

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

v3.1.1 from 2024/10/03

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

1	Introduction	4
2	Installation	4
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 _E X installation	5
3	Migrating from minted version 2	6
4	Basic usage	7
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
5	Floating listings	10
6	Configuration	12
6.1	minted config file <code>.latexminted_config</code>	12
6.2	macOS compatibility	14
7	Options	14
7.1	Package options	14
7.2	Setting options for commands and environments	17
7.3	Command and environment options	18
8	Defining shortcuts	32
9	FAQ and Troubleshooting	33
10	Acknowledgements	36
11	Implementation	37
11.1	Required packages	37
11.2	Exception handling	37
11.3	Python executable and minimum supported version	37
11.4	Timestamp	38
11.5	Jobname MD5 and derived file names	39
11.6	Package options	40
11.6.1	Package option definitions	40
11.6.2	Package options that are no longer supported or deprecated	43
11.6.3	Package option processing	44
11.7	Util	44
11.7.1	Check whether a string matches the regex <code>^[0-9A-Za-z_-]+\$</code>	45
11.8	State	46
11.9	Calling minted executable	47
11.10	Config detection	48

11.11	Options	51
11.11.1	Option processing	51
11.11.2	Option handlers	58
11.11.3	Option definitions	59
11.12	Caching, styles, and highlighting	62
11.12.1	Cache management	62
11.12.2	Style definitions	64
11.12.3	Lexer-specific line numbering	68
11.12.4	Highlighting code	69
11.13	Public API	74
11.14	Command shortcuts	78
11.15	Float support	80

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 \TeX for inclusion in the document. Here is an example with Ruby code, showing the \TeX 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 \TeX . 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 \TeX 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 \TeX 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. There is also 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 or 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 \TeX 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 \TeX 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 \TeX format.

Currently, `latexminted` requires special permission to run. \TeX must be called 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.

`latexminted` is designed to be compatible with the security requirements for restricted shell escape. Once `latexminted` finishes the security review for restricted shell escape executables, it will function automatically without `-shell-escape` or `-enable-write18`, so long as the default restricted shell escape has not been disabled. It is possible to benefit from these enhanced security capabilities immediately and avoid the need for `-shell-escape` or `-enable-write18` by manually designating `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 (env.)` 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>}<delim><code><delim>`, 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>}<delim><code><delim>`. 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. It is possible to preview your code with different styles using the online demo at 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/. You can also run `pygmentize -L lexers`.

5 Floating listings

`listing (env.)` `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:

```
\listoflistings
```

List of Listings

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>` `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
(default: `_minted`) should use forward slashes, even under Windows. Special characters must be escaped with `\string` or `\detokenize`.

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
(default: `false`) used in generating highlighted code and also writes additional information to the log.
`frozenscache=<boolean>` Use a frozen (static) cache. When `frozenscache=true`, Python and Pygments are
(default: `false`) 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.

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
(default: `false`) commands based on it) in addition to `minted` environments and `\mint` commands (and custom environments/commands based on them).

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 environ-
(default: `false`) ments/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.

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 <iostream> int main() { std::cout << "Hello " << "world" << std::endl; } \end{minted}</pre>	<pre>1 #include <iostream> 2 int main() { 3 std::cout << "Hello " 4 << "world" 5 << 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 $\sum_{i=1}^n i$ def sum_from_one_to(n): r = range(1, n + 1) return sum(r) \end{minted}</pre>	<pre># Returns $\sum_{i=1}^n i$ 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 \TeX 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 \TeX , 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:

```
\definecolor{bg}{rgb}{.9, .9, .9}
\begin{minted}[bgcolor=bg]{php}
<?php
    echo "Hello, $x";
?>
\end{minted}
```

```
<?php
    echo "Hello, $x";
?>
```

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)
 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, \hboxfuzz 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)
 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{_ \rfloor}, _)
 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. 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 \TeX 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 `breaksymbolsepriht`. 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][\colorbox]{#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}`, `\rightarrow`)
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>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: <code><doc encoding></code>)
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: <code><none></code>)
A format to execute before printing verbatim text.		
<code>frame</code>	(none leftline topline bottomline lines single)	(default: none)
The type of frame to put around the source code listing. For more sophisticated framing, consider <code>tcolorbox</code> .		
<code>framerule</code>	(dimension)	(default: 0.4pt)
Width of the frame.		
<code>framesep</code>	(dimension)	(default: <code>\fboxsep</code>)
Distance between frame and content.		
<code>funcnamehighlighting</code>	(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

```

- listparameters** (macro) (default: *<empty>*)
fancyvrb option for setting list-related lengths to modify spacing around lines of code. For example, listparameters=\setlength{\topsep}{0pt} will remove space before and after a minted environment.
- literalenvname** (string) (default: *MintedVerbatim*)
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 envname. There should be

few if any situations where modifying `literalenvname` rather than `envname` is actually necessary.

`literatecomment` (single macro or backslash-escaped string) (default: `<none>`)
This is for compatibility with literate programming formats, such as the `.dtx` format commonly used for writing \LaTeX packages. If all lines of code begin with `<literatecomment>`, then `<literatecomment>` is removed from the beginning of all lines. For example, for `.dtx`, `literatecomment=\%`.

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).

`mathescape` (boolean) (default: `false`)
Enable \LaTeX math mode inside comments. Usage as in package listings. See the note under `escapeinside` regarding math and ligatures.

`numberblanklines` (boolean) (default: `true`)
Enables or disables numbering of blank lines.

`numberfirstline` (boolean) (default: `false`)
Always number the first line, regardless of `stepnumber`.

`numbers` (left | right | both | none) (default: `none`)
Essentially the same as `linenos`, except the side on which the numbers appear may be specified.

`numbersep` (dimension) (default: `12pt`)
Gap between numbers and start of line.

`obeytabs` (boolean) (default: `false`)
Treat tabs as tabs instead of converting them to spaces—that is, expand them to tab stops determined by `tabsize`. **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.**

`python3` (boolean) (default: `true`)
[For `PythonConsoleLexer` only] Specifies whether Python 3 highlighting is applied.

`rangeregex` (single macro or backslash-escaped string) (default: `<none>`)
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 (“`\`” 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.

`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.

<code>rangeregexmultiline</code>	(boolean) (default: <code>false</code>) “~” and “\$” match at internal newlines, not just at the start/end of the string.
<code>rangestartafterstring</code>	(single macro or backslash-escaped string) (default: <code><none></code>) 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.
<code>rangestartstring</code>	(single macro or backslash-escaped string) (default: <code><none></code>) 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.
<code>rangestopbeforestring</code>	(single macro or backslash-escaped string) (default: <code><none></code>) 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.
<code>rangestopstring</code>	(single macro or backslash-escaped string) (default: <code><none></code>) 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.
<code>resetmargins</code>	(boolean) (default: <code>false</code>) Resets the left margin inside other environments.
<code>rulecolor</code>	(color command) (default: <code>black</code>) The color of the frame.
<code>samepage</code>	(boolean) (default: <code>false</code>) Forces the whole listing to appear on the same page, even if it doesn't fit.

<code>showspaces</code>	(boolean)	(default: false)
	Enables visible spaces: <code>visible_spaces</code> .	
<code>showtabs</code>	(boolean)	(default: false)
	Enables visible tabs—only works in combination with <code>obeytabs</code> .	
<code>space</code>	(macro)	(default: <code>\textvisiblespace</code> , <code>␣</code>)
	Redefine the visible space character. Note that this is only used if <code>showspaces=true</code> .	
<code>spacecolor</code>	(string)	(default: none)
	Set the color of visible spaces. By default (none), they take the color of their surroundings.	
<code>startinline</code>	(boolean)	(default: false)
	[For PHP only] Specifies that the code starts in PHP mode, i.e., leading <code><?php</code> is omitted.	
<code>stepnumber</code>	(integer)	(default: 1)
	Interval at which line numbers appear.	
<code>stepnumberfromfirst</code>	(boolean)	(default: false)
	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.	
<code>stepnumberoffsetvalues</code>	(boolean)	(default: false)
	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.	
<code>stripall</code>	(boolean)	(default: false)
	Strip all leading and trailing whitespace from the input.	
<code>stripnl</code>	(boolean)	(default: false)
	Strip leading and trailing newlines from the input.	
<code>style</code>	(string)	(default: <code><default></code>)
	Sets the stylesheet used by Pygments.	
<code>tab</code>	(macro)	(default: fancyvrb's <code>\FancyVerbTab</code> , <code>↵</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: black)
	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: false)
Enables \LaTeX code inside comments. Usage as in package listings. See the note under <code>escapeinside</code> regarding math and ligatures.		
<code>texcomments</code>	(boolean)	(default: false)
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 tokenized as <code>Comment.Preproc</code> , so <code>texcomments</code> causes preprocessor directives to be treated as literal \LaTeX code. An issue has been opened at the Pygments site; additional details are also available on the minted GitHub site .		
<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 <typename T> T id(T value) { return value; } \end{cppcode}</pre>	<pre>1 template <typename T> 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.

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

`\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.

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

`\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 L^AT_EX or the PDF format. Common issues are described below, with workarounds or solutions. You may also wish to search 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 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@addtolists.** `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}[2024/09/05]
4 \RequirePackage{latex2pydata}[2024/05/16]
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 \TeX 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 \TeX .

`\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 \TeX itself, so using `\jobname` would work correctly in most cases. However, when full shell escape is enabled, no command escaping is performed by \TeX , 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=minted@verbatim,
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 `\AfterEndDocument` with `\minted@clean`; the `\AtEndDocument` is created with the definition of `\minted@clean` so that everything is in the correct order.

```

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@tmpcnt`

Temp counter.

```
263 \newcounter{minted@tmpcnt}
```

`\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 \LaTeX side.

`\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 \setcounter{minted@tmpcnt}{48}
291 \loop\unless\ifnum\value{minted@tmpcnt}>57\relax
292 \expandafter\let\csname minted@isnum\arabic{minted@tmpcnt}\endcsname\relax
293 \expandafter\let\csname minted@isalphanum\arabic{minted@tmpcnt}\endcsname\relax
294 \expandafter\let
295 \csname minted@isalphanumhyphenunderscore\arabic{minted@tmpcnt}\endcsname\relax
296 \stepcounter{minted@tmpcnt}
297 \repeat
298 % [A-Z]
299 \setcounter{minted@tmpcnt}{65}
300 \loop\unless\ifnum\value{minted@tmpcnt}>90\relax
301 \expandafter\let\csname minted@isalpha\arabic{minted@tmpcnt}\endcsname\relax
302 \expandafter\let\csname minted@isalphanum\arabic{minted@tmpcnt}\endcsname\relax
303 \expandafter\let
304 \csname minted@isalphanumhyphenunderscore\arabic{minted@tmpcnt}\endcsname\relax
305 \stepcounter{minted@tmpcnt}

```

```

306 \repeat
307 % [a-z]
308 \setcounter{minted@tmpcnt}{97}
309 \loop\unless\ifnum\value{minted@tmpcnt}>122\relax
310 \expandafter\let\csname minted@isalpha\arabic{minted@tmpcnt}\endcsname\relax
311 \expandafter\let\csname minted@isalphanum\arabic{minted@tmpcnt}\endcsname\relax
312 \expandafter\let
313 \csname minted@isalphanumhyphenunderscore\arabic{minted@tmpcnt}\endcsname\relax
314 \stepcounter{minted@tmpcnt}
315 \repeat
316 % [-]
317 \expandafter\let\csname minted@isalphanumhyphenunderscore45\endcsname\relax
318 % [_]
319 \expandafter\let\csname minted@isalphanumhyphenunderscore95\endcsname\relax
\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 \newcounter{minted@tmpcodebufferlength}
```

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@frozensource}{\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 \let\minted@exec@warning\relax
364 \let\minted@exec@error\relax
365 \ifbool{minted@canexec}{\ShellEscape{#1}\minted@inputexecmessages}{}
366 \def\minted@ShellEscapeNoMessages#1{%
367 \ifbool{minted@canexec}{\ShellEscape{#1}}{}}

\minted@execarg@debug
\minted@execarg@timestamp
368 \def\minted@execarg@debug{%
369 \ifbool{minted@debug}{\detokenize{ --debug }}{}}
370 \def\minted@execarg@timestamp{%
371 \detokenize{ --timestamp }\minted@timestamp\detokenize{ }}

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

\minted@exec@batch
    Run in batch mode, for highlightmode=fast or highlightmode=fastfirst.
374 \def\minted@exec@batch{%
375 \minted@ShellEscapeMaybeMessages{%
376 \MintedExecutable
377 \detokenize{ batch }\minted@execarg@timestamp\minted@execarg@debug
378 \MintedJobnameMdfive}}
```

```

\minted@exec@config
    Detect configuration.
379 \def\minted@exec@config{%
380 \minted@ShellEscapeMaybeMessages{%
381 \MintedExecutable
382 \detokenize{ config }\minted@execarg@timestamp\minted@execarg@debug
383 \MintedJobnameMdfive}}

\minted@exec@styledef
    Create style definition.
384 \def\minted@exec@styledef{%
385 \minted@ShellEscapeMaybeMessages{%
386 \MintedExecutable
387 \detokenize{ styledef }\minted@execarg@timestamp\minted@execarg@debug
388 \MintedJobnameMdfive}}

\minted@exec@highlight
    Highlight code.
389 \def\minted@exec@highlight{%
390 \minted@ShellEscapeMaybeMessages{%
391 \MintedExecutable
392 \detokenize{ highlight }\minted@execarg@timestamp\minted@execarg@debug
393 \MintedJobnameMdfive}}

\minted@exec@clean
    Clean temp files and cache.
394 \def\minted@exec@clean{%
395 \minted@ShellEscapeNoMessages{%
396 \MintedExecutable
397 \detokenize{ clean }\minted@execarg@timestamp\minted@execarg@debug
398 \MintedJobnameMdfive}}

\minted@exec@cleanconfig
    Clean config temp file.
399 \def\minted@exec@cleanconfig{%
400 \minted@ShellEscapeNoMessages{%
401 \MintedExecutable
402 \detokenize{ cleanconfig }\minted@execarg@timestamp\minted@execarg@debug
403 \MintedJobnameMdfive}}

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

```

11.10 Config detection

```

\minted@diddetectconfig
409 \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 `TeX` 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 `TeX` 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.

```
410 \def\minted@detectconfig{%
411   \ifbool{minted@diddetectconfig}%
412     {}%
413     {\ifx\minted@cachedir\@empty
414       \gdef\minted@cachepath{}%
415       \else
416         \gdef\minted@cachepath{\minted@cachedir/}%
417       \fi
418       \ifbool{minted@canexec}{\begingroup\minted@detectconfig@i\endgroup}{}%
419       \global\booltrue{minted@diddetectconfig}}
420 \def\minted@detectconfig@i{%
421   \global\let\minted@executable@version\relax
422   \global\let\minted@executable@timestamp\relax
423   \global\let\minted@config@timestamp\relax
424   \pydatasetfilename{\MintedDataFilename}%
425   \pydatawritedictopen
426   \pydatawritekeyvalue{command}{config}%
427   \pydatawritekeyedefvalue{jobname}{\jobname}%
428   \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
429   \pydatawritekeyedefvalue{cachedir}{\minted@cachedir}%
430   \pydatawritedictclose
431   \pydataclosefilename{\MintedDataFilename}%
432   \minted@exec@config
433   \minted@ensureatletter{%
434     \InputIfFileExists{\MintedConfigFilename}{\relax}{\relax}%
435   }
436   \ifx\minted@executable@version\relax
437     \expandafter\minted@detectconfig@noexecutable
438   \else
439     \expandafter\minted@detectconfig@ii
440   \fi
441 }
442 \def\minted@detectconfig@noexecutable{%
443   \global\boolfalse{minted@canexec}%
444   \ifnum\c_sys_shell_escape_int\endcsname=1\relax
445     \minted@error{minted v3+ executable is not installed or is not added to PATH}%
446   \else
447     \minted@error{minted v3+ executable is not installed, is not added to PATH,
448       or is not permitted with restricted shell escape}%
449   \fi
450 }
451 \def\minted@detectconfig@ii{%
452   \ifx\minted@timestamp\minted@config@timestamp
```

```

450 \expandafter\minted@detectconfig@iii
451 \else
452 \expandafter\minted@detectconfig@wrongtimestamp
453 \fi}
454 \def\minted@detectconfig@wrongtimestamp{%
455 \ifx\minted@timestamp\minted@executable@timestamp
456 \minted@exec@cleanconfig
457 \global\boolfalse{minted@canexec}%
458 \minted@error{minted v3 Python executable could not find output directory;
459 upgrade to TeX distribution that supports \detokenize{TEXMF_OUTPUT_DIRECTORY}
460 or set environment variable \detokenize{TEXMF_OUTPUT_DIRECTORY} manually}}%
461 \else
462 \expandafter\minted@detectconfig@noexecutable
463 \fi}
464 \def\minted@detectconfig@iii{%
465 \minted@exec@cleanconfig
466 \ifx\minted@exec@warning\relax
467 \else
468 \expandafter\minted@exec@warning
469 \fi
470 \ifx\minted@exec@error\relax
471 \expandafter\minted@detectconfig@iv
472 \else
473 \expandafter\minted@detectconfig@error
474 \fi}
475 \def\minted@detectconfig@error{%
476 \global\boolfalse{minted@canexec}%
477 \minted@exec@error}
478 \def\minted@detectconfig@iv{%
479 \expandafter\minted@detectconfig@v\minted@executable@version\relax}
480 \begingroup
481 \catcode\.=12
482 \gdef\minted@detectconfig@v#1.#2.#3\relax{%
483 \def\minted@executable@major{#1}%
484 \def\minted@executable@minor{#2}%
485 \def\minted@executable@patch{#3}%
486 \minted@detectconfig@vi}
487 \endgroup
488 \def\minted@detectconfig@vi{%
489 \ifnum\minted@executable@major>\minted@executable@minmajor\relax
490 \global\booltrue{minted@executable@issupported}%
491 \else\ifnum\minted@executable@major=\minted@executable@minmajor\relax
492 \ifnum\minted@executable@minor>\minted@executable@minminor\relax
493 \global\booltrue{minted@executable@issupported}%
494 \else\ifnum\minted@executable@minor=\minted@executable@minminor\relax
495 \ifnum\minted@executable@patch<\minted@executable@minpatch\relax
496 \else
497 \global\booltrue{minted@executable@issupported}%
498 \fi
499 \fi\fi
500 \fi\fi
501 \ifbool{minted@executable@issupported}%
502 {\ifx\minted@config@cachepath\relax
503 \expandafter\@firstoftwo

```

```

504 \else
505 \expandafter\@secondoftwo
506 \fi
507 {\global\boolfalse{minted@canexec}%
508 \minted@error{minted Python executable returned incomplete configuration data;
509 this may indicate a bug in minted or file corruption}}%
510 {\global\let\minted@cachepath\minted@config@cachepath
511 \minted@highlightmode@init}}%
512 {\global\boolfalse{minted@canexec}%
513 \minted@error{minted Python executable is version \minted@executable@version,
514 but version \minted@executable@minversion+ is required}}}%

```

11.11 Options

11.11.1 Option processing

`\minted@optcats`

`\minted@optkeylist@<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.

```

515 \begingroup
516 \catcode`\,=12
517 \gdef\minted@optcats{fv,py,tex}
518 \endgroup
519 \def\minted@do#1{\expandafter\def\csname minted@optkeylist@#1\endcsname{}}
520 \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.

```

521 \begingroup
522 \catcode`\,=12
523 \gdef\minted@optscopes{global,lexer,globalinline,lexerinline,cmd}
524 \gdef\minted@optscopes@onlyblock{global,lexer,cmd}
525 \endgroup

```

`\minted@iflexerscope`

```

526 \let\minted@iflexerscope@lexer\relax
527 \let\minted@iflexerscope@lexerinline\relax
528 \def\minted@iflexerscope#1#2#3{%
529 \ifcsname minted@iflexerscope@#1\endcsname
530 \expandafter\@firstoftwo
531 \else
532 \expandafter\@secondoftwo
533 \fi
534 {#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^AT_EX 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.

```
535 \def\minted@adoptkey#1#2{%
536   \ifcsname minted@optkeyslist@#1\endcsname
537   \else
538     \minted@fatalerror{Defining options under category "#1" is not supported}%
539   \fi
540   \expandafter\let\expandafter\minted@tmp\csname minted@optkeyslist@#1\endcsname
541   \ifx\minted@tmp\@empty
542     \def\minted@tmp{#2}%
543   \else
544     \expandafter\def\expandafter\minted@tmp\expandafter{\minted@tmp,#2}%
545   \fi
546   \expandafter\let\csname minted@optkeyslist@#1\endcsname\minted@tmp}
547 \newcommand*\mintedpgfkeyscreate}[3][]{%
548   \mintedpgfkeyscreate@i{#1}{#2}{#3}}
549 \begingroup
550 \catcode`\==12
551 \gdef\mintedpgfkeyscreate@i#1#2#3{%
552   \def\minted@do##1{%
```

```

553 \minted@do@i##1=\FV@Sentinel}%
554 \def\minted@do@i##1=##2\FV@Sentinel{%
555 \minted@do@ii##1<>\FV@Sentinel}%
556 \def\minted@do@ii##1<##2>##3\FV@Sentinel{%
557 \minted@adoptkey{#2}{##1}}%
558 \minted@forcsvlist{\minted@do}{#3}%
559 \csname minted@pgfkeyscreate@#2\endcsname{#1}{#3}}
560 \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.

```

561 \def\minted@pgfkeyscreate@fv#1#2{%
562 \if\relax\detokenize{#1}\relax
563 \else
564 \minted@fatalerror{Processor macros are not supported in defining fancyvrb options}%
565 \fi
566 \minted@forcsvlist{\minted@pgfkeycreate@fv}{#2}}
567 \begingroup
568 \catcode`\==12
569 \gdef\minted@pgfkeycreate@fv#1{%
570 \minted@pgfkeycreate@fv@i#1=\FV@Sentinel}
571 \gdef\minted@pgfkeycreate@fv@i#1=##2\FV@Sentinel{%
572 \if\relax\detokenize{#2}\relax
573 \expandafter\minted@pgfkeycreate@fv@ii
574 \else
575 \expandafter\minted@pgfkeycreate@fv@iii
576 \fi
577 {#1}##2\FV@Sentinel}
578 \gdef\minted@pgfkeycreate@fv@ii#1\FV@Sentinel{%
579 \minted@pgfkeycreate@fv@iv{#1}{\minted@const@pgfkeysnovalue}}
580 \gdef\minted@pgfkeycreate@fv@iii#1#2=\FV@Sentinel{%
581 \minted@pgfkeycreate@fv@iv{#1}{#2}}
582 \endgroup
583 \def\minted@pgfkeycreate@fv@iv#1#2{%
584 \def\minted@do##1{%
585 \minted@iflexerscope{##1}%
586 {\minted@do@i{##1}{\minted@lexer}}}%
587 {\minted@do@i{##1}{}}}%
588 \def\minted@do@i##1##2{%
589 \pgfkeys{%
590 /minted/##1/.cd,

```

```

591     #1/.code={%
592     \def\minted@tmp{####1}%
593     \ifx\minted@tmp\minted@const@pgfkeysnovalue
594     \begingroup\fvset{#1}\endgroup
595     \minted@apptoprovidecs{\minted@fvoptlist@##1##2}{#1,}%
596     \else
597     \begingroup\fvset{#1={####1}}\endgroup
598     \minted@apptoprovidecs{\minted@fvoptlist@##1##2}{#1={####1}},}%
599     \fi},
600 }%
601 }%
602 \minted@forcsvlist{\minted@do}{\minted@optscopes}%
603 \ifx\minted@const@pgfkeysnovalue#2\relax
604 \else
605     \pgfkeys{%
606         /minted/global/.cd,
607         #1={#2},
608     }%
609 \fi}
610 \def\minted@usefvopts{%
611     \ifbool{\minted@isinline}%
612     {\minted@forcsvlist{\minted@usefvopts@do}{\minted@optscopes}}%
613     {\minted@forcsvlist{\minted@usefvopts@do}{\minted@optscopes@onlyblock}}%
614 \def\minted@usefvopts@do#1{%
615     \minted@iflexerscope{#1}%
616     {\ifcsname \minted@fvoptlist@#1\minted@lexer\endcsname
617       \expandafter
618       \let\expandafter\minted@tmp\csname \minted@fvoptlist@#1\minted@lexer\endcsname
619       \expandafter\fvset\expandafter{\minted@tmp}%
620       \fi}%
621     {\ifcsname \minted@fvoptlist@#1\endcsname
622       \expandafter
623       \let\expandafter\minted@tmp\csname \minted@fvoptlist@#1\endcsname
624       \expandafter\fvset\expandafter{\minted@tmp}%
625       \fi}}
626 \def\minted@useadditionalfvoptsnopy{%
627     \edef\minted@tmp{\mintedpyoptvalueof{gobble}}%
628     \ifx\minted@tmp\minted@const@pgfkeysnovalue
629     \else
630     \expandafter\minted@useadditionalfvoptsnopy@fvsetvk
631     \expandafter{\minted@tmp}{gobble}%
632     \fi
633     \edef\minted@tmp{\mintedpyoptvalueof{mathescape}}%
634     \ifx\minted@tmp\minted@const@pgfkeysnovalue
635     \else
636     \expandafter\minted@useadditionalfvoptsnopy@fvsetvk
637     \expandafter{\minted@tmp}{mathescape}%
638     \fi}
639 \def\minted@useadditionalfvoptsnopy@fvsetvk#1#2{%
640     \fvset{#2={#1}}}

\minted@pgfkeyscreate@py
\mintedpyoptvalueof
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`.

```

641 \def\minted@pgfkeyscreate@py#1#2{%
642   \minted@forcsvlist{\minted@pgfkeycreate@py{#1}}{#2}}
643 \begingroup
644 \catcode`\==12
645 \catcode`\<=12
646 \catcode`\>=12
647 \gdef\minted@pgfkeycreate@py#1#2{%
648   \minted@pgfkeycreate@py@i{#1}#2=\FV@Sentinel}
649 \gdef\minted@pgfkeycreate@py@i#1#2=#3\FV@Sentinel{%
650   \if\relax\detokenize{#3}\relax
651     \expandafter\minted@pgfkeycreate@py@ii
652   \else
653     \expandafter\minted@pgfkeycreate@py@iii
654   \fi
655   {#1}{#2}#3\FV@Sentinel}
656 \gdef\minted@pgfkeycreate@py@ii#1#2\FV@Sentinel{%
657   \minted@pgfkeycreate@py@iv{#1}{\pgfkeysnovalue}#2<>\FV@Sentinel}
658 \gdef\minted@pgfkeycreate@py@iii#1#2#3=\FV@Sentinel{%
659   \minted@pgfkeycreate@py@iv{#1}{#3}#2<>\FV@Sentinel}
660 \gdef\minted@pgfkeycreate@py@iv#1#2#3<#4>#5\FV@Sentinel{%
661   \if\relax\detokenize{#4}\relax
662     \expandafter\@firstoftwo
663   \else
664     \expandafter\@secondoftwo
665   \fi
666   {\minted@pgfkeycreate@py@v{#1}{#3}{#2}{\minted@const@pgfkeysnovalue}}}%
667   {\minted@pgfkeycreate@py@v{#1}{#3}{#2}{#4}}}%
668 \endgroup
669 \def\minted@pgfkeycreate@py@v#1#2#3#4{%
670   \def\minted@do##1{%
671     \minted@iflexerscope{##1}%
672     {\minted@do@i{##1}{\@minted@lexer}}%
673     {\minted@do@i{##1}{}}}
674   \def\minted@do@i##1##2{%
675     \if\relax\detokenize{#1}\relax
676       \pgfkeys{%
677         /minted/##1/.cd,
678         #2/.code={\expandafter\def\csname minted@pyopt@##1##2@#2\endcsname{####1}},
679       }%
680     \else
681       \pgfkeys{%
682         /minted/##1/.cd,
683         #2/.code={%
684           \def\minted@tmp{####1}%
685           \ifx\minted@tmp\minted@const@pgfkeysnovalue
686             \expandafter\let\csname minted@pyopt@##1##2@#2\endcsname\minted@tmp
687           \else\ifcsname minted@opthandler@immediate@string#1\endcsname

```

```

688         #1{minted@pyopt@##1##2@#2}{####1}%
689     \else
690         \expandafter\def\csname minted@pyopt@##1##2@#2\endcsname{#1{####1}}%
691     \fi\fi},
692 }%
693 \fi
694 \ifx\minted@const@pgfkeysnovalue#4\relax
695     \pgfkeys{%
696         /minted/##1/.cd,
697         #2/.value required,
698     }%
699 \else
700     \pgfkeys{%
701         /minted/##1/.cd,
702         #2/.default={#4},
703     }%
704 \fi
705 }%
706 \minted@forcsvlist{\minted@do}{\minted@optscopes}%
707 \pgfkeys{%
708     /minted/global/.cd,
709     #2={#3},
710 }
711 \def\mintedpyoptvalueof#1{%
712     \ifbool{minted@isinline}%
713     {\minted@pyoptvalueof@inline{#1}}%
714     {\minted@pyoptvalueof@block{#1}}
715 \def\minted@pyoptvalueof@inline#1{%
716     \ifcsname minted@pyopt@cmd@#1\endcsname
717     \unexpanded\expandafter\expandafter\expandafter{%
718         \csname minted@pyopt@cmd@#1\endcsname}%
719     \else\ifcsname minted@pyopt@lexerinline@#1\endcsname
720     \unexpanded\expandafter\expandafter\expandafter{%
721         \csname minted@pyopt@lexerinline@#1\endcsname}%
722     \else\ifcsname minted@pyopt@globalinline@#1\endcsname
723     \unexpanded\expandafter\expandafter\expandafter{%
724         \csname minted@pyopt@globalinline@#1\endcsname}%
725     \else\ifcsname minted@pyopt@lexer@#1\endcsname
726     \unexpanded\expandafter\expandafter\expandafter{%
727         \csname minted@pyopt@lexer@#1\endcsname}%
728     \else
729     \unexpanded\expandafter\expandafter\expandafter{%
730         \csname minted@pyopt@global@#1\endcsname}%
731     \fi\fi\fi\fi}
732 \def\minted@pyoptvalueof@block#1{%
733     \ifcsname minted@pyopt@cmd@#1\endcsname
734     \unexpanded\expandafter\expandafter\expandafter{%
735         \csname minted@pyopt@cmd@#1\endcsname}%
736     \else\ifcsname minted@pyopt@lexer@#1\endcsname
737     \unexpanded\expandafter\expandafter\expandafter{%
738         \csname minted@pyopt@lexer@#1\endcsname}%
739     \else
740     \unexpanded\expandafter\expandafter\expandafter{%
741         \csname minted@pyopt@global@#1\endcsname}%

```

```

742 \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.
743 \def\minted@pgfkeyscreate@tex#1#2{%
744   \minted@forcsvlist{\minted@pgfkeycreate@tex{#1}}{#2}}
745 \begingroup
746 \catcode`\==12
747 \gdef\minted@pgfkeycreate@tex#1#2{%
748   \minted@pgfkeycreate@tex@i{#1}#2=\FV@Sentinel}
749 \gdef\minted@pgfkeycreate@tex@i#1#2=#3\FV@Sentinel{%
750   \if\relax\detokenize{#3}\relax
751     \expandafter\minted@pgfkeycreate@tex@ii
752   \else
753     \expandafter\minted@pgfkeycreate@tex@iii
754   \fi
755   {#1}{#2}#3\FV@Sentinel}
756 \gdef\minted@pgfkeycreate@tex@ii#1#2\FV@Sentinel{%
757   \minted@pgfkeycreate@tex@iv{#1}{#2}{\pgfkeysnovalue}}
758 \gdef\minted@pgfkeycreate@tex@iii#1#2#3=\FV@Sentinel{%
759   \minted@pgfkeycreate@tex@iv{#1}{#2}{#3}}
760 \endgroup
761 \def\minted@pgfkeycreate@tex@iv#1#2#3{%
762   \def\minted@do##1{%
763     \minted@iflexerscope{##1}%
764     {\minted@do@i{##1}{@\minted@lexer}}%
765     {\minted@do@i{##1}{}}}
766   \def\minted@do@i##1#2{%
767     \if\relax\detokenize{#1}\relax
768       \pgfkeys{%
769         /minted/##1/.cd,
770         #2/.code={\expandafter\def\csname minted@texopt@##1##2@#2\endcsname{###1}},
771         #2/.value required,
772       }%
773     \else
774       \pgfkeys{%
775         /minted/##1/.cd,
776         #2/.code={%
777           \def\minted@tmp{###1}%
778           \ifx\minted@tmp\minted@const\pgfkeysnovalue
779             \expandafter\let\csname minted@texopt@##1##2@#2\endcsname\minted@tmp
780           \else\ifcsname minted@opthandler@immediate@\string#1\endcsname

```

```

781         #1{minted@texopt@##1##2@#2}{####1}%
782     \else
783         \expandafter\def\csname minted@texopt@##1##2@#2\endcsname{#1{####1}}%
784     \fi\fi},
785     #2/.value required,
786 }%
787 \fi
788 }%
789 \minted@forcsvlist{\minted@do}{\minted@optscopes}%
790 \pgfkeys{%
791     /minted/global/.cd,
792     #2={#3},
793 }%
794 \def\mintedtexoptvalueof#1{%
795     \ifbool{minted@isinline}%
796     {\minted@texoptvalueof@inline{#1}}%
797     {\minted@texoptvalueof@block{#1}}%
798 \def\minted@texoptvalueof@inline#1{%
799     \ifcsname minted@texopt@cmd@#1\endcsname
800     \unexpanded\expandafter\expandafter\expandafter{%
801         \csname minted@texopt@cmd@#1\endcsname}%
802     \else\ifcsname minted@texopt@lexerinline@#1\endcsname
803     \unexpanded\expandafter\expandafter\expandafter{%
804         \csname minted@texopt@lexerinline@#1\endcsname}%
805     \else\ifcsname minted@texopt@globalinline@#1\endcsname
806     \unexpanded\expandafter\expandafter\expandafter{%
807         \csname minted@texopt@globalinline@#1\endcsname}%
808     \else\ifcsname minted@texopt@lexer@#1\endcsname
809     \unexpanded\expandafter\expandafter\expandafter{%
810         \csname minted@texopt@lexer@#1\endcsname}%
811     \else
812     \unexpanded\expandafter\expandafter\expandafter{%
813         \csname minted@texopt@global@#1\endcsname}%
814     \fi\fi\fi\fi}
815 \def\minted@texoptvalueof@block#1{%
816     \ifcsname minted@texopt@cmd@#1\endcsname
817     \unexpanded\expandafter\expandafter\expandafter{%
818         \csname minted@texopt@cmd@#1\endcsname}%
819     \else\ifcsname minted@texopt@lexer@#1\endcsname
820     \unexpanded\expandafter\expandafter\expandafter{%
821         \csname minted@texopt@lexer@#1\endcsname}%
822     \else
823     \unexpanded\expandafter\expandafter\expandafter{%
824         \csname minted@texopt@global@#1\endcsname}%
825     \fi\fi}
826 \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.

```

827 \def\minted@opthandler@deforrestrictedescape#1#2{%
828   \if\relax\detokenize{#2}\relax
829     \expandafter\def\csname#1\endcsname{#2}%
830   \else\if\relax\detokenize\expandafter{\@gobble#2}\relax
831     \ifcat\relax\noexpand#2%
832       \expandafter\expandafter\expandafter\minted@opthandler@deforrestrictedescape@i
833       \expandafter\@gobble\string#2\FV@Sentinel{#1}{#2}%
834     \else
835       \FVExtraDetokenizeREscVArg{\expandafter\def\csname#1\endcsname}{#2}%
836     \fi
837   \else
838     \FVExtraDetokenizeREscVArg{\expandafter\def\csname#1\endcsname}{#2}%
839   \fi\fi}
840 \def\minted@opthandler@deforrestrictedescape@i#1#2\FV@Sentinel#3#4{%
841   \ifcsname minted@isalpha\number`#1\endcsname
842     \expandafter\def\csname#3\endcsname{#4}%
843   \else
844     \FVExtraDetokenizeREscVArg{\expandafter\def\csname#3\endcsname}{#4}%
845   \fi}
846 \expandafter\let\csname
847   minted@opthandler@immediate@\string\minted@opthandler@deforrestrictedescape
848   \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.

```

849 \mintedpgfkeyscreate{fv}{
850   baselinestretch,
851   beameroverlays,
852   backgroundcolor,
853   backgroundcolorvphantom,
854   bgcolor,
855   bgcolorpadding,
856   bgcolorvphantom,
857   breakafter,
858   breakafterinrun,
859   breakaftersymbolpost,
860   breakaftersymbolpre,
861   breakanywhere,
862   breakanywhereinlinestretch,
863   breakanywheresymbolpost,
864   breakanywheresymbolpre,
865   breakautoindent,

```

866 breakbefore,
867 breakbeforeinrun,
868 breakbeforesymbolpost,
869 breakbeforesymbolpre,
870 breakbytoken,
871 breakbytokenanywhere,
872 breakindent,
873 breakindentnchars,
874 breaklines,
875 breaksymbol,
876 breaksymbolindent,
877 breaksymbolindentleft,
878 breaksymbolindentleftnchars,
879 breaksymbolindentnchars,
880 breaksymbolindentright,
881 breaksymbolindentrightnchars,
882 breaksymbolleft,
883 breaksymbolright,
884 breaksymbolsep,
885 breaksymbolsepleft,
886 breaksymbolsepleftnchars,
887 breaksymbolsepncchars,
888 breaksymbolsepright,
889 breaksymbolseprightnchars,
890 curlyquotes,
891 fillcolor,
892 firstline,
893 firstnumber,
894 fontencoding,
895 fontfamily,
896 fontseries,
897 fontshape,
898 fontsize,
899 formatcom,
900 frame,
901 framerule,
902 framesep,
903 highlightcolor,
904 highlightlines,
905 label,
906 labelposition,
907 lastline,
908 linenos,
909 listparameters,
910 numberblanklines,
911 numberfirstline,
912 numbers,
913 numbersep,
914 obeytabs,
915 resetmargins,
916 rulecolor,
917 samepage,
918 showspaces,
919 showtabs,

```

920 space,
921 spacecolor,
922 stepnumber,
923 stepnumberfromfirst,
924 stepnumberoffsetvalues,
925 tab,
926 tabcolor=black,
927 tabsize,
928 xleftmargin,
929 xrightmargin,
930 }

```

minted (passed to Python)

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

```

931 \mintedpgfkeyscreate{py}{
932   autogobble<true>=false,
933   encoding=utf8,
934   funcnamehighlighting<true>=true,
935   gobble=0,
936   gobblefilter=0,
937   keywordcase=none,
938   literalenvname=MintedVerbatim,
939   mathescape<true>=false,
940   python3<true>=true,
941   rangeregexmatchnumber=1,
942   rangeregexdotall<true>=false,
943   rangeregexmultiline<true>=false,
944   startinline<true>=false,
945   stripall<true>=false,
946   stripnl<true>=false,
947   texcl<true>=false,
948   texcomments<true>=false,
949 }
950 \mintedpgfkeyscreate[\minted@opthandler@deforrestrictedescape]{py}{
951   codetagify=,
952   escapeinside=,
953   literatecomment=,
954   rangestartstring=,
955   rangestartafterstring=,
956   rangestopstring=,
957   rangestopbeforestring=,
958   rangeregex=,
959 }
960 \let\minted@tmplexer\minted@lexer
961 \def\minted@lexer{php}
962 \pgfkeys{
963   /minted/lexerinline/.cd,
964   startinline=true,
965 }
966 \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`.

```

967 \mintedpgfkeyscreate{tex}{
968   envname=Verbatim,
969   ignorelexererrors=false,
970   style=default,
971 }
972 \pgfkeys{
973   /minted/globalinline/.cd,
974   envname=VerbEnv,
975 }
976 \expandafter\def\expandafter\minted@usetexoptsnonpygments\expandafter{%
977   \minted@usetexoptsnonpygments
978   \edef\minted@literalenvname{\mintedpyoptvalueof{literalenvname}}%
979   \edef\minted@envname{\mintedtexoptvalueof{envname}}%
980   \expandafter\def\expandafter\minted@literalenv\expandafter{%
981     \csname \minted@literalenvname\endcsname}%
982   \expandafter\def\expandafter\minted@endliteralenv\expandafter{%
983     \csname end\minted@literalenvname\endcsname}%
984   \expandafter\expandafter\expandafter
985     \let\expandafter\minted@literalenv\csname \minted@envname\endcsname
986   \expandafter\expandafter\expandafter
987     \let\expandafter\minted@endliteralenv\csname end\minted@envname\endcsname}%
988 \ifcsname minted@def@optcl\endcsname
989   \ifx\minted@def@optcl\relax
990     \let\minted@def@optcl\minted@undefined
991   \fi
992 \fi
993 \providecommand{\minted@def@optcl}[4] [] {%
994   \minted@warning{Macro \string\minted@def@optcl\space is deprecated with minted v3
995     and no longer has any effect}}

```

11.12 Caching, styles, and highlighting

11.12.1 Cache management

`\minted@addcachefilename`

`\minted@cachefile<n>`

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

```

996 \newcounter{minted@numcachefiles}
997 \def\minted@addcachefilename#1{%
998   \ifbool{minted@canexec}%
999     {\stepcounter{minted@numcachefiles}%
1000    \expandafter
1001      \xdef\csname minted@cachefile\arabic{minted@numcachefiles}\endcsname{#1}}%
1002   {}}

```

`\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 \AfterEndDocument so that all typesetting is complete, and thus the cache list is complete. \minted@fasthighlightmode@checkend is placed within the same \AfterEndDocument to guarantee correct ordering.

```

1003 \def\minted@clean{%
1004   \ifbool{minted@canexec}%
1005     {\ifbool{minted@diddetectconfig}{\minted@clean@i}{}}%
1006     {}%
1007   \def\minted@clean@i{%
1008     \ifnum\value{minted@numcachefiles}>0\relax
1009       \expandafter\minted@savecachelist
1010     \fi
1011     \ifbool{minted@fasthighlightmode}%
1012       {}%
1013       {\ifnum\value{minted@numcachefiles}>0\relax
1014         \expandafter\minted@exec@clean
1015       \else
1016         \expandafter\minted@exec@cleantemp
1017       \fi
1018       \global\boolfalse{minted@canexec}}%
1019     \setcounter{minted@numcachefiles}{0}%
1020   \def\minted@savecachelist{%
1021     \pydatasetfilename{\MintedDataFilename}%
1022     \minted@fasthighlightmode@checkstart
1023     \pydatawritedictopen
1024     \pydatawritekeyvalue{command}{clean}%
1025     \pydatawritekeyedefvalue{jobname}{\jobname}%
1026     \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
1027     \pydatawritekeyedefvalue{cachepath}{\minted@cachepath}%
1028     \pydatawritekey{cachefiles}%
1029     \pydatawritemlvaluestart
1030     \pydatawritemlvalueline{[]}%
1031     \setcounter{minted@tmpcnt}{1}%
1032     \loop\unless\ifnum\value{minted@tmpcnt}>\value{minted@numcachefiles}\relax
1033       \expandafter\minted@savecachelist@writecachefile\expandafter{%
1034         \csname minted@cachefile\arabic{minted@tmpcnt}\endcsname}%
1035       \expandafter\global\expandafter
1036       \let\csname minted@cachefile\arabic{minted@tmpcnt}\endcsname\minted@undefined
1037       \stepcounter{minted@tmpcnt}%
1038     \repeat
1039     \pydatawritemlvalueline{[]}%
1040     \pydatawritemlvalueend
1041     \pydatawritedictclose
1042     \ifbool{minted@fasthighlightmode}{\pydataclosefilename{\MintedDataFilename}}%
1043   \begingroup
1044     \catcode`\=12
1045     \catcode`\,=12
1046     \gdef\minted@savecachelist@writecachefile#1{%
1047       \expandafter\pydatawritemlvalueline\expandafter{\expandafter"#1",}}
1048   \endgroup
1049 \AfterEndDocument{%
1050   \minted@clean
1051   \minted@fasthighlightmode@checkend}

```

11.12.2 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.

```
1052 \def\minted@patch@PygmentsZsq{%
1053   \ifcsname\minted@styleprefix Zsq\endcsname
1054     \ifcsstring{\minted@styleprefix Zsq}{\char`'}{\minted@patch@PygmentsZsq@i}{}%
1055   \fi}
1056 \begingroup
1057 \catcode`\'= \active
1058 \gdef\minted@patch@PygmentsZsq@i{\def\PYGZsq{'}}
1059 \endgroup
1060 \def\minted@patch@PygmentsZhy{%
1061   \ifcsname\minted@styleprefix Zhy\endcsname
1062     \ifcsstring{\minted@styleprefix Zhy}{\char`\-}{\def\PYGZhy{\mbox{-}}}{}%
1063   \fi}
1064 \def\minted@patch@ignorelexererrors{%
1065   \edef\minted@tmp{\mintedtexoptvalueof{ignorelexererrors}}%
1066   \ifdefstring{\minted@tmp}{true}%
1067     {\expandafter\let\csname\minted@styleprefix @tok@err\endcsname\relax}%
1068   }%
1069 \def\minted@patch@PygmentsStyledef{%
1070   \minted@patch@PygmentsZsq
1071   \minted@patch@PygmentsZhy
1072   \minted@patch@ignorelexererrors}
```

`\minted@VerbatimPygments`

Enable fancyvrb features for Pygments macros.

```
1073 \def\minted@VerbatimPygments{%
1074   \expandafter\minted@VerbatimPygments@i\expandafter{%
1075     \csname\minted@styleprefix\endcsname}}%
1076 \def\minted@VerbatimPygments@i#1{%
1077   \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.

```
1078 \def\minted@standardcatcodes{%
1079   \catcode`\=0
1080   \catcode`\{=1
1081   \catcode`\}=2
1082   \catcode`\#=6
1083   \catcode`\ =10
1084   \catcode`\@=11
1085   \catcode`\`=12
1086   \catcode`\==12
1087   \catcode`\+=12
1088   \catcode`\.=12
1089   \catcode`\,=12
1090   \catcode`\[=12
1091   \catcode`\]=12
1092   \catcode`\%=14}
```

`\minted@defstyle`

Define highlighting style macros.

```
1093 \def\minted@defstyle{%
1094   \edef\minted@tmp{\mintedtexoptvalueof{style}}%
1095   \expandafter\minted@defstyle@i\expandafter{\minted@tmp}}
1096 \def\minted@defstyle@i#1{%
1097   \minted@ifalphanumhyphenunderscore{#1}%
1098   {\minted@defstyle@ii{#1}}%
1099   {\minted@error{Highlighting style is set to "#1" but only style names with
1100     alphanumeric characters, hyphens, and underscores are supported;
1101     falling back to default style}%
1102     \minted@defstyle@ii{default}}}%
1103 \def\minted@defstyle@ii#1{%
1104   \ifcsname minted@styledef@#1\endcsname
1105     \expandafter\@firstoftwo
1106   \else
1107     \expandafter\@secondoftwo
1108   \fi
1109   {\csname minted@styledef@#1\endcsname
1110    \minted@patch@PygmentsStyledef
1111    \minted@VerbatimPygments}%
1112   {\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 `_ 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 _ file regardless of whether it exists, unless it is wrapped in the \begingroup... \endgroup.`

```
1113 \def\minted@catchfiledef#1#2{%
1114   \CatchFileDef{#1}{#2}{\minted@standardcatcodes\endlinechar=-1}}
1115 \def\minted@defstyle@load#1{%
1116   \minted@detectconfig
1117   \ifbool{minted@cache}%
1118     {\edef\minted@styledeffilename{#1\detokenize{.style.minted}}%
1119      \edef\minted@styledeffilepath{\minted@cachepath\minted@styledeffilename}%
1120      \IfFileExists{\minted@styledeffilepath}%
1121      {\minted@defstyle@input{#1}}%
```

```

1122     {\ifbool{minted@canexec}%
1123     {\minted@defstyle@generate{#1}}%
1124     {\minted@error{Missing definition for highlighting style "#1" (minted executable
1125       is unavailable or disabled); attempting to substitute fallback style}%
1126     \minted@defstyle@fallback{#1}}}%
1127   {\edef\minted@styledeffilename{%
1128     \detokenize{_\MintedJobnameMdfive\detokenize{.style.minted}}%
1129     \let\minted@styledeffilepath\minted@styledeffilename
1130     \ifbool{minted@canexec}%
1131     {\minted@defstyle@generate{#1}}%
1132     {\minted@error{Missing definition for highlighting style "#1" (minted executable
1133       is unavailable or disabled); attempting to substitute fallback style}%
1134     \minted@defstyle@fallback{#1}}}%
1135   \def\minted@defstyle@input#1{%
1136     \begingroup
1137     \minted@catchfiledef{\minted@tmp}{\minted@styledeffilepath}%
1138     \minted@tmp
1139     \ifcsname\minted@styleprefix\endcsname
1140       \expandafter\@firstoftwo
1141     \else
1142       \expandafter\@secondoftwo
1143     \fi
1144     {\expandafter\global\expandafter\let\csname minted@styledef@#1\endcsname\minted@tmp
1145     \endgroup
1146     \ifbool{minted@cache}{\minted@addcachefilename{\minted@styledeffilename}}{-}%
1147     \csname minted@styledef@#1\endcsname
1148     \minted@patch@PygmentsStyledef
1149     \minted@VerbatimPygments}%
1150   {\endgroup
1151     \ifbool{minted@canexec}%
1152     {\minted@warning{Invalid or corrupted style definition file
1153       "\minted@styledeffilename"; attempting to regenerate}%
1154     \minted@defstyle@generate{#1}}%
1155     {\minted@error{Invalid or corrupted style definition file
1156       "\minted@styledeffilename"; attempting to substitute fallback style
1157       (minted executable is unavailable or disabled)}%
1158     \minted@defstyle@fallback{#1}}}%
1159   \def\minted@defstyle@generate#1{%
1160     \pydatasetfilename{\MintedDataFilename}%
1161     \minted@fasthighlightmode@checkstart
1162     \pydatawritedictopen
1163     \pydatawritekeyvalue{command}{styledef}%
1164     \pydatawritekeyedefvalue{jobname}{\jobname}%
1165     \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
1166     \pydatawritekeyedefvalue{currentfilepath}{\CurrentFilePath}%
1167     \pydatawritekeyedefvalue{currentfile}{\CurrentFile}%
1168     \pydatawritekeyedefvalue{inputlineno}{\the\inputlineno}%
1169     \pydatawritekeyedefvalue{cachepath}{\minted@cachepath}%
1170     \pydatawritekeyedefvalue{styledeffilename}{\minted@styledeffilename}%
1171     \pydatawritekeyvalue{style}{#1}%
1172     \pydatawritekeyedefvalue{commandprefix}{\minted@styleprefix}%
1173     \pydatawritedictclose
1174     \ifbool{minted@fasthighlightmode}%
1175     {\minted@defstyle@fallback{#1}}%

```

```

1176 {\pydataclosefilename{\MintedDataFilename}}%
1177 \begingroup
1178 \minted@exec@styledef
1179 \ifx\minted@exec@warning\relax
1180 \else
1181 \expandafter\minted@exec@warning
1182 \fi
1183 \ifx\minted@exec@error\relax
1184 \expandafter\minted@defstyle@generate@i
1185 \else
1186 \expandafter\minted@defstyle@generate@error
1187 \fi
1188 {#1}}
1189 \def\minted@defstyle@generate@i#1{%
1190 \endgroup
1191 \begingroup
1192 \minted@catchfiledef{\minted@tmp}{\minted@styledeffilepath}%
1193 \minted@tmp
1194 \ifcsname\minted@styleprefix\endcsname
1195 \expandafter\@firstoftwo
1196 \else
1197 \expandafter\@secondoftwo
1198 \fi
1199 {\expandafter\global\expandafter\let\csname\minted@styledef@#1\endcsname\minted@tmp
1200 \endgroup
1201 \ifbool{\minted@cache}{\minted@addcachefilename{\minted@styledeffilename}}{}%
1202 \csname\minted@styledef@#1\endcsname
1203 \minted@patch@PygmentsStyledef
1204 \minted@VerbatimPygments}%
1205 {\endgroup
1206 \minted@error{Failed to create style definition file "\minted@styledeffilename"
1207 (no error message, see "\MintedErrlogFilename" if it exists);
1208 attempting to substitute fallback style}%
1209 \minted@defstyle@fallback{#1}}
1210 \def\minted@defstyle@generate@error#1{%
1211 \minted@exec@error
1212 \endgroup
1213 \minted@defstyle@fallback{#1}}
1214 \def\minted@defstyle@fallback#1{%
1215 \ifstrequal{#1}{default}%
1216 {\expandafter\global\expandafter
1217 \let\csname\minted@styledef@default\endcsname\minted@styledeffallback}%
1218 {\ifcsname\minted@styledef@default\endcsname
1219 \else
1220 \minted@defstyle@load{default}%
1221 \fi
1222 \expandafter\let\expandafter\minted@tmp\csname\minted@styledef@default\endcsname
1223 \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.
1224 \def\minted@styledeffallback{%
1225 \expandafter\def\csname\minted@styleprefix\endcsname##1##2{##2}%

```

```

1226 \expandafter\def\csname\minted@styleprefix Zbs\endcsname{\char`\\}%
1227 \expandafter\def\csname\minted@styleprefix Zus\endcsname{\char`\_}%
1228 \expandafter\def\csname\minted@styleprefix Zob\endcsname{\char`\{}%
1229 \expandafter\def\csname\minted@styleprefix Zcb\endcsname{\char`\}}%
1230 \expandafter\def\csname\minted@styleprefix Zca\endcsname{\char`\^}%
1231 \expandafter\def\csname\minted@styleprefix Zam\endcsname{\char`\&%
1232 \expandafter\def\csname\minted@styleprefix Zlt\endcsname{\char`\<%
1233 \expandafter\def\csname\minted@styleprefix Zgt\endcsname{\char`\>%
1234 \expandafter\def\csname\minted@styleprefix Zsh\endcsname{\char`\#}%
1235 \expandafter\def\csname\minted@styleprefix Zpc\endcsname{\char`\}%
1236 \expandafter\def\csname\minted@styleprefix Zdl\endcsname{\char`\$}%
1237 \expandafter\def\csname\minted@styleprefix Zhy\endcsname{\char`\-}%
1238 \expandafter\def\csname\minted@styleprefix Zsq\endcsname{\char`\'}%
1239 \expandafter\def\csname\minted@styleprefix Zdq\endcsname{\char`\"}%
1240 \expandafter\def\csname\minted@styleprefix Zti\endcsname{\char`\~}%
1241 \minted@patch@PygmentsStyledef
1242 \minted@VerbatimPygments}

```

11.12.3 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.

```

1243 \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`.

```

1244 \def\minted@lexerlinenoson{%
1245   \ifcsname c@minted@lexer\minted@lexer\endcsname
1246   \else
1247     \newcounter{minted@lexer\minted@lexer}%
1248   \fi
1249   \setcounter{minted@FancyVerbLineTemp}{\value{FancyVerbLine}}%
1250   \setcounter{FancyVerbLine}{\value{minted@lexer\minted@lexer}}%
1251 \def\minted@lexerlinenosoff{%
1252   \setcounter{minted@lexer\minted@lexer}{\value{FancyVerbLine}}%
1253   \setcounter{FancyVerbLine}{\value{minted@FancyVerbLineTemp}}%
1254 \ifbool{minted@inputlexerlinenos}%
1255   {\let\minted@inputlexerlinenoson\minted@lexerlinenoson
1256    \let\minted@inputlexerlinenosoff\minted@lexerlinenosoff}%
1257   {\let\minted@inputlexerlinenoson\relax
1258    \let\minted@inputlexerlinenosoff\relax
1259   \ifbool{minted@lexerlinenos}
1260     {}%
1261     {\let\minted@lexerlinenoson\relax
1262      \let\minted@lexerlinenosoff\relax}}

```

`\minted@codewrapper`

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

```
1263 \def\minted@codewrapper#1{%
1264   \ifcsname minted@inputfilepath\endcsname
1265     \minted@inputlexerlinenoson
1266   \else
1267     \minted@lexerlinenoson
1268   \fi
1269   #1%
1270   \ifcsname minted@inputfilepath\endcsname
1271     \minted@inputlexerlinenosoff
1272   \else
1273     \minted@lexerlinenosoff
1274   \fi}
```

11.12.4 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.

```
1275 \def\minted@debug@input{%
1276   \ifbool{minted@debug}%
1277     {\immediate\typeout{%
1278       minted debug: \string\input\space at
1279       \ifx\CurrentFile@empty\else\CurrentFile\space\fi line \the\inputlineno}}%
1280   {}}
1281 \def\minted@highlight{%
1282   \minted@defstyle
1283   \pydatasetbuffername{minted@tmpdatabuffer}%
1284   \pydatabufferkeyvalue{command}{highlight}%
1285   \pydatabufferkey{code}%
1286   \pydatabuffermlvaluestart
1287   \setcounter{minted@tmpcnt}{1}%
1288   \loop\unless\ifnum\value{minted@tmpcnt}>\value{minted@tmpcodebufferlength}\relax
1289     \expandafter\let\expandafter
1290       \minted@tmp\csname minted@tmpcodebufferline\arabic{minted@tmpcnt}\endcsname
1291     \expandafter\pydatabuffermlvalueline\expandafter{\minted@tmp}%
1292     \stepcounter{minted@tmpcnt}%
1293   \repeat
```

```

1294 \pydatabuffermvalueend
1295 \minted@highlight@i}
1296 \def\minted@highlightinputfile{%
1297 \minted@defstyle
1298 \edef\minted@inputfilemdfivesum{\pdf@filemdfivesum{\minted@inputfilepath}}%
1299 \ifx\minted@inputfilemdfivesum\empty
1300 \expandafter\@firstoftwo
1301 \else
1302 \expandafter\@secondoftwo
1303 \fi
1304 {\minted@error{Cannot find input file "\minted@inputfilepath"; inserting placeholder}%
1305 \minted@insertplaceholder}%
1306 {\pydatasetbuffername{\minted@tmpdatabuffer}%
1307 \pydatabufferkeyvalue{command}{highlight}%
1308 \pydatabufferkeyedefvalue{inputfilepath}{\minted@inputfilepath}%
1309 \pydatabufferkeyedefvalue{inputfilemdfivesum}{\minted@inputfilemdfivesum}%
1310 \minted@highlight@i}}
1311 \def\minted@def@FV@GetKeyValues@standardcatcodes{%
1312 \let\minted@FV@GetKeyValues@orig\FV@GetKeyValues
1313 \def\FV@GetKeyValues##1{%
1314 \begingroup
1315 \minted@standardcatcodes
1316 \minted@FV@GetKeyValues@i{##1}}%
1317 \def\minted@FV@GetKeyValues@i##1[##2]{%
1318 \endgroup
1319 \let\FV@GetKeyValues\minted@FV@GetKeyValues@orig
1320 \let\minted@FV@GetKeyValues@i\minted@undefined
1321 \FV@GetKeyValues{##1}[##2]}}
1322 \def\minted@highlight@i{%
1323 \pydatabufferkeyedefvalue{pyopt.lexer}{\minted@lexer}%
1324 \pydatabufferkeyedefvalue{pyopt.commandprefix}{\minted@styleprefix}%
1325 \minted@forcsvlist{\minted@highlight@bufferpykeys}{\minted@optkeyslist@py}%
1326 \ifbool{\minted@cache}%
1327 {\edef\minted@highlightfilename{\pydatabuffermdfivesum\detokenize{.highlight.minted}}%
1328 \edef\minted@highlightfilepath{\minted@cachepath\minted@highlightfilename}%
1329 \IfFileExists{\minted@highlightfilepath}%
1330 {\minted@codewrapper{%
1331 \minted@def@FV@GetKeyValues@standardcatcodes
1332 \minted@debug@input
1333 \input{\minted@highlightfilepath}}%
1334 \minted@addcachefilename{\minted@highlightfilename}}%
1335 {\ifbool{\minted@canexec}%
1336 {\minted@iffasthighlightmode@buffertempfile
1337 \minted@highlight@create}%
1338 {\minted@error{Cannot highlight code (minted executable is unavailable or
1339 disabled); attempting to typeset without highlighting}%
1340 \minted@highlight@fallback}}}%
1341 {\edef\minted@highlightfilename{%
1342 \detokenize{_\MintedJobnameMdfive\detokenize{.highlight.minted}}%
1343 \let\minted@highlightfilepath\minted@highlightfilename
1344 \ifbool{\minted@canexec}%
1345 {\minted@highlight@create}%
1346 {\minted@error{Cannot highlight code (minted executable is unavailable or
1347 disabled); attempting to typeset without highlighting}%

```

```

1348     \minted@highlight@fallback}}}%
1349     \pydataclearbuffername{\minted@tmpdatabuffer}}
1350 \def\minted@highlight@bufferpykeys#1{%
1351     \edef\minted@tmp{\mintedpyoptvalueof{#1}}%
1352     \ifx\minted@tmp\minted@const@pgfkeysnovalue
1353     \else
1354         \pydatabufferkeyedefvalue{pyopt.#1}{\minted@tmp}%
1355     \fi}
1356 \def\minted@highlight@create{%
1357     \pydatasetfilename{\MintedDataFilename}%
1358     \minted@fasthighlightmode@checkstart
1359     \pydatawritedictopen
1360     \pydatawritebuffer
1361     \pydatawritekeyedefvalue{jobname}{\jobname}%
1362     \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
1363     \pydatawritekeyedefvalue{currentfilepath}{\CurrentFilePath}%
1364     \pydatawritekeyedefvalue{currentfile}{\CurrentFile}%
1365     \pydatawritekeyedefvalue{inputlineno}{\the\inputlineno}%
1366     \pydatawritekeyedefvalue{cachepath}{\minted@cachepath}%
1367     \pydatawritekeyedefvalue{highlightfilename}{\minted@highlightfilename}%
1368     \pydatawritedictclose
1369     \ifbool{\minted@fasthighlightmode}%
1370     {\minted@insertplaceholder}%
1371     {\pydataclosefilename{\MintedDataFilename}%
1372     \begingroup
1373     \minted@exec@highlight
1374     \IfFileExists{\minted@highlightfilepath}%
1375     {\ifx\minted@exec@warning\relax
1376     \else
1377         \expandafter\minted@exec@warning
1378     \fi
1379     \ifx\minted@exec@error\relax
1380     \else
1381         \expandafter\minted@exec@error
1382     \fi
1383     \endgroup
1384     \minted@codewrapper{%
1385         \minted@def@FV@GetKeyValues@standardcatcodes
1386         \minted@debug@input
1387         \input{\minted@highlightfilepath}}}%
1388     \ifbool{\minted@cache}{\minted@addcachefilename{\minted@highlightfilename}}{}}%
1389     {\ifx\minted@exec@warning\relax
1390     \else
1391         \expandafter\minted@exec@warning
1392     \fi
1393     \ifx\minted@exec@error\relax
1394     \minted@error{Minted executable failed during syntax highlighting
1395     but returned no error message (see if "\MintedErrlogFilename" exists)}%
1396     \else
1397         \expandafter\minted@exec@error
1398     \fi
1399     \endgroup
1400     \minted@highlight@fallback}}}%
1401 \def\minted@highlight@fallback{%

```

```

1402 \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 TEX during the reading process,
    so it is not possible to reconstruct the original file bytes within TEX, 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.

1403 \begingroup
1404 \def\minted@set@tempfileextension#1{%
1405   \if\relax\detokenize{#1}\relax
1406   \else
1407     \expandafter\global\expandafter
1408     \let\csname minted@buffertempfileextension@#1\endcsname\relax
1409   \fi}
1410 \minted@forcsvlist{\minted@set@tempfileextension}{
1411   listing,
1412   out,
1413   outfile,
1414   output,
1415   temp,
1416   tempfile,
1417   tmp,
1418   verb,
1419   vrb,
1420 }
1421 \endgroup
1422 \begingroup
1423 \catcode`\/=12
1424 \catcode`\.=12
1425 \gdef\minted@iffasthighlightmode@buffertempfile{%
1426   \ifbool{minted@fasthighlightmode}%
1427   {\ifcsname minted@inputfilepath\endcsname
1428     \expandafter\@firstofone
1429   \else
1430     \expandafter\@gobble
1431   \fi
1432   {\expandafter
1433     \minted@iffasthighlightmode@buffertempfile@i\minted@inputfilepath/\FV@Sentinel}}}%
1434   {}
1435 \gdef\minted@iffasthighlightmode@buffertempfile@i#1/#2\FV@Sentinel{%
1436   \if\relax\detokenize{#2}\relax
1437     \expandafter\@firstoftwo
1438   \else
1439     \expandafter\@secondoftwo

```

```

1440 \fi
1441 {\minted@iffasthighlightmode@buffertempfile@ii#1.\FV@Sentinel}%
1442 {\minted@iffasthighlightmode@buffertempfile@ii#2\FV@Sentinel}}
1443 \gdef\minted@iffasthighlightmode@buffertempfile@iii#1.#2\FV@Sentinel{%
1444 \if\relax\detokenize{#2}\relax
1445 \expandafter\@gobble
1446 \else
1447 \expandafter\@firstofone
1448 \fi
1449 {\minted@iffasthighlightmode@buffertempfile@iii#2\FV@Sentinel}}
1450 \gdef\minted@iffasthighlightmode@buffertempfile@iii#1.\FV@Sentinel{%
1451 \ifcsname minted@buffertempfileextension@#1\endcsname
1452 \expandafter\@firstofone
1453 \else
1454 \expandafter\@gobble
1455 \fi
1456 {\minted@iffasthighlightmode@buffertempfile@iv}}
1457 \endgroup
1458 \def\minted@iffasthighlightmode@buffertempfile@iv{%
1459 \begingroup
1460 \setcounter{minted@tmpcodebufferlength}{0}%
1461 \newread\minted@intempfile
1462 \openin\minted@intempfile=\minted@inputfilepath
1463 \endlinechar=-1%
1464 \let\do\@makeother\FVExtraDoSpecials
1465 \loop\unless\ifeof\minted@intempfile
1466 \read\minted@intempfile to\minted@intempfileline
1467 \stepcounter{minted@tmpcodebufferlength}%
1468 \expandafter\global\expandafter\let\csname
1469 minted@tmpcodebufferline\arabic{minted@tmpcodebufferlength}%
1470 \endcsname\minted@intempfileline
1471 \repeat
1472 \closein\minted@intempfile
1473 \expandafter\ifx\csname
1474 minted@tmpcodebufferline\arabic{minted@tmpcodebufferlength}%
1475 \endcsname\@empty
1476 \expandafter\global\expandafter\let\csname
1477 minted@tmpcodebufferline\arabic{minted@tmpcodebufferlength}%
1478 \endcsname\minted@undefined
1479 \addtocounter{minted@tmpcodebufferlength}{-1}%
1480 \fi
1481 \endgroup
1482 \edef\minted@inputfilemdfivesum@check{\pdf@filemdfivesum{\minted@inputfilepath}}%
1483 \ifx\minted@inputfilemdfivesum@check\minted@inputfilemdfivesum
1484 \expandafter\@gobble
1485 \else
1486 \expandafter\@firstofone
1487 \fi
1488 {\VerbatimClearBuffer[buffername=minted@tmpcodebuffer]}%
1489 \ifnum\value{minted@tmpcodebufferlength}>0\relax
1490 \expandafter\@firstofone
1491 \else
1492 \expandafter\@gobble
1493 \fi

```

```

1494     {\minted@iffasthighlightmode@buffertempfile@v}}
1495 \def \minted@iffasthighlightmode@buffertempfile@v{%
1496   \pydatabufferkey{code}%
1497   \pydatabuffermlvaluestart
1498   \setcounter{minted@tmpcnt}{1}%
1499   \loop\unless\ifnum\value{minted@tmpcnt}>\value{minted@tmpcodebufferlength}\relax
1500     \expandafter\let\expandafter
1501       \minted@tmp\csname minted@tmpcodebufferline\arabic{minted@tmpcnt}\endcsname
1502     \expandafter\pydatabuffermlvalueline\expandafter{\minted@tmp}%
1503     \stepcounter{minted@tmpcnt}%
1504   \repeat
1505   \pydatabuffermlvalueend
1506   \VerbatimClearBuffer[buffername=minted@tmpcodebuffer]}

```

11.13 Public API

`\setminted`

Set global or lexer-level options.

```

1507 \newcommand{\setminted}[2] [] {%
1508   \ifstrepty{#1}%
1509   {\pgfkeys{/minted/global/.cd,#2}}%
1510   {\let\minted@tmplexer\minted@lexer
1511     \edef\minted@lexer{#1}%
1512     \pgfkeys{/minted/lexer/.cd,#2}%
1513     \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.

```

1514 \newcommand{\setmintedinline}[2] [] {%
1515   \ifstrepty{#1}%
1516   {\pgfkeys{/minted/globalinline/.cd,#2}}%
1517   {\let\minted@tmplexer\minted@lexer
1518     \edef\minted@lexer{#1}%
1519     \pgfkeys{/minted/lexerinline/.cd,#2}%
1520     \let\minted@lexer\minted@tmplexer}}

```

`\usemintedstyle`

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

```

1521 \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`.

```

1522 \def \mintinline{%
1523   \FVExtraRobustCommand\RobustMintInline\FVExtraUnexpandedReadStar0ArgMArgBVar}%
1524   \FVExtraPDFstringdefDisableCommands{%
1525     \def \RobustMintInline{}}

```

```

1526 \newrobustcmd{\RobustMintInline}[2][{}]{%
1527   \ifbool{FVExtraRobustCommandExpanded}%
1528     {\@ifnextchar\bgroup
1529       {\FVExtraReadVArg{\RobustMintInlineProcess{#1}{#2}}}%
1530       {\minted@error{Inline delimiters must be paired curly braces in this context}}}%
1531     {\FVExtraReadVArg{\RobustMintInlineProcess{#1}{#2}}}}
1532 \def\RobustMintInlineProcess@highlight#1#2#3{%
1533   \begingroup
1534   \booltrue{minted@isinline}%
1535   \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1536   \edef\minted@lexer{#2}%
1537   \minted@usefvopts
1538   \minted@usetexoptsnonpygments
1539   \FVExtraDetokenizeVArg{%
1540     \FVExtraRetokenizeVArg{\RobustMintInlineProcess@highlight@i}{\FV@CatCodes}}{#3}}
1541 \def\RobustMintInlineProcess@highlight@i#1{%
1542   \expandafter\def\csname minted@tmpcodebufferline1\endcsname{#1}%
1543   \setcounter{minted@tmpcodebufferlength}{1}%
1544   \let\minted@highlight@fallback\RobustMintInlineProcess@highlight@fallback
1545   \minted@highlight
1546   \setcounter{minted@tmpcodebufferlength}{0}%
1547   \endgroup}
1548 \def\RobustMintInlineProcess@highlight@fallback{%
1549   \minted@useadditionalfvoptsnoopy
1550   \fvset{extra=true}%
1551   \minted@codewrapper{%
1552     \expandafter\let\expandafter\minted@tmp\csname minted@tmpcodebufferline1\endcsname
1553     \expandafter\Verb\expandafter{\minted@tmp}}}%
1554 \def\RobustMintInlineProcess@placeholder#1#2#3{%
1555   \begingroup
1556   \booltrue{minted@isinline}%
1557   \minted@insertplaceholder
1558   \endgroup}
1559 \def\RobustMintInlineProcess@verbatim#1#2#3{%
1560   \begingroup
1561   \booltrue{minted@isinline}%
1562   \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1563   \edef\minted@lexer{#2}%
1564   \minted@usefvopts
1565   \minted@useadditionalfvoptsnoopy
1566   \minted@usetexoptsnonpygments
1567   \fvset{extra=true}%
1568   \minted@codewrapper{\Verb{#3}}%
1569   \endgroup}
1570 \ifbool{minted@placeholder}%
1571   {\let\RobustMintInlineProcess\RobustMintInlineProcess@placeholder}%
1572   {\ifbool{minted@verbatim}%
1573     {\let\RobustMintInlineProcess\RobustMintInlineProcess@verbatim}%
1574     {\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).

```

1575 \def\mint{%
1576   \FVExtraRobustCommand\RobustMint\FVExtraUnexpandedReadStar0ArgMArgBVar{
1577   \FVExtraPdfstringdefDisableCommands{%
1578     \def\RobustMint{}}
1579   \newrobustcmd{\RobustMint}[2][{}]{%
1580     \ifbool{FVExtraRobustCommandExpanded}%
1581       {\@ifnextchar\bgroup
1582         {\FVExtraReadVArg{\RobustMintProcess{#1}{#2}}}%
1583         {\minted@error{Delimiters must be paired curly braces in this context}}}%
1584       {\FVExtraReadVArg{\RobustMintProcess{#1}{#2}}}%
1585   \def\RobustMintProcess@highlight#1#2#3{%
1586     \begingroup
1587     \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1588     \edef\minted@lexer{#2}%
1589     \minted@usefvopts
1590     \minted@usetexoptsnonpygments
1591     \FVExtraDetokenizeVArg{%
1592       \FVExtraRetokenizeVArg{\RobustMintProcess@highlight@i}{\FV@CatCodes}}{#3}}
1593   \def\RobustMintProcess@highlight@i#1{%
1594     \expandafter\def\csname minted@tmpcodebufferline1\endcsname{#1}%
1595     \setcounter{minted@tmpcodebufferlength}{1}%
1596     \let\minted@highlight@fallback\RobustMintProcess@highlight@fallback
1597     \minted@highlight
1598     \setcounter{minted@tmpcodebufferlength}{0}%
1599     \endgroup}
1600   \def\RobustMintProcess@highlight@fallback{%
1601     \minted@useadditionalfvoptsnoy
1602     \minted@codewrapper{%
1603       \VerbatimInsertBuffer[buffername=minted@tmpcodebuffer,insertenvname=\minted@envname]}%
1604   \def\RobustMintProcess@placeholder#1#2#3{%
1605     \minted@insertplaceholder}
1606   \def\RobustMintProcess@verbatim#1#2#3{%
1607     \begingroup
1608     \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1609     \edef\minted@lexer{#2}%
1610     \minted@usefvopts
1611     \minted@useadditionalfvoptsnoy
1612     \minted@usetexoptsnonpygments
1613     \FVExtraDetokenizeVArg{%
1614       \FVExtraRetokenizeVArg{\RobustMintProcess@verbatim@i}{\FV@CatCodes}}{#3}}
1615   \def\RobustMintProcess@verbatim@i#1{%
1616     \expandafter\def\csname minted@tmpcodebufferline1\endcsname{#1}%
1617     \setcounter{minted@tmpcodebufferlength}{1}%
1618     \minted@codewrapper{%
1619       \VerbatimInsertBuffer[buffername=minted@tmpcodebuffer,insertenvname=\minted@envname]}%
1620     \setcounter{minted@tmpcodebufferlength}{0}%
1621     \endgroup}
1622   \ifbool{minted@placeholder}%
1623     {\let\RobustMintProcess\RobustMintProcess@placeholder}%
1624     {\ifbool{minted@verbatim}%
1625       {\let\RobustMintProcess\RobustMintProcess@verbatim}%

```

```

1626     {\let\RobustMintProcess\RobustMintProcess@highlight}}
minted (env.)
    Highlight a longer piece of code inside a verbatim environment.
1627 \newenvironment{minted}[2] []%
1628 {\VerbatimEnvironment
1629  \MintedBegin{#1}{#2}}%
1630 {\MintedEnd}
1631 \def\MintedBegin@highlight#1#2{%
1632  \ifstrepty{#1}{\pgfkeys{/minted/cmd/.cd,#1}}%
1633  \edef\minted@lexer{#2}%
1634  \minted@usefvopts
1635  \minted@usetexoptsnonpygments
1636  \begin{VerbatimBuffer}[buffername=minted@tmpcodebuffer,globalbuffer=true]}
1637 \def\MintedEnd@highlight{%
1638  \end{VerbatimBuffer}%
1639  \let\minted@highlight@fallback\MintedEnv@highlight@fallback
1640  \minted@highlight
1641  \VerbatimClearBuffer[buffername=minted@tmpcodebuffer]}
1642 \def\MintedEnv@highlight@fallback{%
1643  \minted@useadditionalfvoptsnoopy
1644  \minted@codewrapper{%
1645    \VerbatimInsertBuffer[buffername=minted@tmpcodebuffer,insertenvname=\minted@envname]}}
1646 \def\MintedBegin@placeholder#1#2{%
1647  \begin{VerbatimBuffer}[buffername=minted@tmpcodebuffer]}
1648 \def\MintedEnd@placeholder{%
1649  \end{VerbatimBuffer}%
1650  \minted@insertplaceholder}
1651 \def\MintedBegin@verbatim#1#2{%
1652  \ifstrepty{#1}{\pgfkeys{/minted/cmd/.cd,#1}}%
1653  \edef\minted@lexer{#2}%
1654  \minted@usefvopts
1655  \minted@useadditionalfvoptsnoopy
1656  \minted@usetexoptsnonpygments
1657  \begin{\minted@envname}}
1658 \def\MintedEnd@verbatim{%
1659  \end{\minted@envname}}
1660 \ifbool{minted@placeholder}%
1661 {\let\MintedBegin\MintedBegin@placeholder
1662  \let\MintedEnd\MintedEnd@placeholder}%
1663 {\ifbool{minted@verbatim}%
1664  {\let\MintedBegin\MintedBegin@verbatim
1665   \let\MintedEnd\MintedEnd@verbatim}%
1666  {\let\MintedBegin\MintedBegin@highlight
1667   \let\MintedEnd\MintedEnd@highlight}}
\inputminted
    Highlight an external source file.
1668 \def\minted@readinputmintedargs#1#{%
1669  \minted@readinputmintedargs@i{#1}}
1670 \def\minted@readinputmintedargs@i#1#2#3{%
1671  \FVExtraAlwaysUnexpanded{\minted@readinputmintedargs#1{#2}{#3}}}
1672 \FVExtrapdfstringdefDisableCommands{%
1673  \makeatletter
1674  \def\minted@readinputmintedargs@i#1#2#3{%

```

```

1675 \detokenize{<input from file ">#3\detokenize{>}}}%
1676 \makeatother}
1677 \def\inputminted{%
1678 \FVExtraRobustCommand\RobustInputMinted\minted@readinputmintedargs}
1679 \FVExtraPdfstringdefDisableCommands{%
1680 \def\RobustInputMinted{}}
1681 \newrobustcmd{\RobustInputMinted}[3][[]]{%
1682 \RobustInputMintedProcess{#1}{#2}{#3}}
1683 \def\RobustInputMintedProcess@highlight#1#2#3{%
1684 \begingroup
1685 \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1686 \edef\minted@lexer{#2}%
1687 \edef\minted@inputfilepath{#3}%
1688 \minted@usefvopts
1689 \minted@usetexoptsnonpygments
1690 \let\minted@highlight@fallback\RobustInputMintedProcess@highlight@fallback
1691 \minted@highlightinputfile
1692 \endgroup}
1693 \def\RobustInputMintedProcess@highlight@fallback{%
1694 \minted@useadditionalfvoptsnoy
1695 \minted@codewrapper{%
1696 \csname\minted@envname Input\endcsname{\minted@inputfilepath}}}%
1697 \def\RobustInputMintedProcess@placeholder#1#2#3{%
1698 \minted@insertplaceholder}
1699 \def\RobustInputMintedProcess@verbatim#1#2#3{%
1700 \begingroup
1701 \ifstrempy{#1}{-}{\pgfkeys{/minted/cmd/.cd,#1}}%
1702 \edef\minted@lexer{#2}%
1703 \edef\minted@inputfilepath{#3}%
1704 \minted@usefvopts
1705 \minted@useadditionalfvoptsnoy
1706 \minted@usetexoptsnonpygments
1707 \minted@codewrapper{%
1708 \csname\minted@envname Input\endcsname{\minted@inputfilepath}}}%
1709 \endgroup}
1710 \ifbool{minted@placeholder}%
1711 {\let\RobustInputMintedProcess\RobustInputMintedProcess@placeholder}%
1712 {\ifbool{minted@verbatim}%
1713 {\let\RobustInputMintedProcess\RobustInputMintedProcess@verbatim}%
1714 {\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 `\FVExtraReadOArgBeforeVEnv` strips a following `^^M` (basically the newline), but `fancyvrb` environments expect `^^M` before the start of environment contents.

```

1715 \newcommand{\newminted}[3][]{%
1716   \ifstrempy{#1}%
1717   {\newminted@i{#2code}{#2}{#3}}%
1718   {\newminted@i{#1}{#2}{#3}}}
1719 \begingroup
1720 \catcode\^M=\active%
1721 \gdef\newminted@i#1#2#3{%
1722   \expandafter\def\csname#1i\endcsname##1{%
1723     \begin{minted}[#3,##1]{#2}^M}%
1724   \newenvironment{#1}%
1725     {\VerbatimEnvironment%
1726      \FVExtraRead0ArgBeforeVEnv{\csname#1i\endcsname}}%
1727     {\end{minted}}}%
1728   \newenvironment{#1*}[1]%
1729     {\VerbatimEnvironment%
1730      \begin{minted}[#3,##1]{#2}}%
1731     {\end{minted}}}%
1732 \endgroup

```

\newmint

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

```

1733 \newcommand{\newmint}[3][]{%
1734   \ifstrempy{#1}%
1735   {\edef\minted@tmp{#2}}%
1736   {\edef\minted@tmp{#1}}}
1737 \expandafter\newmint@i\expandafter{\minted@tmp}{#2}{#3}}
1738 \def\newmint@i#1#2#3{%
1739   \expandafter\newcommand\csname#1\endcsname{%
1740     \expandafter\FVExtraRobustCommand\csname RobustNewMint#1\endcsname
1741     \FVExtraUnexpandedReadStar0ArgBVar}%
1742   \FVExtraPDFstringdefDisableCommands{%
1743     \expandafter\def\csname RobustNewMint#1\endcsname{}}%
1744   \expandafter\newrobustcmd\csname RobustNewMint#1\endcsname{%
1745     \FVExtraRead0ArgBeforeVArg{\csname RobustNewMint#1i\endcsname}}%
1746   \expandafter\def\csname RobustNewMint#1i\endcsname##1{%
1747     \ifbool{FVExtraRobustCommandExpanded}%
1748     {\@ifnextchar\bgroup
1749      {\FVExtraReadVArg{\csname RobustNewMint#1ii\endcsname{##1}}}%
1750      {\minted@error{Delimiters must be paired curly braces in this context}}}%
1751     {\FVExtraReadVArg{\csname RobustNewMint#1ii\endcsname{##1}}}}
1752   \expandafter\def\csname RobustNewMint#1ii\endcsname##1##2{%
1753     \RobustMintProcess{#3,##1}{#2}{##2}}

```

\newmintedfile

Define a new language-specific alias for \inputminted.

```

1754 \def\minted@readnewmintedfileargs#1{%
1755   \minted@readnewmintedfileargs@i{#1}}
1756 \def\minted@readnewmintedfileargs@i#1#2{%
1757   \FVExtraAlwaysUnexpanded{\minted@readnewmintedfileargs#1{#2}}}
1758 \FVExtraPDFstringdefDisableCommands{%
1759   \makeatletter
1760   \def\minted@readnewmintedfileargs@i#1#2{%
1761     \detokenize{<input from file "#2\detokenize{>}}}%
1762   \makeatother}
1763 \newcommand{\newmintedfile}[3][]{%

```

```

1764 \ifstrepty{#1}%
1765 {\edef\minted@tmp{#2file}}%
1766 {\edef\minted@tmp{#1}}%
1767 \expandafter\newmintedfile@i\expandafter{\minted@tmp}{#2}{#3}}
1768 \def\newmintedfile@i#1#2#3{%
1769 \expandafter\newcommand\csname#1\endcsname{%
1770 \expandafter\FVExtraRobustCommand\csname RobustNewMintedFile#1\endcsname
1771 \minted@readnewmintedfileargs}%
1772 \FVExtraPDFstringdefDisableCommands{%
1773 \expandafter\def\csname RobustNewMintedFile#1\endcsname{}}%
1774 \expandafter\newrobustcmd\csname RobustNewMintedFile#1\endcsname[2] []{%
1775 \RobustInputMintedProcess{#3,#1}{#2}{##2}}}

\newmintinline
Define an alias for \mintinline.
1776 \newcommand{\newmintinline}[3] []{%
1777 \ifstrepty{#1}%
1778 {\edef\minted@tmp{#2inline}}%
1779 {\edef\minted@tmp{#1}}%
1780 \expandafter\newmintinline@i\expandafter{\minted@tmp}{#2}{#3}}
1781 \def\newmintinline@i#1#2#3{%
1782 \expandafter\newcommand\csname#1\endcsname{%
1783 \expandafter\FVExtraRobustCommand\csname RobustNewMintInline#1\endcsname
1784 \FVExtraUnexpandedReadStar0ArgBVar}%
1785 \FVExtraPDFstringdefDisableCommands{%
1786 \expandafter\def\csname RobustNewMintInline#1\endcsname{}}%
1787 \expandafter\newrobustcmd\csname RobustNewMintInline#1\endcsname{%
1788 \FVExtraRead0ArgBeforeVArg{\csname RobustNewMintInline#1@i\endcsname}}%
1789 \expandafter\def\csname RobustNewMintInline#1@i\endcsname##1{%
1790 \ifbool{FVExtraRobustCommandExpanded}%
1791 {\@ifnextchar\bgroup
1792 {\FVExtraReadVArg{\csname RobustNewMintInline#1@ii\endcsname{##1}}}%
1793 {\minted@error{Inline delimiters must be paired curly braces in this context}}}%
1794 {\FVExtraReadVArg{\csname RobustNewMintInline#1@ii\endcsname{##1}}}%
1795 \expandafter\def\csname RobustNewMintInline#1@ii\endcsname##1##2{%
1796 \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.

```

1797 \ifbool{minted@newfloat}%
1798 {\@ifundefined{minted@float@within}%
1799 {\DeclareFloatingEnvironment[fileext=lol,placement=tbp]{listing}}%
1800 {\def\minted@tmp#1{%
1801 \DeclareFloatingEnvironment[fileext=lol,placement=tbp,within=#1]{listing}}%
1802 \expandafter\minted@tmp\expandafter{\minted@float@within}}%
1803 {\@ifundefined{minted@float@within}%
1804 {\newfloat{listing}{tbp}{lol}}%
1805 {\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.
1806 \ifminted@newfloat\else
\listingcaption
    The name that is displayed before each individual listings caption and its number.
    The macro \listingscaption can be redefined by the user.
1807 \newcommand{\listingscaption}{Listing}
    The following definition should not be changed by the user.
1808 \floatname{listing}{\listingscaption}
\listoflistingscaption
    The caption that is displayed for the list of listings.
1809 \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.
1810 \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.
1811 \fi

```