endcsname\relax
314   \edef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}
315 \repeat
316 % [-]
317 \expandafter\let\csname minted@isalphanumhyphenunderscore45\endcsname\relax
318 % [_]
319 \expandafter\let\csname minted@isalphanumhyphenunderscore95\endcsname\relax
```

$\text{\minted@ifalphanumhyphenunderscore}$

Conditional based on whether first argument is ASCII alphanumeric, hyphen, or underscore.

```
320 \def\minted@ifalphanumhyphenunderscore#1#2#3{%
321   \if\relax\detokenize{#1}\relax
322     \expandafter\@firstoftwo
323   \else
324     \expandafter\@secondoftwo
325   \fi
326   {#3}%
327   {\expandafter\minted@ifalphanumhyphenunderscore@i\detokenize{#1}\FV@Sentinel{#2}{#3}}}
328 \def\minted@ifalphanumhyphenunderscore@i#1#2\FV@Sentinel{%
329   \if\relax#2\relax
330     \expandafter\minted@ifalphanumhyphenunderscore@iii
331   \else
332     \expandafter\minted@ifalphanumhyphenunderscore@ii
```

```

333 \fi
334 #1#2\FV@Sentinel}
335 \def\minted@ifalphanumhyphenunderscore@ii#1#2\FV@Sentinel{%
336 \ifcsname minted@isalphanumhyphenunderscore\number`#1\endcsname
337 \expandafter\minted@ifalphanumhyphenunderscore@i
338 \else
339 \expandafter\minted@ifalphanumhyphenunderscore@false
340 \fi
341 #2\FV@Sentinel}
342 \def\minted@ifalphanumhyphenunderscore@iii#1\FV@Sentinel{%
343 \ifcsname minted@isalphanumhyphenunderscore\number`#1\endcsname
344 \expandafter\minted@ifalphanumhyphenunderscore@true
345 \else
346 \expandafter\minted@ifalphanumhyphenunderscore@false
347 \fi
348 \FV@Sentinel}
349 \def\minted@ifalphanumhyphenunderscore@true\FV@Sentinel#1#2{#1}
350 \def\minted@ifalphanumhyphenunderscore@false#1\FV@Sentinel#2#3{#3}

```

## 11.8 State

`\minted@lexer`

Current lexer (language). Should be the empty macro if not set; it is used within `\ifcsname... \endcsname` to check for the existence of lexer-specific settings macros.

```
351 \let\minted@lexer\@empty
```

`minted@isinline`

Whether in command or environment.

```
352 \newbool{minted@isinline}
```

`minted@tmpcodebufferlength`

Length of buffer in which code to be highlighted is stored.

```
353 \def\minted@tmpcodebufferlength{0}
```

## 11.9 Calling minted executable

`minted@canexec`

```

354 \newbool{minted@canexec}
355 \booltrue{minted@canexec}
356 \ifnum\csname c_sys_shell_escape_int\endcsname=0\relax
357 \boolfalse{minted@canexec}
358 \fi
359 \ifbool{minted@frozensave}{\boolfalse{minted@canexec}}{}
360 \ifbool{minted@placeholder}{\boolfalse{minted@canexec}}{}
361 \ifbool{minted@verbatim}{\boolfalse{minted@canexec}}{}

```

`\minted@ShellEscapeMaybeMessages`

`\minted@ShellEscapeNoMessages`

```

362 \def\minted@ShellEscapeMaybeMessages#1{%
363 \ifbool{minted@debug}%
364 {\immediate\typeout{%
365 minted debug: shell escape at
366 \ifx\CurrentFile\@empty\else\CurrentFile\space\fi line \the\inputlineno: #1}}%

```

```

367     {}%
368     \let\minted@exec@warning\relax
369     \let\minted@exec@error\relax
370     \ifbool{minted@canexec}{\ShellEscape{#1}\minted@inputexecmessages{}}{}
371 \def\minted@ShellEscapeNoMessages#1{%
372     \ifbool{minted@debug}%
373     {\immediate\typeout{%
374         minted debug: shell escape at
375         \ifx\CurrentFile\empty\else\CurrentFile\space\fi line \the\inputlineno: #1}}%
376     {}%
377     \ifbool{minted@canexec}{\ShellEscape{#1}}{}}

\minted@execarg@debug
\minted@execarg@timestamp

378 \def\minted@execarg@debug{%
379     \ifbool{minted@debug}{\detokenize{ --debug }}{}}
380 \def\minted@execarg@timestamp{%
381     \detokenize{ --timestamp }\minted@timestamp\detokenize{ }}

\minted@inputexecmessages
    If temp file containing warning and/or error messages exists, \input and then
    delete.
382 \def\minted@inputexecmessages{%
383     \minted@ensureatletter{\InputIfFileExists{\MintedMessageFilename}{}}{}}

\minted@exec@batch
    Run in batch mode, for highlightmode=fast or highlightmode=fastfirst.
384 \def\minted@exec@batch{%
385     \minted@ShellEscapeMaybeMessages{%
386         \MintedExecutable
387         \detokenize{ batch }\minted@execarg@timestamp\minted@execarg@debug
388         \MintedJobnameMdfive}}

\minted@exec@config
    Detect configuration.
389 \def\minted@exec@config{%
390     \minted@ShellEscapeMaybeMessages{%
391         \MintedExecutable
392         \detokenize{ config }\minted@execarg@timestamp\minted@execarg@debug
393         \MintedJobnameMdfive}}

\minted@exec@styledef
    Create style definition.
394 \def\minted@exec@styledef{%
395     \minted@ShellEscapeMaybeMessages{%
396         \MintedExecutable
397         \detokenize{ styledef }\minted@execarg@timestamp\minted@execarg@debug
398         \MintedJobnameMdfive}}

\minted@exec@highlight
    Highlight code.
399 \def\minted@exec@highlight{%
400     \minted@ShellEscapeMaybeMessages{%
401         \MintedExecutable
402         \detokenize{ highlight }\minted@execarg@timestamp\minted@execarg@debug
403         \MintedJobnameMdfive}}

```



```

\minted@exec@clean
    Clean temp files and cache.
404 \def\minted@exec@clean{%
405   \minted@ShellEscapeNoMessages{%
406     \MintedExecutable
407     \detokenize{ clean }\minted@execarg@timestamp\minted@execarg@debug
408     \MintedJobnameMdfive}}

\minted@exec@cleanconfig
    Clean config temp file.
409 \def\minted@exec@cleanconfig{%
410   \minted@ShellEscapeNoMessages{%
411     \MintedExecutable
412     \detokenize{ cleanconfig }\minted@execarg@timestamp\minted@execarg@debug
413     \MintedJobnameMdfive}}

\minted@exec@cleantemp
    Clean all temp files.
414 \def\minted@exec@cleantemp{%
415   \minted@ShellEscapeNoMessages{%
416     \MintedExecutable
417     \detokenize{ cleantemp }\minted@execarg@timestamp\minted@execarg@debug
418     \MintedJobnameMdfive}}

```

## 11.10 Config detection

```

\minted@diddetectconfig

```

```

419 \newbool{\minted@diddetectconfig}

```

```

\minted@detectconfig

```

When the `\minted@canexec` bool is defined, it is set false if shell escape is completely disabled (`\c_sys_shell_escape_int=0`) or if execution is disabled by package options, so those cases don't need to be handled here.

If the Python executable is available, then it will create a `.config.minted` file to notify the  $\LaTeX$  side that it is present. This `.config.minted` file always contains a timestamp `\minted@executable@timestamp`, which is the timestamp passed directly to the executable as a command-line option. If the executable finds a `.data.minted` file, then it will extract the timestamp from this file and save it in the `.config.minted` file as `\minted@config@timestamp`; otherwise, the `.config.minted` file will not contain this timestamp. When  $\LaTeX$  loads the `.config.minted` file, the presence and values of these timestamps is used to determine whether the executable is present and whether the correct `.data.minted` file was located by the executable.

```

420 \def\minted@detectconfig{%
421   \ifbool{\minted@diddetectconfig}%
422     {}%
423     {\ifx\minted@cachedir\@empty
424       \gdef\minted@cachepath{}%
425     \else
426       \gdef\minted@cachepath{\minted@cachedir/}%
427     \fi
428     \ifbool{\minted@canexec}{\begingroup\minted@detectconfig@i\endgroup}{}%
429     \global\booltrue{\minted@diddetectconfig}}
430 \def\minted@detectconfig@i{%

```

```

431 \global\let\minted@executable@version\relax
432 \global\let\minted@executable@timestamp\relax
433 \global\let\minted@config@timestamp\relax
434 \pydatasetfilename{\MintedDataFilename}%
435 \pydatawritedictopen
436 \pydatawritekeyvalue{command}{config}%
437 \pydatawritekeyedvalue{jobname}{\jobname}%
438 \pydatawritekeyedvalue{timestamp}{\minted@timestamp}%
439 \pydatawritekeyedvalue{cachedir}{\minted@cachedir}%
440 \pydatawritedictclose
441 \pydataclosefilename{\MintedDataFilename}%
442 \minted@exec@config
443 \minted@ensureatletter{%
444   \InputIfFileExists{\MintedConfigFilename}{-}{-}%
445 \ifx\minted@executable@version\relax
446   \expandafter\minted@detectconfig@noexecutableorerrlog
447 \else
448   \expandafter\minted@detectconfig@ii
449 \fi}
450 \def\minted@detectconfig@noexecutableorerrlog{%
451 \global\boolfalse\minted@canexec}%
452 \ifnum\csname c_sys_shell_escape_int\endcsname=1\relax
453   \expandafter\@firstoftwo
454 \else
455   \expandafter\@secondoftwo
456 \fi
457   {\IfFileExists{\MintedErrlogFilename}%
458     {\minted@error{minted v3+ executable is not installed or is not added to PATH,
459       or MiKTeX is being used with \detokenize{-aux-directory} or
460       \detokenize{-output-directory} without setting a
461       \detokenize{TEXMF_OUTPUT_DIRECTORY} environment variable;
462       or there was an unexpected error (check "\MintedErrlogFilename")}}%
463     {\minted@error{minted v3+ executable is not installed or is not added to PATH,
464       or MiKTeX is being used with \detokenize{-aux-directory} or
465       \detokenize{-output-directory} without setting a
466       \detokenize{TEXMF_OUTPUT_DIRECTORY} environment variable}}}%
467   {\IfFileExists{\MintedErrlogFilename}%
468     {\minted@error{minted v3+ executable is not installed, is not added to PATH,
469       or is not permitted with restricted shell escape;
470       or MiKTeX is being used with \detokenize{-aux-directory} or
471       \detokenize{-output-directory} without setting a
472       \detokenize{TEXMF_OUTPUT_DIRECTORY} environment variable;
473       or there was an unexpected error (check "\MintedErrlogFilename")}}%
474     {\minted@error{minted v3+ executable is not installed, is not added to PATH,
475       or is not permitted with restricted shell escape;
476       or MiKTeX is being used with \detokenize{-aux-directory} or
477       \detokenize{-output-directory} without setting a
478       \detokenize{TEXMF_OUTPUT_DIRECTORY} environment variable}}}}
479 \def\minted@detectconfig@ii{%
480 \ifx\minted@timestamp\minted@config@timestamp
481   \expandafter\minted@detectconfig@iii
482 \else
483   \expandafter\minted@detectconfig@wrongtimestamp
484 \fi}

```

```

485 \def\minted@detectconfig@wrongtimestamp{%
486   \ifx\minted@timestamp\minted@executable@timestamp
487     \minted@exec@cleanconfig
488     \global\boolfalse{minted@canexec}%
489     \minted@error{minted v3 Python executable could not find output directory;
490       upgrade to TeX distribution that supports \detokenize{TEXMF_OUTPUT_DIRECTORY}
491       or set environment variable \detokenize{TEXMF_OUTPUT_DIRECTORY} manually)}%
492   \else
493     \expandafter\minted@detectconfig@noexecutableorerrlog
494   \fi}
495 \def\minted@detectconfig@iii{%
496   \minted@exec@cleanconfig
497   \ifx\minted@exec@warning\relax
498   \else
499     \expandafter\minted@exec@warning
500   \fi
501   \ifx\minted@exec@error\relax
502     \expandafter\minted@detectconfig@iv
503   \else
504     \expandafter\minted@detectconfig@error
505   \fi}
506 \def\minted@detectconfig@error{%
507   \global\boolfalse{minted@canexec}%
508   \minted@exec@error}
509 \def\minted@detectconfig@iv{%
510   \expandafter\minted@detectconfig@v\minted@executable@version\relax}
511 \begingroup
512 \catcode\.=12
513 \gdef\minted@detectconfig@v#1.#2.#3\relax{%
514   \def\minted@executable@major{#1}%
515   \def\minted@executable@minor{#2}%
516   \def\minted@executable@patch{#3}%
517   \minted@detectconfig@vi}
518 \endgroup
519 \def\minted@detectconfig@vi{%
520   \ifnum\minted@executable@major>\minted@executable@minmajor\relax
521     \global\booltrue{minted@executable@issupported}%
522   \else\ifnum\minted@executable@major=\minted@executable@minmajor\relax
523     \ifnum\minted@executable@minor>\minted@executable@minminor\relax
524       \global\booltrue{minted@executable@issupported}%
525     \else\ifnum\minted@executable@minor=\minted@executable@minminor\relax
526       \ifnum\minted@executable@patch<\minted@executable@minpatch\relax
527         \else
528           \global\booltrue{minted@executable@issupported}%
529         \fi
530       \fi\fi
531     \fi\fi
532   \ifbool{minted@executable@issupported}%
533     {\ifx\minted@config@cachepath\relax
534       \expandafter\@firstoftwo
535     \else
536       \expandafter\@secondoftwo
537     \fi
538     {\global\boolfalse{minted@canexec}%

```

```

539     \minted@error{minted Python executable returned incomplete configuration data;
540     this may indicate a bug in minted or file corruption}}%
541     {\global\let\minted@cachepath\minted@config@cachepath
542     \minted@highlightmode@init
543     \ifbool{minted@fasthighlightmode}{\newread\minted@intempfile}{}}}%
544     {\global\boolfalse{minted@canexec}}%
545     \minted@error{minted Python executable is version \minted@executable@version,
546     but version \minted@executable@minversion+ is required}}

```

## 11.11 Options

### 11.11.1 Option processing

`\minted@optcats`

`\minted@optkeyslist@<optcat>`

Option categories, along with lists of keys for each.

- `fv`: Passed on to `fancyvrb`. Options are stored in scope-specific lists, rather than in individual per-option macros.
- `py`: Passed to Python. Options are stored in scope-specific, individual per-option macros. Some of these are passed to `fancyvrb` when the Python executable isn't available or is disabled.
- `tex`: Processed in  $\LaTeX$ . Options are stored in scope-specific, individual per-option macros.

```

547 \begingroup
548 \catcode`\,=12
549 \gdef\minted@optcats{fv,py,tex}
550 \endgroup
551 \def\minted@do#1{\expandafter\def\csname minted@optkeyslist@#1\endcsname{}}
552 \minted@forcsvlist{\minted@do}{\minted@optcats}

```

`\minted@optscopes`

`\minted@optscopes@onlyblock`

Scopes for options. `cmd` scope is the scope of a single command or environment.

```

553 \begingroup
554 \catcode`\,=12
555 \gdef\minted@optscopes{global,lexer,globalinline,lexerinline,cmd}
556 \gdef\minted@optscopes@onlyblock{global,lexer,cmd}
557 \endgroup

```

`\minted@iflexerscope`

```

558 \let\minted@iflexerscope@lexer\relax
559 \let\minted@iflexerscope@lexerinline\relax
560 \def\minted@iflexerscope#1#2#3{%
561   \ifcsname minted@iflexerscope@#1\endcsname
562     \expandafter\@firstoftwo
563   \else
564     \expandafter\@secondoftwo
565   \fi
566   {#2}{#3}}

```

## `\mintedpgfkeyscreate`

Core macro for defining options.

Syntax: `\mintedpgfkeyscreate` [`<processor>`] {`<option category>`} {`<key(=value)? list>`}.

- Optional `<processor>` is a macro that processes `<value>`. It can take two forms.
  1. It can take a single argument. In this case, it is used to wrap `<value>`: `\processor{<value>}`. It is not invoked until `<value>` wrapped in `<processor>` is actually used.
  2. It can take two arguments. The first is the `<csname>` that the processed `<value>` should be stored in, and the second is `<value>`. In this case, `<processor>` is invoked immediately and stores the processed `<value>` in `<csname>`. See `\minted@opthandler@deforrestrictedescape` for an example of implementing this sort of `<processor>`.

`<processor>` is only supported for `py` and `tex` options.

- `<option category>` is `fv` (for fancyvrb), `py` (Python side of minted), or `tex` (L<sup>A</sup>T<sub>E</sub>X side of minted).
- If only `<key>` is given, then `<value>` defaults to `\pgfkeysnovalue`. In that case, options are defined so that they can be used in the future, but they are ignored until an explicit `<value>` is provided later. `fv` options are typically defined only with `<key>`. `py` and `tex` options are currently required to have an initial `<value>`. If a `<key>` is given an initial `<value>` when it is defined, then that `<value>` is stored for the `<key>` in the global scope. When an initial value is needed for a different scope such as `lexer` or `inline`, `\pgfkeys` is used directly (`\setminted` and `\setmintedinline` don't yet exist).
- `py` only: A default value for `<key>` (value used when only `<key>` is given without a value) can be specified with the syntax `key<default>=value`. Default values for `fv` options are already defined in `fancyvrb`, and currently the few `tex` options are the sort that always need an explicit value for clarity.

```
567 \def\minted@adoptkey#1#2{%
568   \ifcsname minted@optkeyslist@#1\endcsname
569   \else
570     \minted@fatalerror{Defining options under category "#1" is not supported}%
571   \fi
572   \expandafter\let\expandafter\minted@tmp\csname minted@optkeyslist@#1\endcsname
573   \ifx\minted@tmp\@empty
574     \def\minted@tmp{#2}%
575   \else
576     \expandafter\def\expandafter\minted@tmp\expandafter{\minted@tmp,#2}%
577   \fi
578   \expandafter\let\csname minted@optkeyslist@#1\endcsname\minted@tmp}
579 \newcommand*\mintedpgfkeyscreate}[3] [] {%
580   \mintedpgfkeyscreate@i{#1}{#2}{#3}}
581 \begingroup
582 \catcode`\==12
583 \gdef\mintedpgfkeyscreate@i#1#2#3{%
584   \def\minted@do##1{%
```

```

585     \minted@do@i##1=\FV@Sentinel}%
586 \def\minted@do@i##1=#2\FV@Sentinel{%
587     \minted@do@ii##1<>\FV@Sentinel}%
588 \def\minted@do@ii##1<##2>##3\FV@Sentinel{%
589     \minted@adoptkey{#2}{##1}}%
590 \minted@forcsvlist{\minted@do}{#3}%
591 \csname minted@pgfkeyscreate@#2\endcsname{#1}{#3}}
592 \endgroup

\minted@pgfkeyscreate@fv
\minted@fvoptlist@<scope>(@<lexer>)?
\minted@usefvopts
\minted@usefvoptsnopy
    Syntax: \minted@pgfkeyscreate@fv{<key(=value)? list>}.
    Options are stored in scope-specific lists. They are applied by passing these lists to
    \fvset. Individual option values are not retrievable.
    The \begingroup\fvset{...}\endgroup checks fancyvrb options at definition
    time so that any errors are caught immediately instead of when the options are used
    later elsewhere.
    \minted@usefvopts applies options via \fvset. \minted@useadditionalfvoptsnopy
    applies additional options that are usually handled on the Python side and is intended
    for situations where Python is not available or is not used, such as purely verbatim
    typesetting.
593 \def\minted@pgfkeyscreate@fv#1#2{%
594     \if\relax\detokenize{#1}\relax
595     \else
596         \minted@fatalerror{Processor macros are not supported in defining fancyvrb options}%
597     \fi
598     \minted@forcsvlist{\minted@pgfkeycreate@fv}{#2}}
599 \begingroup
600 \catcode`\==12
601 \gdef\minted@pgfkeycreate@fv#1{%
602     \minted@pgfkeycreate@fv@i#1=\FV@Sentinel}
603 \gdef\minted@pgfkeycreate@fv@i#1=#2\FV@Sentinel{%
604     \if\relax\detokenize{#2}\relax
605         \expandafter\minted@pgfkeycreate@fv@ii
606     \else
607         \expandafter\minted@pgfkeycreate@fv@iii
608     \fi
609     {#1}#2\FV@Sentinel}
610 \gdef\minted@pgfkeycreate@fv@ii#1\FV@Sentinel{%
611     \minted@pgfkeycreate@fv@iv{#1}{\minted@const@pgfkeysnovalue}}
612 \gdef\minted@pgfkeycreate@fv@iii#1#2=\FV@Sentinel{%
613     \minted@pgfkeycreate@fv@iv{#1}{#2}}
614 \endgroup
615 \def\minted@pgfkeycreate@fv@iv#1#2{%
616     \def\minted@do##1{%
617         \minted@iflexerscope{##1}%
618         {\minted@do@i{##1}{@\minted@lexer}}%
619         {\minted@do@i{##1}{}}}%
620 \def\minted@do@i##1##2{%
621     \pgfkeys{%
622         /minted/##1/.cd,

```

```

623     #1/.code={%
624     \def\minted@tmp{###1}%
625     \ifx\minted@tmp\minted@const@pgfkeysnovalue
626     \begingroup\fvset{#1}\endgroup
627     \minted@apptoprovidecs{\minted@fvoptlist@##1##2}{#1,}%
628     \else
629     \begingroup\fvset{#1={###1}}\endgroup
630     \minted@apptoprovidecs{\minted@fvoptlist@##1##2}{#1={###1},}%
631     \fi},
632   }%
633 }%
634 \minted@forcsvlist{\minted@do}{\minted@optscopes}%
635 \ifx\minted@const@pgfkeysnovalue#2\relax
636 \else
637   \pgfkeys{%
638     /minted/global/.cd,
639     #1={#2},
640   }%
641 \fi}
642 \def\minted@usefvopts{%
643 \ifbool{\minted@isinline}%
644   {\minted@forcsvlist{\minted@usefvopts@do}{\minted@optscopes}}%
645   {\minted@forcsvlist{\minted@usefvopts@do}{\minted@optscopes@onlyblock}}%
646 \def\minted@usefvopts@do#1{%
647 \minted@iflexerscope{#1}%
648   {\ifcsname \minted@fvoptlist@#1\minted@lexer\endcsname
649     \expandafter
650     \let\expandafter\minted@tmp\csname \minted@fvoptlist@#1\minted@lexer\endcsname
651     \expandafter\fvset\expandafter{\minted@tmp}%
652     \fi}%
653   {\ifcsname \minted@fvoptlist@#1\endcsname
654     \expandafter
655     \let\expandafter\minted@tmp\csname \minted@fvoptlist@#1\endcsname
656     \expandafter\fvset\expandafter{\minted@tmp}%
657     \fi}}
658 \def\minted@useadditionalfvoptsnoopy{%
659 \edef\minted@tmp{\minted@pyoptvalueof{gobble}}%
660 \ifx\minted@tmp\minted@const@pgfkeysnovalue
661 \else
662   \expandafter\minted@useadditionalfvoptsnoopy@fvsetvk
663   \expandafter{\minted@tmp}{gobble}%
664 \fi
665 \edef\minted@tmp{\minted@pyoptvalueof{mathescape}}%
666 \ifx\minted@tmp\minted@const@pgfkeysnovalue
667 \else
668   \expandafter\minted@useadditionalfvoptsnoopy@fvsetvk
669   \expandafter{\minted@tmp}{mathescape}%
670 \fi}
671 \def\minted@useadditionalfvoptsnoopy@fvsetvk#1#2{%
672   \fvset{#2={#1}}}

```

`\minted@pgfkeyscreate@py`

`\minted@pyoptvalueof`

Syntax: `\minted@pgfkeyscreate@py{<processor>}{<key(<default>)?=<initial value>`

*list*}).

Currently, initial values are required. The key processing macros are written to handle the possibility of optional initial values: If no initial value is set, use `\pgfkeysnovalue`, which is skipped in passing data to the Python side to invoke defaults.

`\mintedpyoptvalueof` is used for retrieving values via `\edef`.

```

673 \def\minted@pgfkeyscreate@py#1#2{%
674   \minted@forcsvlist{\minted@pgfkeycreate@py{#1}}{#2}}
675 \begingroup
676 \catcode`\==12
677 \catcode`\<=12
678 \catcode`\>=12
679 \gdef\minted@pgfkeycreate@py#1#2{%
680   \minted@pgfkeycreate@py{i{#1}#2=\FV@Sentinel}
681 \gdef\minted@pgfkeycreate@py@i#1#2=#3\FV@Sentinel{%
682   \if\relax\detokenize{#3}\relax
683     \expandafter\minted@pgfkeycreate@py@ii
684   \else
685     \expandafter\minted@pgfkeycreate@py@iii
686   \fi
687   {#1}{#2}#3\FV@Sentinel}
688 \gdef\minted@pgfkeycreate@py@ii#1#2\FV@Sentinel{%
689   \minted@pgfkeycreate@py@iv{#1}{\pgfkeysnovalue}#2<>\FV@Sentinel}
690 \gdef\minted@pgfkeycreate@py@iii#1#2#3=\FV@Sentinel{%
691   \minted@pgfkeycreate@py@iv{#1}{#3}#2<>\FV@Sentinel}
692 \gdef\minted@pgfkeycreate@py@iv#1#2#3<#4>#5\FV@Sentinel{%
693   \if\relax\detokenize{#4}\relax
694     \expandafter\@firstoftwo
695   \else
696     \expandafter\@secondoftwo
697   \fi
698   {\minted@pgfkeycreate@py@v{#1}{#3}{#2}{\minted@const@pgfkeysnovalue}}}%
699   {\minted@pgfkeycreate@py@v{#1}{#3}{#2}{#4}}}}
700 \endgroup
701 \def\minted@pgfkeycreate@py@v#1#2#3#4{%
702   \def\minted@do##1{%
703     \minted@iflexerscope{##1}%
704     {\minted@do@i{##1}{\minted@lexer}}%
705     {\minted@do@i{##1}{}}}
706   \def\minted@do@i##1##2{%
707     \if\relax\detokenize{#1}\relax
708       \pgfkeys{%
709         /minted/##1/.cd,
710         #2/.code={\expandafter\def\csname minted@pyopt@##1##2@#2\endcsname{###1}},
711       }%
712     \else
713       \pgfkeys{%
714         /minted/##1/.cd,
715         #2/.code={%
716           \def\minted@tmp{###1}%
717           \ifx\minted@tmp\minted@const@pgfkeysnovalue
718             \expandafter\let\csname minted@pyopt@##1##2@#2\endcsname\minted@tmp
719           \else\ifcsname minted@opthandler@immediate@\string#1\endcsname

```



```

720         #1{minted@pyopt@##1##2@#2}{####1}%
721     \else
722         \expandafter\def\csname minted@pyopt@##1##2@#2\endcsname{#1{####1}}%
723     \fi\fi},
724 }%
725 \fi
726 \ifx\minted@const@pgfkeysnovalue#4\relax
727     \pgfkeys{%
728         /minted/##1/.cd,
729         #2/.value required,
730     }%
731 \else
732     \pgfkeys{%
733         /minted/##1/.cd,
734         #2/.default={#4},
735     }%
736 \fi
737 }%
738 \minted@forcsvlist{\minted@do}{\minted@optscopes}%
739 \pgfkeys{%
740     /minted/global/.cd,
741     #2={#3},
742 }}
743 \def\mintedpyoptvalueof#1{%
744     \ifbool{minted@isinline}%
745         {\minted@pyoptvalueof@inline{#1}}%
746         {\minted@pyoptvalueof@block{#1}}
747 \def\minted@pyoptvalueof@inline#1{%
748     \ifcsname minted@pyopt@cmd@#1\endcsname
749         \unexpanded\expandafter\expandafter\expandafter{%
750             \csname minted@pyopt@cmd@#1\endcsname}%
751     \else\ifcsname minted@pyopt@lexerinline@minted@lexer @#1\endcsname
752         \unexpanded\expandafter\expandafter\expandafter{%
753             \csname minted@pyopt@lexerinline@minted@lexer @#1\endcsname}%
754     \else\ifcsname minted@pyopt@globalinline@#1\endcsname
755         \unexpanded\expandafter\expandafter\expandafter{%
756             \csname minted@pyopt@globalinline@#1\endcsname}%
757     \else\ifcsname minted@pyopt@lexer@minted@lexer @#1\endcsname
758         \unexpanded\expandafter\expandafter\expandafter{%
759             \csname minted@pyopt@lexer@minted@lexer @#1\endcsname}%
760     \else
761         \unexpanded\expandafter\expandafter\expandafter{%
762             \csname minted@pyopt@global@#1\endcsname}%
763     \fi\fi\fi\fi}
764 \def\minted@pyoptvalueof@block#1{%
765     \ifcsname minted@pyopt@cmd@#1\endcsname
766         \unexpanded\expandafter\expandafter\expandafter{%
767             \csname minted@pyopt@cmd@#1\endcsname}%
768     \else\ifcsname minted@pyopt@lexer@minted@lexer @#1\endcsname
769         \unexpanded\expandafter\expandafter\expandafter{%
770             \csname minted@pyopt@lexer@minted@lexer @#1\endcsname}%
771     \else
772         \unexpanded\expandafter\expandafter\expandafter{%
773             \csname minted@pyopt@global@#1\endcsname}%

```

```

774 \fi\fi}
\minted@pgfkeyscreate@tex
\mintedtexoptvalueof
\minted@usetexoptsnonpygments
    Syntax: \minted@pgfkeyscreate@tex{<processor>}{<key=initial value list>}.
    Currently, initial values are required. The key processing macros are written
    to handle the possibility of optional initial values: If no initial value is set, use
    \pgfkeysnovalue.
    \mintedtexoptvalueof is used for retrieving values via \edef.
    \minted@usetexoptsnonpygments applies the tex options that aren't used by
    Pygments. It is initially empty and is redefined after tex options are defined. Unlike the
    \minted@usefvopts case, it isn't possible to simply loop through all defined options;
    more specialized per-option handling is required, since some options are handled in
    separate Pygments-related macros and there is no equivalent of \fvset.
775 \def\minted@pgfkeyscreate@tex#1#2{%
776 \minted@forcsvlist{\minted@pgfkeycreate@tex{#1}}{#2}}
777 \begingroup
778 \catcode`\==12
779 \gdef\minted@pgfkeycreate@tex#1#2{%
780 \minted@pgfkeycreate@tex@i{#1}#2=\FV@Sentinel}
781 \gdef\minted@pgfkeycreate@tex@i#1#2=#3\FV@Sentinel{%
782 \if\relax\detokenize{#3}\relax
783 \expandafter\minted@pgfkeycreate@tex@ii
784 \else
785 \expandafter\minted@pgfkeycreate@tex@iii
786 \fi
787 {#1}{#2}#3\FV@Sentinel}
788 \gdef\minted@pgfkeycreate@tex@ii#1#2\FV@Sentinel{%
789 \minted@pgfkeycreate@tex@iv{#1}{#2}{\pgfkeysnovalue}}
790 \gdef\minted@pgfkeycreate@tex@iii#1#2#3=\FV@Sentinel{%
791 \minted@pgfkeycreate@tex@iv{#1}{#2}{#3}}
792 \endgroup
793 \def\minted@pgfkeycreate@tex@iv#1#2#3{%
794 \def\minted@do##1{%
795 \minted@iflexerscope{##1}%
796 {\minted@do@i{##1}{@\minted@lexer}}%
797 {\minted@do@i{##1}{}}}
798 \def\minted@do@i##1##2{%
799 \if\relax\detokenize{#1}\relax
800 \pgfkeys{%
801 /minted/##1/.cd,
802 #2/.code={\expandafter\def\csname minted@texopt@##1##2@#2\endcsname{###1}},
803 #2/.value required,
804 }%
805 \else
806 \pgfkeys{%
807 /minted/##1/.cd,
808 #2/.code={%
809 \def\minted@tmp{###1}%
810 \ifx\minted@tmp\minted@const\pgfkeysnovalue
811 \expandafter\let\csname minted@texopt@##1##2@#2\endcsname\minted@tmp
812 \else\ifcsname minted@opthandler@immediate\string#1\endcsname

```

```

813         #1{minted@texopt@##1##2@#2}{####1}%
814     \else
815         \expandafter\def\csname minted@texopt@##1##2@#2\endcsname{#1{####1}}%
816     \fi\fi},
817     #2/.value required,
818 }%
819 \fi
820 }%
821 \minted@forcsvlist{\minted@do}{\minted@optscopes}%
822 \pgfkeys{%
823     /minted/global/.cd,
824     #2={#3},
825 }}
826 \def\mintedtexoptvalueof#1{%
827     \ifbool{minted@isinline}%
828     {\minted@texoptvalueof@inline{#1}}%
829     {\minted@texoptvalueof@block{#1}}%
830 \def\minted@texoptvalueof@inline#1{%
831     \ifcsname minted@texopt@cmd@#1\endcsname
832     \unexpanded\expandafter\expandafter\expandafter{%
833     \csname minted@texopt@cmd@#1\endcsname}%
834     \else\ifcsname minted@texopt@lexerinline@minted@lexer @#1\endcsname
835     \unexpanded\expandafter\expandafter\expandafter{%
836     \csname minted@texopt@lexerinline@minted@lexer @#1\endcsname}%
837     \else\ifcsname minted@texopt@globalinline@#1\endcsname
838     \unexpanded\expandafter\expandafter\expandafter{%
839     \csname minted@texopt@globalinline@#1\endcsname}%
840     \else\ifcsname minted@texopt@lexer@minted@lexer @#1\endcsname
841     \unexpanded\expandafter\expandafter\expandafter{%
842     \csname minted@texopt@lexer@minted@lexer @#1\endcsname}%
843     \else
844     \unexpanded\expandafter\expandafter\expandafter{%
845     \csname minted@texopt@global@#1\endcsname}%
846     \fi\fi\fi\fi}
847 \def\minted@texoptvalueof@block#1{%
848     \ifcsname minted@texopt@cmd@#1\endcsname
849     \unexpanded\expandafter\expandafter\expandafter{%
850     \csname minted@texopt@cmd@#1\endcsname}%
851     \else\ifcsname minted@texopt@lexer@minted@lexer @#1\endcsname
852     \unexpanded\expandafter\expandafter\expandafter{%
853     \csname minted@texopt@lexer@minted@lexer @#1\endcsname}%
854     \else
855     \unexpanded\expandafter\expandafter\expandafter{%
856     \csname minted@texopt@global@#1\endcsname}%
857     \fi\fi}
858 \def\minted@usetexoptsnonpygments{}

```

### 11.11.2 Option handlers

`\minted@opthandler@deforrestrictedescape`

Syntax: `\minted@opthandler@deforrestrictedescape{<csname>}{<value>}`. *<value>* is processed and then the result is stored in *<csname>*.

Leave *<value>* unchanged if a single macro. Otherwise process it with `\FVExtraDetokenizeREscVArg`, which performs backslash escapes but restricted to ASCII symbols and punctuation.

This guarantees exact output (no issues with spaces due to detokenizing alphabetical control sequences).

The `\minted@opthandler@immediate@<macro_name>` tells option processing to invoke the macro immediately, instead of simply storing it as a value wrapper that will only be invoked when the value is used. This provides immediate error messages in the event of invalid escapes. `\FVExtraDetokenizeREscVArg` is not fully expandable, so waiting to invoke it later when *<value>* is expanded (`\edef`) isn't an option.

```

859 \def\minted@opthandler@deforrestrictedescape#1#2{%
860   \if\relax\detokenize{#2}\relax
861     \expandafter\def\csname#1\endcsname{#2}%
862   \else\if\relax\detokenize\expandafter{\@gobble#2}\relax
863     \ifcat\relax\noexpand#2%
864       \expandafter\expandafter\expandafter\minted@opthandler@deforrestrictedescape@i
865         \expandafter\@gobble\string#2\FV@Sentinel{#1}{#2}%
866     \else
867       \FVExtraDetokenizeREscVArg{\expandafter\def\csname#1\endcsname}{#2}%
868     \fi
869   \else
870     \FVExtraDetokenizeREscVArg{\expandafter\def\csname#1\endcsname}{#2}%
871   \fi\fi}
872 \def\minted@opthandler@deforrestrictedescape@i#1#2\FV@Sentinel#3#4{%
873   \ifcsname minted@isalpha\number`#1\endcsname
874     \expandafter\def\csname#3\endcsname{#4}%
875   \else
876     \FVExtraDetokenizeREscVArg{\expandafter\def\csname#3\endcsname}{#4}%
877   \fi}
878 \expandafter\let\csname
879   minted@opthandler@immediate@\string\minted@opthandler@deforrestrictedescape
880   \endcsname\relax

```

### 11.11.3 Option definitions

#### **fancyvrb**

- `tabcolor`: Visible tabs should have a specified color so that they don't change colors when used to indent multiline strings or comments.

```

881 \mintedpgfkeyscreate{fv}{
882   baselinestretch,
883   beameroverlays,
884   backgroundcolor,
885   backgroundcolorvphantom,
886   bgcolor,
887   bgcolorpadding,
888   bgcolorvphantom,
889   breakafter,
890   breakafterinrun,
891   breakaftersymbolpost,
892   breakaftersymbolpre,
893   breakanywhere,
894   breakanywhereinlinestretch,
895   breakanywheresymbolpost,
896   breakanywheresymbolpre,
897   breakautoindent,

```

898 breakbefore,  
899 breakbeforeinrun,  
900 breakbeforesymbolpost,  
901 breakbeforesymbolpre,  
902 breakbytoken,  
903 breakbytokenanywhere,  
904 breakindent,  
905 breakindentnchars,  
906 breaklines,  
907 breaksymbol,  
908 breaksymbolindent,  
909 breaksymbolindentleft,  
910 breaksymbolindentleftnchars,  
911 breaksymbolindentnchars,  
912 breaksymbolindentright,  
913 breaksymbolindentrightnchars,  
914 breaksymbolleft,  
915 breaksymbolright,  
916 breaksymbolsep,  
917 breaksymbolsepleft,  
918 breaksymbolsepleftnchars,  
919 breaksymbolsepnchars,  
920 breaksymbolsepright,  
921 breaksymbolseprightnchars,  
922 curlyquotes,  
923 fillcolor,  
924 firstline,  
925 firstnumber,  
926 fontencoding,  
927 fontfamily,  
928 fontseries,  
929 fontshape,  
930 fontsize,  
931 formatcom,  
932 frame,  
933 framerule,  
934 framesep,  
935 highlightcolor,  
936 highlightlines,  
937 label,  
938 labelposition,  
939 lastline,  
940 linenos,  
941 listparameters,  
942 numberblanklines,  
943 numberfirstline,  
944 numbers,  
945 numbersep,  
946 obeytabs,  
947 rellabel,  
948 resetmargins,  
949 rulecolor,  
950 samepage,  
951 showspaces,

```

952 showtabs,
953 space,
954 spacecolor,
955 stepnumber,
956 stepnumberfromfirst,
957 stepnumberoffsetvalues,
958 tab,
959 tabcolor=black,
960 tabsize,
961 vspace,
962 xleftmargin,
963 xrightmargin,
964 }

```

### minted (passed to Python)

- PHP should use `startinline` for `\mintinline`.

```

965 \mintedpgfkeyscreate{py}{
966 autogobble<true>=false,
967 encoding=utf8,
968 extrakeywords=,
969 extrakeywordsconstant=,
970 extrakeywordsdeclaration=,
971 extrakeywordsnamespace=,
972 extrakeywordspseudo=,
973 extrakeywordsreserved=,
974 extrakeywordstype=,
975 funcnamehighlighting<true>=true,
976 gobble=0,
977 gobblefilter=0,
978 keywordcase=none,
979 literalenvname=MintedVerbatim,
980 mathescape<true>=false,
981 python3<true>=true,
982 rangeregexmatchnumber=1,
983 rangeregexdotall<true>=false,
984 rangeregexmultiline<true>=false,
985 startinline<true>=false,
986 stripall<true>=false,
987 stripnl<true>=false,
988 texcl<true>=false,
989 texcomments<true>=false,
990 }
991 \mintedpgfkeyscreate[\minted@opthandler@deforrestrictedescape]{py}{
992 codetagify=,
993 escapeinside=,
994 literatecomment=,
995 rangestartstring=,
996 rangestartafterstring=,
997 rangestopstring=,
998 rangestopbeforestring=,
999 rangeregex=,
1000 }

```

```

1001 \let\minted@tmplexer\minted@lexer
1002 \def\minted@lexer{php}
1003 \pgfkeys{
1004   /minted/lexerinline/.cd,
1005   startinline=true,
1006 }
1007 \let\minted@lexer\minted@tmplexer

```

### minted (kept in $\LaTeX$ )

- The `\minted@def@optcl` is for backward compatibility with versions of `tcolorbox` that used this to define an `envname` option under `minted v2`.

```

1008 \mintedpgfkeyscreate{tex}{
1009   envname=Verbatim,
1010   ignorelexererrors=false,
1011   style=default,
1012 }
1013 \pgfkeys{
1014   /minted/globalinline/.cd,
1015   envname=VerbEnv,
1016 }
1017 \expandafter\def\expandafter\minted@usetexoptsnonpygments\expandafter{%
1018   \minted@usetexoptsnonpygments
1019   \edef\minted@literalenvname{\mintedpyoptvalueof{literalenvname}}%
1020   \edef\minted@envname{\mintedtexoptvalueof{envname}}%
1021   \expandafter\def\expandafter\minted@literalenv\expandafter{%
1022     \csname \minted@literalenvname\endcsname}%
1023   \expandafter\def\expandafter\minted@endliteralenv\expandafter{%
1024     \csname end\minted@literalenvname\endcsname}%
1025   \expandafter\expandafter\expandafter
1026     \let\expandafter\minted@literalenv\csname \minted@envname\endcsname
1027   \expandafter\expandafter\expandafter
1028     \let\expandafter\minted@endliteralenv\csname end\minted@envname\endcsname}%
1029   \ifcsname \minted@def@optcl\endcsname
1030     \ifx\minted@def@optcl\relax
1031       \let\minted@def@optcl\minted@undefined
1032     \fi
1033   \fi
1034   \providecommand{\minted@def@optcl}[4] [] {%
1035     \minted@warning{Macro \string\minted@def@optcl\space is deprecated with minted v3
1036       and no longer has any effect}}

```

## 11.12 Caching, styles, and highlighting

### 11.12.1 State

#### `minted@didcreatefiles`

Track whether any style definitions or highlighted code files were generated, so that file cleanup can be optimized. This is set true whenever there is an attempt to create style definitions or highlighted code files; even if the attempt fails, there will typically be leftover temp files.

```

1037 \newbool{\minted@didcreatefiles}

```

### 11.12.2 Cache management

```
\minted@addcachefilename  
\minted@numcachefiles  
\minted@cachefile<n>  
\minted@cachechecksum  
\mintedoldcachechecksum
```

Track cache files that are used, so that unused files can be removed.

The number of files is stored in a global macro rather than a counter to avoid counter scope issues. For example, `\includeonly` tracks and resets counters.

Also track the overall state of the cache using a sum of MD5 hashes of cache file names combined with the number of cache files. When no new cache files are created, this is used in determining whether the cache should be cleaned.

```
1038 \def\minted@numcachefiles{0}  
1039 \def\minted@addcachefilename#1{%  
1040   \ifbool{minted@canexec}%  
1041     {\xdef\minted@numcachefiles{\the\numexpr\minted@numcachefiles+1\relax}%  
1042     \expandafter\xdef\csname minted@cachefile\minted@numcachefiles\endcsname{#1}%  
1043     \edef\minted@tmp{\pdf@mdfivesum{#1}}%  
1044     \expandafter\minted@cachechecksum@update\expandafter{\minted@tmp}}%  
1045   }  
1046 \let\minted@cachechecksum\relax  
1047 \ifcsname mintedoldcachechecksum\endcsname  
1048 \else  
1049   \let\mintedoldcachechecksum\relax  
1050 \fi  
1051 \edef\minted@cachechecksum@files{\pdf@mdfivesum{}}  
1052 \expandafter\let\expandafter\minted@intfromhex\csname int_from_hex:n\endcsname  
1053 \expandafter\let\expandafter\minted@inttoHex\csname int_to_Hex:n\endcsname  
1054 \expandafter\let\expandafter\minted@intmod\csname int_mod:nn\endcsname  
1055 \def\minted@intmodsixteen#1{\minted@intmod{#1}{16}}  
1056 \def\minted@cachechecksum@update#1{%  
1057   \xdef\minted@cachechecksum@files{%  
1058     \expandafter\minted@cachechecksum@files@calc  
1059     \minted@cachechecksum@files\FV@Sentinel#1\FV@Sentinel}%  
1060   \xdef\minted@cachechecksum{%  
1061     \detokenize\expandafter{\minted@cachechecksum@files}%  
1062     \detokenize{:}%  
1063     \detokenize\expandafter{\minted@numcachefiles}}}  
1064 \def\minted@cachechecksum@files@calc#1#2\FV@Sentinel#3#4\FV@Sentinel{%  
1065   \minted@inttoHex{%  
1066     \the\numexpr  
1067       \minted@intmodsixteen{\minted@intfromhex{#1}+\minted@intfromhex{#3}}%  
1068     \relax}%  
1069   \if\relax\detokenize{#2}\relax  
1070     \expandafter\@gobble  
1071   \else  
1072     \expandafter\@firstofone  
1073   \fi  
1074   {\minted@cachechecksum@files@calc#2\FV@Sentinel#4\FV@Sentinel}}
```

```
\minted@clean
```

If the Python executable is available and was used, clean up temp files. If a cache is in use, also update the cache index and remove unused cache files.



Only create a .data.minted file if there is a cache list to save. Otherwise, no file is needed.

Runs within the hook `enddocument/afterlastpage` so that all typesetting is complete, and thus the cache list is complete. `\minted@fasthighlightmode@checkend` is included in the hook to guarantee correct ordering.

```

1075 \def\minted@clean{%
1076   \ifbool{minted@canexec}%
1077     {\ifbool{minted@didcreatefiles}%
1078       {\minted@clean@i}%
1079       {\ifbool{minted@fasthighlightmode}%
1080         {\minted@clean@i}%
1081         {\ifbool{minted@cache}%
1082           {\ifx\minted@cachechecksum\mintedoldcachechecksum
1083             \else
1084               \expandafter\minted@clean@i
1085             \fi}%
1086           {}}}%
1087     \ifbool{minted@fasthighlightmode}{\global\boolfalse{minted@canexec}}}%
1088   {}}
1089 \def\minted@clean@i{%
1090   \ifnum\minted@numcachefiles>0\relax
1091     \expandafter\minted@savecachelist
1092   \fi
1093   \ifbool{minted@fasthighlightmode}%
1094     {}%
1095     {\ifnum\minted@numcachefiles>0\relax
1096       \expandafter\minted@exec@clean
1097     \else
1098       \expandafter\minted@exec@cleantemp
1099     \fi}%
1100   \gdef\minted@numcachefiles{0}}
1101 \def\minted@savecachelist{%
1102   \pydatasetfilename{\MintedDataFilename}%
1103   \minted@fasthighlightmode@checkstart
1104   \pydatawritedictopen
1105   \pydatawritekeyvalue{command}{clean}%
1106   \pydatawritekeyedefvalue{jobname}{\jobname}%
1107   \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
1108   \pydatawritekeyedefvalue{cachepath}{\minted@cachepath}%
1109   \pydatawritekey{cachefiles}%
1110   \pydatawritelvalueliststart
1111   \pydatawritelvalueline{[]}%
1112   \gdef\minted@tempindex{1}%
1113   \loop\unless\ifnum\minted@tempindex>\minted@numcachefiles\relax
1114     \expandafter\minted@savecachelist@writecachefile\expandafter{%
1115       \csname minted@cachefile\minted@tempindex\endcsname}%
1116     \expandafter\global\expandafter
1117     \let\csname minted@cachefile\minted@tempindex\endcsname\minted@undefined
1118     \xdef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}%
1119   \repeat
1120   \pydatawritelvalueline{[]}%
1121   \pydatawritelvalueend
1122   \pydatawritedictclose

```

```

1123 \ifbool{minted@fasthighlightmode}{\pydataclosefilename{\MintedDataFilename}}
1124 \begingroup
1125 \catcode`\="=12
1126 \catcode`\,=12
1127 \gdef\minted@savecachelist@writecachefile#1{%
1128 \expandafter\pydatawritemlvalue\expandafter{\expandafter"#1",}}
1129 \endgroup
1130 \AddToHook{enddocument/afterlastpage}{%
1131 \minted@clean
1132 \minted@fasthighlightmode@checkend
1133 \ifbool{minted@cache}%
1134 {\immediate\write\@auxout{%
1135 \xdef\string\mintedoldcachechecksum{\string\detokenize{\minted@cachechecksum}}}}%
1136 {}

```

### 11.12.3 Style definitions

#### `\minted@patch@PygmentsStyledef`

The macros generated by Pygments must be patched: the single quote macro is redefined for upquote compatibility, and the hyphen is redefined to prevent unintended line breaks under LuaTeX.

```

1137 \def\minted@patch@PygmentsZsq{%
1138 \ifcsname\minted@styleprefix Zsq\endcsname
1139 \ifcsstring{\minted@styleprefix Zsq}{\char`'}{\minted@patch@PygmentsZsq@i}{}%
1140 \fi}
1141 \begingroup
1142 \catcode`\'=\active
1143 \gdef\minted@patch@PygmentsZsq@i{\def\PYGZsq{'}}
1144 \endgroup
1145 \def\minted@patch@PygmentsZhy{%
1146 \ifcsname\minted@styleprefix Zhy\endcsname
1147 \ifcsstring{\minted@styleprefix Zhy}{\char`-}{\def\PYGZhy{\mbox{-}}}{}%
1148 \fi}
1149 \def\minted@patch@ignorelexererrors{%
1150 \edef\minted@tmp{\mintedtexoptvalueof{ignorelexererrors}}%
1151 \ifdefstring{\minted@tmp}{true}%
1152 {\expandafter\let\csname\minted@styleprefix @tok@err\endcsname\relax}%
1153 {}
1154 \def\minted@patch@PygmentsStyledef{%
1155 \minted@patch@PygmentsZsq
1156 \minted@patch@PygmentsZhy
1157 \minted@patch@ignorelexererrors}

```

#### `\minted@VerbatimPygments`

Enable fancyvrb features for Pygments macros.

```

1158 \def\minted@VerbatimPygments{%
1159 \expandafter\minted@VerbatimPygments@i\expandafter{%
1160 \csname\minted@styleprefix\endcsname}}
1161 \def\minted@VerbatimPygments@i#1{%
1162 \VerbatimPygments{#1}{#1}}

```

#### `\minted@standardcatcodes`

Set standard catcodes. Used before `\input` of style definitions and in reading the optional argument of environments that wrap Pygments output.

```

1163 \def\minted@standardcatcodes{%
1164   \catcode`\=0
1165   \catcode`\{=1
1166   \catcode`\}=2
1167   \catcode`\#=6
1168   \catcode`\ =10
1169   \catcode`\@=11
1170   \catcode`\`=12
1171   \catcode`\==12
1172   \catcode`\+=12
1173   \catcode`\.=12
1174   \catcode`\,=12
1175   \catcode`\[=12
1176   \catcode`\]=12
1177   \catcode`\%=14}

```

### `\minted@defstyle`

Define highlighting style macros.

```

1178 \def\minted@defstyle{%
1179   \edef\minted@tmp{\mintedtexoptvalueof{style}}%
1180   \expandafter\minted@defstyle@i\expandafter{\minted@tmp}}
1181 \def\minted@defstyle@i#1{%
1182   \minted@ifalphanumhyphenunderscore{#1}%
1183   {\minted@defstyle@ii{#1}}%
1184   {\minted@error{Highlighting style is set to "#1" but only style names with
1185     alphanumeric characters, hyphens, and underscores are supported;
1186     falling back to default style}%
1187     \minted@defstyle@ii{default}}}}
1188 \def\minted@defstyle@ii#1{%
1189   \ifcsname minted@styledef@#1\endcsname
1190     \expandafter\@firstoftwo
1191   \else
1192     \expandafter\@secondoftwo
1193   \fi
1194   {\csname minted@styledef@#1\endcsname
1195     \minted@patch@PygmentsStyledef
1196     \minted@VerbatimPygments}%
1197   {\minted@defstyle@load{#1}}}

```

### `\minted@defstyle@load`

Certain catcodes are required when loading Pygments style definitions from file.

- At sign @ would be handled by the `\makeatletter` within the Pygments style definition if the style were brought in via `\input`, but `\makeatletter` doesn't affect tokenization with the `catchfile` approach.
- Percent % may not have its normal meaning within a .dtx file.
- Backtick ` is made active by some babel package options, such as `magyar`.
- Catcodes for other symbolic/non-alphanumeric characters may (probably rarely) not have their normal definitions.

`\endlinechar` also requires special handling to avoid introducing unwanted spaces.

The `\begingroup... \endgroup` around `\minted@exec@styledef` and associated messages is necessary to prevent errors related to the message file. If a style

does not exist, then the Python executable will create a `<hash>.message.minted` file, which is brought in via `\InputIfFileExists` and generates an error message. After this, there is an attempt to load the default style. If the default style needs to be generated, then `\InputIfFileExists` will attempt to bring in a `<hash>.message.minted` file regardless of whether it exists, unless it is wrapped in the `\begingroup... \endgroup`.

```

1198 \def\minted@catchfiledef#1#2{%
1199   \CatchFileDef{#1}{#2}{\minted@standardcatcodes\endlinechar=-1}}
1200 \def\minted@defstyle@load#1{%
1201   \minted@detectconfig
1202   \ifbool{minted@cache}%
1203     {\edef\minted@styledeffilename{#1\detokenize{.style.minted}}%
1204      \edef\minted@styledeffilepath{\minted@cachepath\minted@styledeffilename}%
1205      \IfFileExists{\minted@styledeffilepath}%
1206       {\minted@defstyle@input{#1}}%
1207       {\ifbool{minted@canexec}%
1208        {\minted@defstyle@generate{#1}}%
1209        {\minted@error{Missing definition for highlighting style "#1" (minted executable
1210         is unavailable or disabled); attempting to substitute fallback style}%
1211         \minted@defstyle@fallback{#1}}}}%
1212   {\edef\minted@styledeffilename{%
1213     \detokenize{_\MintedJobnameMdfive\detokenize{.style.minted}}%
1214     \let\minted@styledeffilepath\minted@styledeffilename
1215     \ifbool{minted@canexec}%
1216     {\minted@defstyle@generate{#1}}%
1217     {\minted@error{Missing definition for highlighting style "#1" (minted executable
1218      is unavailable or disabled); attempting to substitute fallback style}%
1219     \minted@defstyle@fallback{#1}}}}
1220 \def\minted@defstyle@input#1{%
1221   \begingroup
1222   \minted@catchfiledef{\minted@tmp}{\minted@styledeffilepath}%
1223   \minted@tmp
1224   \ifcsname\minted@styleprefix\endcsname
1225     \expandafter\@firstoftwo
1226   \else
1227     \expandafter\@secondoftwo
1228   \fi
1229   {\expandafter\global\expandafter\let\csname minted@styledef@#1\endcsname\minted@tmp
1230    \endgroup
1231    \ifbool{minted@cache}{\minted@addcachefilename{\minted@styledeffilename}}{}}%
1232   \csname minted@styledef@#1\endcsname
1233   \minted@patch@PygmentsStyledef
1234   \minted@VerbatimPygments}%
1235 \endgroup
1236 \ifbool{minted@canexec}%
1237   {\minted@warning{Invalid or corrupted style definition file
1238    "\minted@styledeffilename"; attempting to regenerate}}%
1239   \minted@defstyle@generate{#1}}%
1240 {\minted@error{Invalid or corrupted style definition file
1241  "\minted@styledeffilename"; attempting to substitute fallback style
1242  (minted executable is unavailable or disabled)}}%
1243 \minted@defstyle@fallback{#1}}}}
1244 \def\minted@defstyle@generate#1{%
1245   \pydatasetfilename{\MintedDataFilename}}%

```

```

1246 \minted@fasthighlightmode@checkstart
1247 \pydatawritedictopen
1248 \pydatawritekeyvalue{command}{styledef}%
1249 \pydatawritekeyedefvalue{jobname}{\jobname}%
1250 \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
1251 \pydatawritekeyedefvalue{currentfilepath}{\CurrentFilePath}%
1252 \pydatawritekeyedefvalue{currentfile}{\CurrentFile}%
1253 \pydatawritekeyedefvalue{inputlineno}{\the\inputlineno}%
1254 \pydatawritekeyedefvalue{cachepath}{\minted@cachepath}%
1255 \pydatawritekeyedefvalue{styledeffilename}{\minted@styledeffilename}%
1256 \pydatawritekeyvalue{style}{#1}%
1257 \pydatawritekeyedefvalue{commandprefix}{\minted@styleprefix}%
1258 \pydatawritedictclose
1259 \ifbool{minted@fasthighlightmode}%
1260   {\minted@defstyle@fallback{#1}}%
1261   {\pydataclosefilename{\MintedDataFilename}}%
1262   \begingroup
1263   \minted@exec@styledef
1264   \global\booltrue{minted@didcreatefiles}%
1265   \ifx\minted@exec@warning\relax
1266   \else
1267     \expandafter\minted@exec@warning
1268   \fi
1269   \ifx\minted@exec@error\relax
1270     \expandafter\minted@defstyle@generate@i
1271   \else
1272     \expandafter\minted@defstyle@generate@error
1273   \fi
1274   {#1}}}
1275 \def\minted@defstyle@generate@i#1{%
1276   \endgroup
1277   \begingroup
1278   \minted@catchfiledef{\minted@tmp}{\minted@styledeffilepath}%
1279   \minted@tmp
1280   \ifcsname\minted@styleprefix\endcsname
1281     \expandafter\@firstoftwo
1282   \else
1283     \expandafter\@secondoftwo
1284   \fi
1285   {\expandafter\global\expandafter\let\csname\minted@styledef@#1\endcsname\minted@tmp
1286   \endgroup
1287   \ifbool{minted@cache}{\minted@addcachefilename{\minted@styledeffilename}}{}}%
1288   \csname\minted@styledef@#1\endcsname
1289   \minted@patch@PygmentsStyledef
1290   \minted@VerbatimPygments}%
1291 \endgroup
1292 \minted@error{Failed to create style definition file "\minted@styledeffilename"
1293   (no error message, see "\MintedErrlogFilename" if it exists);
1294   attempting to substitute fallback style}%
1295 \minted@defstyle@fallback{#1}}}
1296 \def\minted@defstyle@generate@error#1{%
1297   \minted@exec@error
1298   \endgroup
1299   \minted@defstyle@fallback{#1}}

```

```

1300 \def\minted@defstyle@fallback#1{%
1301   \ifstrequal{#1}{default}%
1302     {\expandafter\global\expandafter
1303       \let\csname minted@styledef@default\endcsname\minted@styledeffallback}%
1304   {\ifcsname minted@styledef@default\endcsname
1305     \else
1306       \minted@defstyle@load{default}%
1307     \fi
1308   \expandafter\let\expandafter\minted@tmp\csname minted@styledef@default\endcsname
1309   \expandafter\global\expandafter\let\csname minted@styledef@#1\endcsname\minted@tmp}}

```

`\minted@styledeffallback`

Basic style definition to make `.highlight.minted` cache files usable if no styles exist, not even the default style, and no styles can be generated.

```

1310 \def\minted@styledeffallback{%
1311   \expandafter\def\csname\minted@styleprefix\endcsname##1##2{##2}%
1312   \expandafter\def\csname\minted@styleprefix Zbs\endcsname{\char`\\}%
1313   \expandafter\def\csname\minted@styleprefix Zus\endcsname{\char`\_}%
1314   \expandafter\def\csname\minted@styleprefix Zob\endcsname{\char`\{}%
1315   \expandafter\def\csname\minted@styleprefix Zcb\endcsname{\char`\}}%
1316   \expandafter\def\csname\minted@styleprefix Zca\endcsname{\char`\^}%
1317   \expandafter\def\csname\minted@styleprefix Zam\endcsname{\char`\&%
1318   \expandafter\def\csname\minted@styleprefix Zlt\endcsname{\char`\<%
1319   \expandafter\def\csname\minted@styleprefix Zgt\endcsname{\char`\>%
1320   \expandafter\def\csname\minted@styleprefix Zsh\endcsname{\char`\#}%
1321   \expandafter\def\csname\minted@styleprefix Zpc\endcsname{\char`\}%
1322   \expandafter\def\csname\minted@styleprefix Zdl\endcsname{\char`\$}%
1323   \expandafter\def\csname\minted@styleprefix Zhy\endcsname{\char`\-}%
1324   \expandafter\def\csname\minted@styleprefix Zsq\endcsname{\char`\'}%
1325   \expandafter\def\csname\minted@styleprefix Zdq\endcsname{\char`\"}%
1326   \expandafter\def\csname\minted@styleprefix Zti\endcsname{\char`\~}%
1327   \minted@patch@PygmentsStyledef
1328   \minted@VerbatimPygments}

```

#### 11.12.4 Lexer-specific line numbering

`\minted@FancyVerbLineTemp`

Temporary counter for storing and then restoring the value of `FancyVerbLine`. When using the `lexerlinenos` option, we need to store the current value of `FancyVerbLine`, then set `FancyVerbLine` to the current value of a lexer-specific counter, and finally restore `FancyVerbLine` to its initial value after the current chunk of code has been typeset.

```

1329 \newcounter{minted@FancyVerbLineTemp}

```

```

\minted@lexerlinenoson
\minted@lexerlinenosoff
\minted@inputlexerlinenoson
\minted@inputlexerlinenosoff

```

Line counters on a per-lexer basis for `\minted` and `\mintinline`; line counters on a per-lexer basis for `\inputminted`.

```

1330 \def\minted@lexerlinenoson{%
1331   \ifcsname c@minted@lexer\minted@lexer\endcsname
1332   \else

```

```

1333     \newcounter{minted@lexer\minted@lexer}%
1334     \fi
1335     \setcounter{minted@FancyVerbLineTemp}{\value{FancyVerbLine}}%
1336     \setcounter{FancyVerbLine}{\value{minted@lexer\minted@lexer}}
1337 \def\minted@lexerlinenosoff{%
1338     \setcounter{minted@lexer\minted@lexer}{\value{FancyVerbLine}}%
1339     \setcounter{FancyVerbLine}{\value{minted@FancyVerbLineTemp}}
1340 \ifbool{minted@inputlexerlinenos}%
1341 {\let\minted@inputlexerlinenoson\minted@lexerlinenoson
1342  \let\minted@inputlexerlinenosoff\minted@lexerlinenosoff}%
1343 {\let\minted@inputlexerlinenoson\relax
1344  \let\minted@inputlexerlinenosoff\relax
1345  \ifbool{minted@lexerlinenos}
1346   {}%
1347   {\let\minted@lexerlinenoson\relax
1348    \let\minted@lexerlinenosoff\relax}}

```

### `\minted@codewrapper`

Wrapper around typeset code. `\minted@inputfilepath` will exist when the code is brought in from an external file.

```

1349 \def\minted@codewrapper#1{%
1350     \ifcsname minted@inputfilepath\endcsname
1351         \minted@inputlexerlinenoson
1352     \else
1353         \minted@lexerlinenoson
1354     \fi
1355     #1%
1356     \ifcsname minted@inputfilepath\endcsname
1357         \minted@inputlexerlinenosoff
1358     \else
1359         \minted@lexerlinenosoff
1360     \fi}

```

## 11.12.5 Highlighting code

### `\minted@highlight`

#### `\minted@highlightinputfile`

Highlight code previously stored in buffer `minted@tmpdatabuffer`, or code in an external file.

`\minted@defstyle` will invoke `\minted@detectconfig` the first time a style is loaded, so no separate `\minted@detectconfig` is needed.

The default `\minted@highlight@fallback` inserts a placeholder. Typically commands/environments will redefine the fallback locally to insert a verbatim approximation of code that could not be highlighted.

Python-related options are buffered/written under a `pyopt` namespace. This prevents the possibility of naming collisions between options and other data that must be passed to Python.

Some data such as `jobname`, `timestamp`, and `cachepath` should be written to file, but not used in hashing because otherwise it would unnecessarily make the cache files dependent on irrelevant data.

```

1361 \def\minted@debug@input{%
1362     \ifbool{minted@debug}%
1363     {\immediate\typeout{%

```

```

1364     mnted debug: \string\input\space at
1365     \ifx\CurrentFile\@empty\else\CurrentFile\space\fi line \the\inputlineno}}%
1366     {}}
1367 \def\mnted@highlight{%
1368   \mnted@defstyle
1369   \pydatasetbuffername{mnted@tmpdatabuffer}%
1370   \pydatabufferkeyvalue{command}{highlight}%
1371   \pydatabufferkey{code}%
1372   \pydatabuffermlvaluestart
1373   \gdef\mnted@tempindex{1}%
1374   \loop\unless\ifnum\mnted@tempindex>\mnted@tmpcodebufferlength\relax
1375     \expandafter\let\expandafter
1376     \mnted@tmp\csname mnted@tmpcodebufferline\mnted@tempindex\endcsname
1377     \expandafter\pydatabuffermlvalue\expandafter{\mnted@tmp}%
1378     \xdef\mnted@tempindex{\the\numexpr\mnted@tempindex+1\relax}%
1379   \repeat
1380   \pydatabuffermlvalueend
1381   \mnted@highlight@i}
1382 \def\mnted@highlightinputfile{%
1383   \mnted@defstyle
1384   \edef\mnted@inputfilemdfivesum{\pdf@filemdfivesum{\mnted@inputfilepath}}%
1385   \ifx\mnted@inputfilemdfivesum\@empty
1386     \expandafter\@firstoftwo
1387   \else
1388     \expandafter\@secondoftwo
1389   \fi
1390   {\mnted@error{Cannot find input file "\mnted@inputfilepath"; inserting placeholder}}%
1391   \mnted@insertplaceholder}%
1392   {\pydatasetbuffername{mnted@tmpdatabuffer}%
1393   \pydatabufferkeyvalue{command}{highlight}%
1394   \pydatabufferkeyedefvalue{inputfilepath}{\mnted@inputfilepath}%
1395   \pydatabufferkeyedefvalue{inputfilemdfivesum}{\mnted@inputfilemdfivesum}%
1396   \mnted@highlight@i}}
1397 \def\mnted@def@FV@GetKeyValues@standardcatcodes{%
1398   \let\mnted@FV@GetKeyValues@orig\FV@GetKeyValues
1399   \def\FV@GetKeyValues##1{%
1400     \begingroup
1401     \mnted@standardcatcodes
1402     \mnted@FV@GetKeyValues@i{##1}}%
1403   \def\mnted@FV@GetKeyValues@i##1[##2]{%
1404     \endgroup
1405     \let\FV@GetKeyValues\mnted@FV@GetKeyValues@orig
1406     \let\mnted@FV@GetKeyValues@i\mnted@undefined
1407     \FV@GetKeyValues{##1}[##2]}}
1408 \def\mnted@highlight@i{%
1409   \pydatabufferkeyedefvalue{pyopt.lexer}{\mnted@lexer}%
1410   \pydatabufferkeyedefvalue{pyopt.commandprefix}{\mnted@styleprefix}%
1411   \mnted@forcsvlist{\mnted@highlight@bufferpykeys}{\mnted@optkeyslist@py}%
1412   \ifbool{mnted@cache}%
1413     {\edef\mnted@highlightfilename{\pydatabuffermdfivesum\detokenize{.highlight.mnted}}}%
1414     \edef\mnted@highlightfilepath{\mnted@cachepath\mnted@highlightfilename}%
1415     \IfFileExists{\mnted@highlightfilepath}%
1416     {\mnted@codewrapper{%
1417       \mnted@def@FV@GetKeyValues@standardcatcodes

```



```

1418     \minted@debug@input
1419     \input{\minted@highlightfilepath}}%
1420     \minted@addcachefilename{\minted@highlightfilename}}%
1421     {\ifbool{minted@canexec}%
1422     {\ifbool{minted@fasthighlightmode}%
1423     {\ifcsname minted@processedfilename@\minted@highlightfilename\endcsname
1424     \expandafter\@firstoftwo
1425     \else
1426     \expandafter\@secondoftwo
1427     \fi
1428     {\minted@insertplaceholder}%
1429     {\expandafter\global\expandafter\let
1430     \csname minted@processedfilename@\minted@highlightfilename\endcsname\relax
1431     \minted@iffasthighlightmode@buffertempfile
1432     \minted@highlight@create}}%
1433     {\minted@iffasthighlightmode@buffertempfile
1434     \minted@highlight@create}}%
1435     {\minted@error{Cannot highlight code (minted executable is unavailable or
1436     disabled); attempting to typeset without highlighting}}%
1437     \minted@highlight@fallback}}}%
1438     {\edef\minted@highlightfilename{%
1439     \detokenize{_\MintedJobnameMdfive\detokenize{.highlight.minted}}}%
1440     \let\minted@highlightfilepath\minted@highlightfilename
1441     \ifbool{minted@canexec}%
1442     {\minted@highlight@create}%
1443     {\minted@error{Cannot highlight code (minted executable is unavailable or
1444     disabled); attempting to typeset without highlighting}}%
1445     \minted@highlight@fallback}}%
1446     \pydataclearbuffername{minted@tmpdatabuffer}}
1447     \def\minted@highlight@bufferpykeys#1{%
1448     \edef\minted@tmp{\mintedpyoptvalueof{#1}}%
1449     \ifx\minted@tmp\minted@const@pgfkeysnovalue
1450     \else
1451     \pydatabufferkeyedefvalue{pyopt.#1}{\minted@tmp}%
1452     \fi}
1453     \def\minted@highlight@create{%
1454     \pydatasetfilename{\MintedDataFilename}%
1455     \minted@fasthighlightmode@checkstart
1456     \pydatawritedictopen
1457     \pydatawritebuffer
1458     \pydatawritekeyedefvalue{jobname}{\jobname}%
1459     \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
1460     \pydatawritekeyedefvalue{currentfilepath}{\CurrentFilePath}%
1461     \pydatawritekeyedefvalue{currentfile}{\CurrentFile}%
1462     \pydatawritekeyedefvalue{inputlineno}{\the\inputlineno}%
1463     \pydatawritekeyedefvalue{cachepath}{\minted@cachepath}%
1464     \pydatawritekeyedefvalue{highlightfilename}{\minted@highlightfilename}%
1465     \pydatawritedictclose
1466     \ifbool{minted@fasthighlightmode}%
1467     {\minted@insertplaceholder}%
1468     {\pydataclosefilename{\MintedDataFilename}}%
1469     \begingroup
1470     \minted@exec@highlight
1471     \global\booltrue{minted@didcreatefiles}}%

```

```

1472 \IfFileExists{\minted@highlightfilepath}%
1473 {\ifx\minted@exec@warning\relax
1474 \else
1475 \expandafter\minted@exec@warning
1476 \fi
1477 \ifx\minted@exec@error\relax
1478 \else
1479 \expandafter\minted@exec@error
1480 \fi
1481 \endgroup
1482 \minted@codewrapper{%
1483 \minted@def@FV@GetKeyValues@standardcatcodes
1484 \minted@debug@input
1485 \input{\minted@highlightfilepath}}%
1486 \ifbool{minted@cache}{\minted@addcachefilename{\minted@highlightfilename}}{}}%
1487 {\ifx\minted@exec@warning\relax
1488 \else
1489 \expandafter\minted@exec@warning
1490 \fi
1491 \ifx\minted@exec@error\relax
1492 \minted@error{Minted executable failed during syntax highlighting
1493 but returned no error message (see if "\MintedErrlogFilename" exists)}}%
1494 \else
1495 \expandafter\minted@exec@error
1496 \fi
1497 \endgroup
1498 \minted@highlight@fallback}}%
1499 \def\minted@highlight@fallback{%
1500 \minted@insertplaceholder}

```

#### `\minted@iffasthighlightmode@buffertempfile`

With caching, when `fasthighlightmode=true` and `\minted@inputfilepath` has a file extension that has been designated for temp files, read the file into a temp buffer, and then if that succeeds copy the code into the highlighting buffer. This can avoid errors with temp files that are modified or deleted before highlighting occurs when `fasthighlightmode=true`, since that delays highlighting until the end of the document.

This is not done for all highlighted external files because it adds overhead and complexity. When files are read, it is not possible to determine the newline sequence (`\n` versus `\r\n`), and trailing whitespace is discarded by  $\text{T}_{\text{E}}\text{X}$  during the reading process, so it is not possible to reconstruct the original file bytes within  $\text{T}_{\text{E}}\text{X}$ , only an (essentially equivalent) approximation. As a result, files that are read are hashed a second time after reading to reduce the chance that they were modified after initial hashing but before reading.

```

1501 \begingroup
1502 \def\minted@set@tempfileextension#1{%
1503 \if\relax\detokenize{#1}\relax
1504 \else
1505 \expandafter\global\expandafter
1506 \let\csname minted@buffertempfileextension@#1\endcsname\relax
1507 \fi}
1508 \minted@forcsvlist{\minted@set@tempfileextension}{
1509 listing,

```

```

1510 out,
1511 outfile,
1512 output,
1513 temp,
1514 tempfile,
1515 tmp,
1516 verb,
1517 vrb,
1518 }
1519 \endgroup
1520 \begingroup
1521 \catcode`\/=12
1522 \catcode`\.=12
1523 \gdef\minted@iffasthighlightmode@buffertempfile{%
1524   \ifbool{minted@fasthighlightmode}%
1525     {\ifcsname minted@inputfilepath\endcsname
1526       \expandafter\@firstofone
1527     \else
1528       \expandafter\@gobble
1529     \fi
1530     {\expandafter
1531       \minted@iffasthighlightmode@buffertempfile@i\minted@inputfilepath/\FV@Sentinel}}%
1532   {}}
1533 \gdef\minted@iffasthighlightmode@buffertempfile@i#1/#2\FV@Sentinel{%
1534   \if\relax\detokenize{#2}\relax
1535     \expandafter\@firstoftwo
1536   \else
1537     \expandafter\@secondoftwo
1538   \fi
1539   {\minted@iffasthighlightmode@buffertempfile@ii#1.\FV@Sentinel}}
1540   {\minted@iffasthighlightmode@buffertempfile@i#2\FV@Sentinel}}
1541 \gdef\minted@iffasthighlightmode@buffertempfile@ii#1.#2\FV@Sentinel{%
1542   \if\relax\detokenize{#2}\relax
1543     \expandafter\@gobble
1544   \else
1545     \expandafter\@firstofone
1546   \fi
1547   {\minted@iffasthighlightmode@buffertempfile@iii#2\FV@Sentinel}}
1548 \gdef\minted@iffasthighlightmode@buffertempfile@iii#1.\FV@Sentinel{%
1549   \ifcsname minted@buffertempfileextension@#1\endcsname
1550     \expandafter\@firstofone
1551   \else
1552     \expandafter\@gobble
1553   \fi
1554   {\minted@iffasthighlightmode@buffertempfile@iv}}
1555 \endgroup
1556 \def\minted@iffasthighlightmode@buffertempfile@iv{%
1557   \begingroup
1558   \gdef\minted@tmpcodebufferlength{0}%
1559   \openin\minted@intempfile=\minted@inputfilepath
1560   \endlinechar=-1%
1561   \let\do\@makeother\FVExtraDoSpecials
1562   \loop\unless\ifeof\minted@intempfile
1563     \read\minted@intempfile to\minted@intempfileline

```

```

1564 \xdef\minted@tmpcodebufferlength{\the\numexpr\minted@tmpcodebufferlength+1\relax}%
1565 \expandafter\global\expandafter\let\csname
1566   minted@tmpcodebufferline\minted@tmpcodebufferlength
1567   \endcsname\minted@intempfileline
1568 \repeat
1569 \closein\minted@intempfile
1570 \expandafter\ifx\csname
1571   minted@tmpcodebufferline\minted@tmpcodebufferlength\endcsname\@empty
1572   \expandafter\global\expandafter\let\csname
1573     minted@tmpcodebufferline\minted@tmpcodebufferlength\endcsname\minted@undefined
1574   \xdef\minted@tmpcodebufferlength{\the\numexpr\minted@tmpcodebufferlength-1\relax}%
1575 \fi
1576 \endgroup
1577 \edef\minted@inputfilemdfivesum@check{\pdf@filemdfivesum{\minted@inputfilepath}}%
1578 \ifx\minted@inputfilemdfivesum@check\minted@inputfilemdfivesum
1579   \expandafter\@gobble
1580 \else
1581   \expandafter\@firstofone
1582 \fi
1583 {\VerbatimClearBuffer[buffername=minted@tmpcodebuffer]}%
1584 \ifnum\minted@tmpcodebufferlength>0\relax
1585   \expandafter\@firstofone
1586 \else
1587   \expandafter\@gobble
1588 \fi
1589 {\minted@iffasthighlightmode@buffertempfile@v}%
1590 \def\minted@iffasthighlightmode@buffertempfile@v{%
1591   \pydatabufferkey{code}%
1592   \pydatabuffermlvaluestart
1593   \gdef\minted@tempindex{1}%
1594   \loop\unless\ifnum\minted@tempindex>\minted@tmpcodebufferlength\relax
1595     \expandafter\let\expandafter
1596       \minted@tmp\csname minted@tmpcodebufferline\minted@tempindex\endcsname
1597     \expandafter\pydatabuffermlvalue\expandafter{\minted@tmp}%
1598     \xdef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}%
1599   \repeat
1600   \pydatabuffermlvalueend
1601   \VerbatimClearBuffer[buffername=minted@tmpcodebuffer]}

```

### 11.13 Public API

#### `\setminted`

Set global or lexer-level options.

```

1602 \newcommand{\setminted}[2] [] {%
1603   \ifstrempy{#1}%
1604     {\pgfkeys{/minted/global/.cd,#2}}%
1605     {\let\minted@tmplexer\minted@lexer
1606       \edef\minted@lexer{#1}%
1607       \pgfkeys{/minted/lexer/.cd,#2}%
1608       \let\minted@lexer\minted@tmplexer}}

```

#### `\setmintedinline`

Set global or lexer-level options, but only for inline (`\mintinline`) content. These settings will override the corresponding `\setminted` settings.

```

1609 \newcommand{\setmintedinline}[2] [] {%
1610   \ifstrempy{#1}%
1611   {\pgfkeys{/minted/globalinline/.cd,#2}}%
1612   {\let\minted@tmplexer\minted@lexer
1613    \edef\minted@lexer{#1}%
1614    \pgfkeys{/minted/lexerinline/.cd,#2}%
1615    \let\minted@lexer\minted@tmplexer}}

```

### `\usemintedstyle`

Set style. This is a holdover from minted v1, before `\setminted` could be used to set the style.

```
1616 \newcommand{\usemintedstyle}[2] [] {\setminted[#1]{style={#2}}}
```

### `\mintinline`

Define an inline command. This is modeled after the reimplemented `\Verb` from `fvextra`. See the `fvextra` documentation for details about expansion handling, argument reading, and (re)tokenization.

Everything needs to be within a `\begingroup... \endgroup` to prevent settings from escaping.

`\RobustMintInlineProcess@verbatim` doesn't need an explicit `\FVExtraRetokenizeVArg` step because this is done when the code is inserted into `\Verb`.

```

1617 \def\mintinline{%
1618   \FVExtraRobustCommand\RobustMintInline\FVExtraUnexpandedReadStar0ArgMargBVarArg}
1619 \FVExtraPDFstringdefDisableCommands{%
1620   \def\RobustMintInline{}}
1621 \newrobustcmd{\RobustMintInline}[2] [] {%
1622   \ifbool{FVExtraRobustCommandExpanded}%
1623   {\@ifnextchar\bgroup
1624    {\FVExtraReadVArg{\RobustMintInlineProcess{#1}{#2}}}%
1625    {\minted@error{Inline delimiters must be paired curly braces in this context}}}%
1626   {\FVExtraReadVArg{\RobustMintInlineProcess{#1}{#2}}}}
1627 \def\RobustMintInlineProcess@highlight#1#2#3{%
1628   \begingroup
1629   \booltrue{minted@isinline}%
1630   \ifstrempy{#1}{\pgfkeys{/minted/cmd/.cd,#1}}%
1631   \edef\minted@lexer{#2}%
1632   \minted@usefvopts
1633   \minted@usetexoptsnonpygments
1634   \FVExtraDetokenizeVArg{%
1635     \FVExtraRetokenizeVArg{\RobustMintInlineProcess@highlight@i}{\FV@CatCodes}}{#3}}
1636 \def\RobustMintInlineProcess@highlight@i#1{%
1637   \expandafter\def\csname minted@tmpcodebufferline1\endcsname{#1}%
1638   \gdef\minted@tmpcodebufferlength{1}%
1639   \let\minted@highlight@fallback\RobustMintInlineProcess@highlight@fallback
1640   \minted@highlight
1641   \gdef\minted@tmpcodebufferlength{0}%
1642   \endgroup}
1643 \def\RobustMintInlineProcess@highlight@fallback{%
1644   \minted@useadditionalfvoptsnopy
1645   \fvset{extra=true}%
1646   \minted@codewrapper{%
1647     \expandafter\let\expandafter\minted@tmp\csname minted@tmpcodebufferline1\endcsname
1648     \expandafter\Verb\expandafter{\minted@tmp}}}}
1649 \def\RobustMintInlineProcess@placeholder#1#2#3{%

```

```

1650 \begingroup
1651 \booltrue{minted@isinline}%
1652 \minted@insertplaceholder
1653 \endgroup}
1654 \def\RobustMintInlineProcess@verbatim#1#2#3{%
1655 \begingroup
1656 \booltrue{minted@isinline}%
1657 \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1658 \edef\minted@lexer{#2}%
1659 \minted@usefvopts
1660 \minted@useadditionalfvoptsnopy
1661 \minted@usetexoptsnonpygments
1662 \fvset{extra=true}%
1663 \minted@codewrapper{\Verb{#3}}%
1664 \endgroup}
1665 \ifbool{minted@placeholder}%
1666 {\let\RobustMintInlineProcess\RobustMintInlineProcess@placeholder}%
1667 {\ifbool{minted@verbatim}%
1668 {\let\RobustMintInlineProcess\RobustMintInlineProcess@verbatim}%
1669 {\let\RobustMintInlineProcess\RobustMintInlineProcess@highlight}}

```

`\mint`

Highlight a single line of code. This is essentially a shortcut for the `minted` environment when there is only a single line of code. The implementation follows `\mintinline` for argument reading and processing, but then typesets the code as an environment rather than command. The `\doendpe` ensures proper paragraph indentation for following text (immediately following text with no intervening blank lines does not begin a new paragraph).

```

1670 \def\mint{%
1671 \FVExtraRobustCommand\RobustMint\FVExtraUnexpandedReadStar0ArgMArgBVar}%
1672 \FVExtrapdfstringdefDisableCommands{%
1673 \def\RobustMint{}}
1674 \newrobustcmd{\RobustMint}[2][ ]{%
1675 \ifbool{FVExtraRobustCommandExpanded}%
1676 {\@ifnextchar\bgroup
1677 {\FVExtraReadVArg{\RobustMintProcess{#1}{#2}}}%
1678 {\minted@error{Delimiters must be paired curly braces in this context}}}%
1679 {\FVExtraReadVArg{\RobustMintProcess{#1}{#2}}}}
1680 \def\RobustMintProcess@highlight#1#2#3{%
1681 \begingroup
1682 \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1683 \edef\minted@lexer{#2}%
1684 \minted@usefvopts
1685 \minted@usetexoptsnonpygments
1686 \FVExtraDetokenizeVArg{%
1687 \FVExtraRetokenizeVArg{\RobustMintProcess@highlight@i}{\FV@CatCodes}}{#3}}
1688 \def\RobustMintProcess@highlight@i#1{%
1689 \expandafter\def\csname minted@tmpcodebufferline1\endcsname{#1}%
1690 \gdef\minted@tmpcodebufferlength{1}%
1691 \let\minted@highlight@fallback\RobustMintProcess@highlight@fallback
1692 \minted@highlight
1693 \gdef\minted@tmpcodebufferlength{0}%
1694 \endgroup}
1695 \def\RobustMintProcess@highlight@fallback{%

```

```

1696 \minted@useadditionalfvoptsnopy
1697 \minted@codewrapper{%
1698   \VerbatimInsertBuffer [buffername=minted@tmpcodebuffer , insertenvname=\minted@envname]}}
1699 \def\RobustMintProcess@placeholder#1#2#3{%
1700   \minted@insertplaceholder}
1701 \def\RobustMintProcess@verbatim#1#2#3{%
1702   \begingroup
1703   \ifstrempy{#1}{\pgfkeys{/minted/cmd/.cd,#1}}%
1704   \edef\minted@lexer{#2}%
1705   \minted@usefvopts
1706   \minted@useadditionalfvoptsnopy
1707   \minted@usetexoptsnonpygments
1708   \FVExtraDetokenizeVArg{%
1709     \FVExtraRetokenizeVArg{\RobustMintProcess@verbatim@i}{\FV@CatCodes}}{#3}}
1710 \def\RobustMintProcess@verbatim@i#1{%
1711   \expandafter\def\csname minted@tmpcodebufferline1\endcsname{#1}%
1712   \gdef\minted@tmpcodebufferlength{1}%
1713   \minted@codewrapper{%
1714     \VerbatimInsertBuffer [buffername=minted@tmpcodebuffer , insertenvname=\minted@envname]}}%
1715   \gdef\minted@tmpcodebufferlength{0}%
1716   \endgroup}
1717 \ifbool{minted@placeholder}%
1718   {\let\RobustMintProcess\RobustMintProcess@placeholder}%
1719   {\ifbool{minted@verbatim}%
1720     {\let\RobustMintProcess\RobustMintProcess@verbatim}%
1721     {\let\RobustMintProcess\RobustMintProcess@highlight}}

```

*minted (env.)*

Highlight a longer piece of code inside a verbatim environment.

```

1722 \newenvironment{minted}[2] []%
1723   {\VerbatimEnvironment
1724   \MintedBegin{#1}{#2}}%
1725   {\MintedEnd}
1726 \def\MintedBegin@highlight#1#2{%
1727   \ifstrempy{#1}{\pgfkeys{/minted/cmd/.cd,#1}}%
1728   \edef\minted@lexer{#2}%
1729   \minted@usefvopts
1730   \minted@usetexoptsnonpygments
1731   \begin{VerbatimBuffer} [buffername=minted@tmpcodebuffer , globalbuffer=true]}
1732 \def\MintedEnd@highlight{%
1733   \end{VerbatimBuffer}%
1734   \let\minted@highlight@fallback\MintedEnv@highlight@fallback
1735   \minted@highlight
1736   \VerbatimClearBuffer [buffername=minted@tmpcodebuffer]}
1737 \def\MintedEnv@highlight@fallback{%
1738   \minted@useadditionalfvoptsnopy
1739   \minted@codewrapper{%
1740     \VerbatimInsertBuffer [buffername=minted@tmpcodebuffer , insertenvname=\minted@envname]}}
1741 \def\MintedBegin@placeholder#1#2{%
1742   \begin{VerbatimBuffer} [buffername=minted@tmpcodebuffer]}
1743 \def\MintedEnd@placeholder{%
1744   \end{VerbatimBuffer}%
1745   \minted@insertplaceholder}
1746 \def\MintedBegin@verbatim#1#2{%

```

```

1747 \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1748 \edef\minted@lexer{#2}%
1749 \minted@usefvopts
1750 \minted@useadditionalfvoptsnoppygments
1751 \minted@usetexoptsnonpygments
1752 \begin{\minted@envname}
1753 \def\MintedEnd@verbatim{%
1754 \end{\minted@envname}}
1755 \ifbool{minted@placeholder}%
1756 {\let\MintedBegin\MintedBegin@placeholder
1757 \let\MintedEnd\MintedEnd@placeholder}%
1758 {\ifbool{minted@verbatim}%
1759 {\let\MintedBegin\MintedBegin@verbatim
1760 \let\MintedEnd\MintedEnd@verbatim}%
1761 {\let\MintedBegin\MintedBegin@highlight
1762 \let\MintedEnd\MintedEnd@highlight}}

```

### \inputminted

Highlight an external source file.

```

1763 \def\minted@readinputmintedargs#1#{%
1764 \minted@readinputmintedargs@i{#1}}
1765 \def\minted@readinputmintedargs@i#1#2#3{%
1766 \FVExtraAlwaysUnexpanded{\minted@readinputmintedargs#1{#2}{#3}}
1767 \FVExtrapdfstringdefDisableCommands{%
1768 \makeatletter
1769 \def\minted@readinputmintedargs@i#1#2#3{%
1770 \detokenize{<input from file "#3\detokenize{>}}}%
1771 \makeatother}
1772 \def\inputminted{%
1773 \FVExtraRobustCommand\RobustInputMinted\minted@readinputmintedargs}
1774 \FVExtrapdfstringdefDisableCommands{%
1775 \def\RobustInputMinted{}}
1776 \newrobustcmd{\RobustInputMinted}[3][]{%
1777 \RobustInputMintedProcess{#1}{#2}{#3}}
1778 \def\RobustInputMintedProcess@highlight#1#2#3{%
1779 \begingroup
1780 \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1781 \edef\minted@lexer{#2}%
1782 \edef\minted@inputfilepath{#3}%
1783 \minted@usefvopts
1784 \minted@usetexoptsnonpygments
1785 \let\minted@highlight@fallback\RobustInputMintedProcess@highlight@fallback
1786 \minted@highlightinputfile
1787 \endgroup}
1788 \def\RobustInputMintedProcess@highlight@fallback{%
1789 \minted@useadditionalfvoptsnoppygments
1790 \minted@codewrapper{%
1791 \csname\minted@envname Input\endcsname{\minted@inputfilepath}}}
1792 \def\RobustInputMintedProcess@placeholder#1#2#3{%
1793 \minted@insertplaceholder}
1794 \def\RobustInputMintedProcess@verbatim#1#2#3{%
1795 \begingroup
1796 \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1797 \edef\minted@lexer{#2}%

```



```

1798 \edef\minted@inputfilepath{#3}%
1799 \minted@usefvopts
1800 \minted@useadditionalfvoptsnopy
1801 \minted@usetexoptsnonpygments
1802 \minted@codewrapper{%
1803   \csname\minted@envname Input\endcsname{\minted@inputfilepath}}%
1804 \endgroup}
1805 \ifbool{minted@placeholder}%
1806 {\let\RobustInputMintedProcess\RobustInputMintedProcess@placeholder}%
1807 {\ifbool{minted@verbatim}%
1808   {\let\RobustInputMintedProcess\RobustInputMintedProcess@verbatim}%
1809   {\let\RobustInputMintedProcess\RobustInputMintedProcess@highlight}}

```

## 11.14 Command shortcuts

Allow the user to define shortcuts for the highlighting commands.

`\newminted`

Define a new language-specific alias for the `minted` environment.

The starred `*` version of the environment takes a mandatory argument containing options. It is retained for backward compatibility purposes with `minted v1` and `v2`. `minted v3` added support for an optional argument to the standard environment, so the starred version is no longer necessary.

The `^M` is needed because `\FVExtraRead0ArgBeforeVEnv` strips a following `^M` (basically the newline), but `fancyvrb` environments expect `^M` before the start of environment contents.

```

1810 \newcommand{\newminted}[3] [] {%
1811   \ifstrempy{#1}%
1812     {\newminted@i{#2code}{#2}{#3}}%
1813     {\newminted@i{#1}{#2}{#3}}
1814 \begingroup
1815 \catcode\^M=\active%
1816 \gdef\newminted@i#1#2#3{%
1817   \expandafter\def\csname#1i\endcsname##1{%
1818     \begin{minted}[#3,##1]{#2}^M}%
1819   \newenvironment{#1}%
1820     {\VerbatimEnvironment%
1821      \FVExtraRead0ArgBeforeVEnv{\csname#1i\endcsname}}%
1822     {\end{minted}}%
1823   \newenvironment{#1*}[1]%
1824     {\VerbatimEnvironment%
1825      \begin{minted}[#3,##1]{#2}}%
1826     {\end{minted}}%
1827 \endgroup

```

`\newmint`

Define a new language-specific alias for the `\mint` short form.

```

1828 \newcommand{\newmint}[3] [] {%
1829   \ifstrempy{#1}%
1830     {\edef\minted@tmp{#2}}%
1831     {\edef\minted@tmp{#1}}
1832   \expandafter\newmint@i\expandafter{\minted@tmp}{#2}{#3}
1833 \def\newmint@i#1#2#3{%
1834   \expandafter\newcommand\csname#1\endcsname{%

```

```

1835 \expandafter\FVExtraRobustCommand\csname RobustNewMint#1\endcsname
1836 \FVExtraUnexpandedReadStar0ArgBVar}%
1837 \FVExtrapdfstringdefDisableCommands{%
1838 \expandafter\def\csname RobustNewMint#1\endcsname{}}%
1839 \expandafter\newrobustcmd\csname RobustNewMint#1\endcsname{%
1840 \FVExtraRead0ArgBeforeVArg{\csname RobustNewMint#1@i\endcsname}}%
1841 \expandafter\def\csname RobustNewMint#1@i\endcsname##1{%
1842 \ifbool{FVExtraRobustCommandExpanded}%
1843 {\@ifnextchar\bgroup
1844 {\FVExtraReadVArg{\csname RobustNewMint#1@ii\endcsname{##1}}}%
1845 {\minted@error{Delimiters must be paired curly braces in this context}}}%
1846 {\FVExtraReadVArg{\csname RobustNewMint#1@ii\endcsname{##1}}}}%
1847 \expandafter\def\csname RobustNewMint#1@ii\endcsname##1##2{%
1848 \RobustMintProcess{#3,##1}{#2}{##2}}

```

#### \newmintedfile

Define a new language-specific alias for \inputminted.

```

1849 \def\minted@readnewmintedfileargs#1{%
1850 \minted@readnewmintedfileargs@i{#1}}
1851 \def\minted@readnewmintedfileargs@i#1#2{%
1852 \FVExtraAlwaysUnexpanded{\minted@readnewmintedfileargs#1{#2}}
1853 \FVExtrapdfstringdefDisableCommands{%
1854 \makeatletter
1855 \def\minted@readnewmintedfileargs@i#1#2{%
1856 \detokenize{<input from file ">#2\detokenize{>}}%
1857 \makeatother}
1858 \newcommand{\newmintedfile}[3][[]]{%
1859 \ifstrempy{#1}%
1860 {\edef\minted@tmp{#2file}}%
1861 {\edef\minted@tmp{#1}}%
1862 \expandafter\newmintedfile@i\expandafter{\minted@tmp}{#2}{#3}}
1863 \def\newmintedfile@i#1#2#3{%
1864 \expandafter\newcommand\csname#1\endcsname{%
1865 \expandafter\FVExtraRobustCommand\csname RobustNewMintedFile#1\endcsname
1866 \minted@readnewmintedfileargs}%
1867 \FVExtrapdfstringdefDisableCommands{%
1868 \expandafter\def\csname RobustNewMintedFile#1\endcsname{}}%
1869 \expandafter\newrobustcmd\csname RobustNewMintedFile#1\endcsname[2][[]]{%
1870 \RobustInputMintedProcess{#3,##1}{#2}{##2}}

```

#### \newmintinline

Define an alias for \mintinline.

```

1871 \newcommand{\newmintinline}[3][[]]{%
1872 \ifstrempy{#1}%
1873 {\edef\minted@tmp{#2inline}}%
1874 {\edef\minted@tmp{#1}}%
1875 \expandafter\newmintinline@i\expandafter{\minted@tmp}{#2}{#3}}
1876 \def\newmintinline@i#1#2#3{%
1877 \expandafter\newcommand\csname#1\endcsname{%
1878 \expandafter\FVExtraRobustCommand\csname RobustNewMintInline#1\endcsname
1879 \FVExtraUnexpandedReadStar0ArgBVar}%
1880 \FVExtrapdfstringdefDisableCommands{%
1881 \expandafter\def\csname RobustNewMintInline#1\endcsname{}}%
1882 \expandafter\newrobustcmd\csname RobustNewMintInline#1\endcsname{%
1883 \FVExtraRead0ArgBeforeVArg{\csname RobustNewMintInline#1@i\endcsname}}%

```

```

1884 \expandafter\def\csname RobustNewMintInline#1@i\endcsname##1{%
1885   \ifbool{FVExtraRobustCommandExpanded}%
1886     {\@ifnextchar\bgroup
1887       {FVExtraReadVArg{\csname RobustNewMintInline#1@ii\endcsname{##1}}}%
1888       {\minted@error{Inline delimiters must be paired curly braces in this context}}}%
1889     {FVExtraReadVArg{\csname RobustNewMintInline#1@i\endcsname{##1}}}}
1890 \expandafter\def\csname RobustNewMintInline#1@ii\endcsname##1##2{%
1891   \RobustMintInlineProcess{#3,##1}{#2}{##2}}

```

## 11.15 Float support

`listing` (*env.*)

Define a new floating environment to use for floated listings. This is defined conditionally based on the `newfloat` package option.

```

1892 \ifbool{minted@newfloat}%
1893   {\@ifundefined{minted@float@within}%
1894     {\DeclareFloatingEnvironment[fileext=lol,placement=tbp]{listing}}%
1895     {\def\minted@tmp#1{%
1896       \DeclareFloatingEnvironment[fileext=lol,placement=tbp,within=#1]{listing}}%
1897       \expandafter\minted@tmp\expandafter{\minted@float@within}}}%
1898   {\@ifundefined{minted@float@within}%
1899     {\newfloat{listing}{tbp}{lol}}%
1900     {\newfloat{listing}{tbp}{lol}[\minted@float@within]}}

```

The following macros only apply when `listing` is created with the `float` package. When `listing` is created with `newfloat`, its properties should be modified using `newfloat`'s `\SetupFloatingEnvironment`.

```

1901 \ifminted@newfloat\else

```

`\listingcaption`

The name that is displayed before each individual listings caption and its number. The macro `\listingcaption` can be redefined by the user.

```

1902 \newcommand{\listingcaption}{Listing}

```

The following definition should not be changed by the user.

```

1903 \floatname{listing}{\listingcaption}

```

`\listoflistingscaption`

The caption that is displayed for the list of listings.

```

1904 \newcommand{\listoflistingscaption}{List of Listings}

```

`\listoflistings`

Used to produce a list of listings (like `\listoffigures` etc.). This may well clash with other packages (for example, `listings`) but we choose to ignore this since these two packages shouldn't be used together in the first place.

```

1905 \providecommand{\listoflistings}{\listof{listing}{\listoflistingscaption}}

```

Again, the preceding macros only apply when `float` is used to create listings, so we need to end the conditional.

```

1906 \fi

```