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

v3.8.0 from 2026/03/03

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

Contents

1 Introduction

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

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

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

```
\begin{minted}{ruby}
  class Foo
    def init
      pi = Math::PI
      @var = "Pi = #{pi}..."
    end
  end
\end{minted}
```

```ruby
class Foo
  def init
    pi = Math::PI
    @var = "Pi = #{pi}..."
  end
end
```

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

2 Installation

2.1 Package manager

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

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

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

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

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

- catchfile
- etoolbox
- float
- fvextra
- latex2pydata
- newfloat
- 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), this can be accomplished by running `pip install latexminted`. This will install latexminted plus all dependencies including Pygments. You may need to modify the command depending on whether you want a system or user (`--user`) installation, depending on whether you are using virtual environments, and depending on whether something like `pip3` is needed instead of `pip`.

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

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

2.2.2 Option 2: Install latexminted within TeX installation

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

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

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

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

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

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

3 Migrating from minted version 2

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

Backward compatibility

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

New features and changes

- Version 3 uses a new minted-specific Python executable called latexminted to perform syntax highlighting. This executable is specifically designed to meet the security requirements for restricted shell escape programs. Once it has passed a security review and is accepted by TEX distributions, it will be possible to highlight code without -shell-escape and its attendant security vulnerabilities. TeX Live 2024+ no longer needs -shell-escape.

  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.
- By default, frozencache now requires both the cache and any external files that are accessed by commands like \inputminted. To avoid the requirement for external files (as in minted version 2), see the new package option cacheignoresfilecontents.
- \newminted now creates an environment that takes an optional argument consisting of options, instead of taking no argument.
- File encoding changes: The new latexminted executable assumes that LaTeX output files are UTF-8, and saves highlighted code as UTF-8. That is, LaTeX should be configured so that everything is UTF-8. The encoding option now defaults to UTF-8. It is only used in decoding files for \inputminted and commands based on it. The outencoding option is no longer supported.
- Added new options for including ranges of code based on literal string delimiters or regular expressions: rangestartstring, rangestartafterstring, rangestopstring, rangestopbeforestring, rangeregex.
- There is now support for custom lexers in standalone Python files. See the documentation for the new .latexminted_config configuration files for details.
- Several package options are no longer supported and result in errors or warnings. The package options finalizecache, outputdir, and kpsewhich are no longer needed given new minted version 3 capabilities. The package options draft and final no longer have any effect and will soon be removed altogether. The new package options placeholder and verbatim are available in cases where using highlighted code should be completely avoided.

4 Basic usage

4.1 The latexminted Python executable and shell escape

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

`latexminted` is designed to be compatible with the security requirements for restricted shell escape. For up-to-date installations of TeX Live 2024+, the `-shell-escape` option is no longer required. The `latexminted` Python executable has been added to TeX Live's list of trusted executables. Compiling a document with minted should typically require no changes from normal LaTeX compilation.

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

- TeX Live: Copy the variable `shell_escape_commands` from the distribution `texmf.cnf` (something like `<texlive>/<yr>/texmf-dist/web2c/texmf.cnf`) into the user `texmf.cnf` (something like `<texlive>/<yr>/texmf.cnf`), and then add `latexminted` to the `shell_escape_commands` list. The location of the `texmf.cnf` files can be determined by running `kpsewhich -all texmf.cnf`. Note that under Windows, this only works when `latexminted` is installed within a TeX Live installation; it is not compatible with `latexminted` being installed in a Python installation.

- MiKTeX: Add a line `AllowedShellCommands[] = latexminted` to the existing list of allowed commands in `miktex.ini`. You may want to modify the user-scoped configuration instead of the system-wide configuration. See the MiKTeX documentation for more details, particularly `initexmf --edit-config-file` and `initexmf --set-config-value`.

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

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

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

See the latexrestricted documentation for details.

4.2 A minimal complete example

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

```
\documentclass{article}

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

\begin{document}
\begin{minted}[bgcolor=Beige, bgcolorpadding=0.5em]{c}
```

```
int main() {
    printf("hello, world");
    return 0;
}
\end{minted}
\end{document}
```

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

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

4.3 Formatting source code

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

```
\begin{minted}{python}
def boring(args = None):
    pass
\end{minted}
```

```
def boring(args = None):
    pass
```

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`:

```
\mint{python}/import this/
```

```
import this
```

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:

```
\mintinline{py}{print("Hello!")}
```

```
print("Hello!")
```

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` 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:

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

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

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

Config settings

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

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

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

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

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

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

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

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

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

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

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

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

6.2 macOS compatibility

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

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

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

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

7 Options

7.1 Package options

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

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

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

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

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

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

`cacheignoresfilecontents=⟨boolean⟩` (default: false)

By default, when highlighted versions of external files are accessed in the cache, they are checked against the original files. If the original files have been modified, then the cache is updated. This requires that all external files be available.

When `cacheignoresfilecontents=true`, this check is disabled. As a result, it is possible to use `frozencache` without having access to the original external files. The files accessed by commands like `\inputminted` are no longer required; only the cache is needed. However, it is also no longer possible to verify that the cache is still valid.

`cacheignoresfilecontents` should be avoided when possible because it disables cache verification for external files. When `cacheignoresfilecontents` is used, it should be enabled as late as possible in the document preparation process.

1. Finalize the document (at least as far as minted is concerned).
2. Set `cacheignoresfilecontents=true` (while `frozencache=false`).
3. Compile. This is necessary to update the cache so that it is usable without access to the original external files.
4. Set `frozencache=true`.

As long as `cacheignoresfilecontents=true`, the document will compile without minted errors or warnings even if the external files are modified or no longer exist.

`debug=⟨boolean⟩` (default: false)

Provide additional information for aid in debugging. This keeps temp files that are used in generating highlighted code and also writes additional information to the log.

`frozencache=⟨boolean⟩` (default: false)

Use a frozen (static) cache. When `frozencache=true`, Python and Pygments are no longer required to compile a document. If a cache file is missing, an error will be reported and there will be no attempt to generate the missing cache file.

By default, any external files accessed through commands like `\inputminted` are still required, so that the cache can be checked against file contents. To disable this, see the package option `cacheignoresfilecontents`.

When using `frozencache` 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⟩` (default: fastfirst)

Determines when code is highlighted. This only has an effect when `cache=true`.

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`, `.tcbtemp`, `.temp`, `.tempfile`, `.tmp`, `.verb`, and `.vrb`. These extensions are correctly identified for files with double file extensions (and in that case the order of extensions doesn't matter).

For temp files with other file extensions, there are two options. Additional file extensions can be registered with `\MintedRegisterTempFileExtension{`⟨*ext*⟩`}`, where ⟨*ext*⟩ is the literal file extension (no macros) including the leading period (for example, "`.tmp`"). Otherwise, `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*⟩
`(default: false)`

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

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*⟩
`(default: false)`

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

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:

```
\begin{minted}[linenos=true]{c++}
#include <iostream>
int main() {
    std::cout << "Hello "
              << "world"
              << std::endl;
}
\end{minted}
```

```
1 #include <iostream>
2 int main() {
3     std::cout << "Hello "
4               << "world"
5               << std::endl;
6 }
```

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

```
\mint[linenos]{perl}|$x=~/foo/|
```

```
1 $x=~/foo/
```

Here's another example: we want to use the LaTeX math mode inside comments:

```
\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}
```

```
# Returns ∑ⁿᵢ₌₁ i
def sum_from_one_to(n):
    r = range(1, n + 1)
    return sum(r)
```

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).

```
\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}
```

```
␣␣␣␣def␣boring(args␣=␣None):
␣␣␣␣␣␣␣␣pass
```

versus

```
def␣boring(args␣=␣None):
␣␣␣␣pass
```

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

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

7.3 Command and environment options

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

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

```
...text.
\begin{minted}[autogobble]{py}
    def f(x):
        return x**2
\end{minted}
```

```
...text.

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

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.

`baseline` (b|c|t) (default: `b`)
Position of the baseline for fancyvrb's `BVerbatim` environment.

`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: `\,\footnotesize\ensuremath{_\rfloor}`, ⌋)
The symbol inserted pre-break for breaks inserted by `breakafter`.

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

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

```
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNever⌋
 ↪  FitOnOneLine'
```

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

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

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

`breakanywheresymbolpre` (string) (default: `\,\footnotesize\ensuremath{_\rfloor}`, ⌋)
The symbol inserted pre-break for breaks inserted by `breakanywhere`.

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

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

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

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

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

```
some_string = 'SomeTextThatGoesOn⌋
 ↪  AndOnForSoLongThatItCouldNeverFitOnOneLine'
```

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

`breakbeforesymbolpost` (string) (default: ⟨*none*⟩)
The symbol inserted post-break for breaks inserted by `breakbefore`.

`breakbeforesymbolpre` (string) (default: `\,\footnotesize\ensuremath{_\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 LaTeX and then insert a manual break is also an option. For example, use escapeinside=||, and then insert |\\| at the appropriate point. (Note that escapeinside does not work within strings.)

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

...text.

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

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

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

```
\begin{minted}[breaklines,
               breakautoindent=false,
               breaksymbolleft=\raisebox{0.8ex}{
                 \small\reflectbox{\carriagereturn}},
               breaksymbolindentleft=0pt,
               breaksymbolsepleft=0pt,
               breaksymbolright=\small\carriagereturn,
               breaksymbolindentright=0pt,
               breaksymbolsepright=0pt]{python}
def f(x):
    return 'Some text ' + str(x) + ' some more text ' +
     ↪  str(x) + ' even more text that goes on for a while'
\end{minted}
```

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

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

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

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

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

breaksymbolindent (dimension) (default: ⟨*breaksymbolindentleftnchars*⟩)
Alias for breaksymbolindentleft.

breaksymbolindentnchars (integer) (default: ⟨*breaksymbolindentleftnchars*⟩)
Alias for breaksymbolindentleftnchars.

breaksymbolindentleft (dimension) (default: ⟨*breaksymbolindentleftnchars*⟩)
The extra left indentation that is provided to make room for breaksymbolleft. This indentation is only applied when there is a breaksymbolleft.

Does not apply to \mintinline.

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

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

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

breaksymbolleft (string) (default: \tiny\ensuremath{\hookrightarrow}, ↪)
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 LaTeX between the two characters specified. All code between the two characters will be interpreted as LaTeX 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 LaTeX 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" LaTeX 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.

```
\setminted{escapeinside=||}
\begin{minted}{py}
def f(x):
    y = x|\colorbox{green}{**}|2
    return y
\end{minted}
```

```
def f(x):
    y = x ** 2
    return y
```

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.

extrakeywords (string) (default: ⟨*none*⟩)

extrakeywordsconstant (string) (default: ⟨*none*⟩)

extrakeywordsdeclaration (string) (default: ⟨*none*⟩)

extrakeywordsnamespace (string) (default: ⟨*none*⟩)

extrakeywordspseudo (string) (default: ⟨*none*⟩)

extrakeywordsreserved (string) (default: ⟨*none*⟩)

extrakeywordstype (string) (default: ⟨*none*⟩)
Extra keywords for the current lexer. See the Pygments documentation for details about the different keyword classes.

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

firstline (integer) (default: 1)
The first line to be shown. All lines before that line are ignored and do not appear in the output.

firstnumber (auto | last | integer) (default: auto = 1)
Line number of the first line.

fontencoding (font encoding) (default: ⟨*doc encoding*⟩)
Set font encoding used for typesetting code. For example, fontencoding=T1.

fontfamily (family name) (default: tt)
The font family to use. tt, courier and helvetica are pre-defined.

fontseries (series name) (default: auto – the same as the current font)
The font series to use.

fontshape (font shape) (default: auto – the same as the current font)
The font shape to use.

fontsize (font size) (default: auto – the same as the current font)
The size of the font to use, as a size command, e.g. \footnotesize.

formatcom (command) (default: ⟨*none*⟩)
A format to execute before printing verbatim text.

frame (none | leftline | topline | bottomline | lines | single) (default: none)
The type of frame to put around the source code listing. For more sophisticated framing, consider tcolorbox.

framerule (dimension) (default: 0.4pt)
Width of the frame.

framesep (dimension) (default: \fboxsep)
Distance between frame and content.

funcnamehighlighting (boolean) (default: true)
[For PHP only] If true, highlights built-in function names.

gobble (integer) (default: 0)
Remove the first n characters from each input line.

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

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

`gobblefilter` (integer) (default: 0)

Remove the first n characters from each input line, using the Pygments gobble filter.

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

`highlightcolor` (string) (default: `LightCyan`)

Set the color used for `highlightlines`, using a predefined color name from color or xcolor, or a color defined via `\definecolor`.

`highlightlines` (string) (default: ⟨*none*⟩)

This highlights a single line or a range of lines based on line numbers. For example, `highlightlines={1, 3-4}`. The line numbers refer to the line numbers that would appear if `linenos=true`, etc. They do not refer to original or actual line numbers before adjustment by `firstnumber`.

The highlighting color can be customized with `highlightcolor`.

`ignorelexererrors` (boolean) (default: `false`)

When lexer errors are shown in highlighted output (default), they are typically displayed as red boxes that surround the relevant text. When lexer errors are ignored, the literal text that caused lexer errors is shown but there is no indication that it caused errors.

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

```
variable = !!!
```

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

```
variable = !!!
```

`keywordcase` (lower | upper | capitalize | none) (default: `none`)

Changes capitalization of keywords.

`label` (string) (default: *empty*)

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

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

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

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

```
\renewcommand{\theFancyVerbLine}{
  \sffamily
  \textcolor[rgb]{0.5,0.5,1.0}{
    \scriptsize\oldstylenums{
      \arabic{FancyVerbLine}}}}

\begin{minted}[linenos,
  firstnumber=11]{python}
def all(iterable):
    for i in iterable:
        if not i:
            return False
    return True
\end{minted}
```

```
11 def all(iterable):
12     for i in iterable:
13         if not i:
14             return False
15     return True
```

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.

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

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

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

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

`rangestartafterstringline` (single macro or backslash-escaped string) (default: ⟨*none*⟩)
Select code starting with the line after this string is found. The selection starts at the beginning of the line.

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

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

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

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

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

`rangestartstringline` (single macro or backslash-escaped string) (default: ⟨*none*⟩)
Select code starting with the line where this string is found. The selection starts at the beginning of the line.

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

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

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

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

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

`rangestopbeforestringline` (single macro or backslash-escaped string) (default: ⟨*none*⟩)
Select code ending with the line before this string is found. The selection ends at the end of the line.

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

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

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

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

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

`rangestopstringline` (single macro or backslash-escaped string) (default: ⟨*none*⟩)
Select code ending with the line where this string is found. The selection ends at the end of the line.

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.

`reflabel` (string) (default: ⟨*none*⟩)
fancyvrb option for creating a label that can be used with `\pageref`.

`resetmargins` (boolean) (default: `false`)
Resets the left margin inside other environments.

`rulecolor` (color command) (default: `black`)
The color of the frame.

`samepage` (boolean) (default: `false`)
Forces the whole listing to appear on the same page, even if it doesn't fit.

`showspaces` (boolean) (default: `false`)
Enables visible spaces: `visible␣spaces`.

`showtabs` (boolean) (default: `false`)
Enables visible tabs—only works in combination with `obeytabs`.

`space` (macro) (default: `\textvisiblespace`, ␣)
Redefine the visible space character. Note that this is only used if `showspaces=true`.

`spacecolor` (string) (default: `none`)
Set the color of visible spaces. By default (`none`), they take the color of their surroundings.

`startinline` (boolean) (default: `false`)
[For PHP only] Specifies that the code starts in PHP mode, i.e., leading `<?php` is omitted.

`stepnumber` (integer) (default: `1`)
Interval at which line numbers appear.

`stepnumberfromfirst` (boolean) (default: `false`)

By default, when line numbering is used with `stepnumber` $\neq 1$, only line numbers that are a multiple of `stepnumber` 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 `stepnumber`, are numbered.

`stepnumberoffsetvalues` (boolean) (default: `false`)
By default, when line numbering is used with `stepnumber` $\neq 1$, only line numbers that are a multiple of `stepnumber` are included. Using `firstnumber` to offset the numbering will change which lines are numbered and which line gets which number, but will not change which *numbers* appear. This option causes `firstnumber` to be ignored in determining which line numbers are a multiple of `stepnumber`. `firstnumber` is still used in calculating the actual numbers that appear. As a result, the line numbers that appear will be a multiple of `stepnumber`, plus `firstnumber` minus 1.

`stripall` (boolean) (default: `false`)
Strip all leading and trailing whitespace from the input.

`stripnl` (boolean) (default: `false`)
Strip leading and trailing newlines from the input.

`style` (string) (default: ⟨*`default`*⟩)
Sets the stylesheet used by Pygments.

`tab` (macro) (default: fancyvrb's `\FancyVerbTab`, ⇥)
Redefine the visible tab character. Note that this is only used if `showtabs=true`. `\rightarrowfill`, ⟶, may be a nice alternative.

`tabcolor` (string) (default: `black`)
Set the color of visible tabs. If `tabcolor=none`, tabs take the color of their surroundings. This is typically undesirable for tabs that indent multiline comments or strings.

`tabsize` (integer) (default: 8)
The number of spaces a tab is equivalent to. If `obeytabs` is *not* active, tabs will be converted into this number of spaces. If `obeytabs` is active, tab stops will be set this number of space characters apart.

`texcl` (boolean) (default: `false`)
Enables LaTeX code inside comments. Usage as in package listings. See the note under `escapeinside` regarding math and ligatures.

`texcomments` (boolean) (default: `false`)
Enables LaTeX code inside comments. The newer name for `texcl`. See the note under `escapeinside` regarding math and ligatures.

`texcomments` fails with multiline C/C++ preprocessor directives, and may fail in some other circumstances. This is because preprocessor directives are tokenized as `Comment.Preproc`, so `texcomments` 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.

`tokenmerge` (bool) (default: `true`)
Merge adjacent tokens of the same type, using the Pygments `tokenmerge` filter.

`vspace` (dimension) (default: `\topsep`)
fancyvrb option for setting the value of the usual vertical list space.

`xleftmargin` (dimension) (default: 0)
Indentation to add before the listing.

`xrightmargin` (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:

```
\newminted{cpp}{gobble=2,linenos}

\begin{cppcode}
  template <typename T>
  T id(T value) {
      return value;
  }
\end{cppcode}
```

```
template <typename T>
T id(T value) {
    return value;
}
```

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

```
\newminted{cpp}{gobble=2,linenos}

\begin{cppcode}[linenos=false,
                frame=single]
  int const answer = 42;
\end{cppcode}
```

```
int const answer = 42;
```

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.

```
\newmint{perl}{showspaces}
\perl/my $foo = $bar;/
```

```
my␣$foo␣=␣$bar;
```

`\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`.

```
\newmintinline{perl}{showspaces}
\perlinline/my $foo = $bar;/
```

```
my␣$foo␣=␣$bar;
```

`\newmintedfile` This creates custom versions of `\inputminted`. The syntax is

`\newmintedfile[`⟨*macro name*⟩`]{`⟨*lexer*⟩`}{`⟨*options*⟩`}`

If no ⟨*macro name*⟩ is given, then the macro is called ⟨*lexer*⟩`file`.

9 FAQ and Troubleshooting

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

- **I am working with a console/REPL lexer, and there are missing characters, some characters appear in the wrong place, or highlighting is incorrect or incomplete.** Make sure that you have the correct lexer. For example, for Python there is `py` for code and `pycon` for console/REPL. Also try using `autogobble` if your code is indented. Some console/REPL lexers do not function correctly with indented code, at least under some versions of Pygments. References: #267, #388.

- **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 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 Notes for package authors

10.1 File extensions for temporary files

If your package highlights code using temporary files with a custom file extension, you may need to register the file extension for compatibility with the `highlightmode` package option. Use `\MintedRegisterTempFileExtension{`⟨*ext*⟩`}`, where ⟨*ext*⟩ is the literal file extension (no macros) including the leading period (for example, "`.tmp`"). There is built-in support for the following file extensions: `.listing`, `.out`, `.outfile`, `.output`, `.tcbtemp`, `.temp`, `.tempfile`, `.tmp`, `.verb`, and `.vrb`.

When a file extension is registered with `\MintedRegisterTempFileExtension`, it will be correctly identified for files with double file extensions (and in that case the order of extensions doesn't matter).

11 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.

12 Implementation

12.1 Exception handling

\minted@error

```
\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.`"

```
\def\minted@fatalerror#1{%
  \minted@error{#1}%
  \batchmode\read -1 to \minted@fatalerror@exitnow}
```

\minted@warning

```
\def\minted@warning#1{\PackageWarning{minted}{#1}}
```

12.2 Required packages

```
\RequirePackage{catchfile}
\RequirePackage{etoolbox}
\RequirePackage{fvextra}
\IfPackageAtLeastTF{fvextra}{2026/02/25}%
 {}{\minted@fatalerror{package fvextra is outdated; upgrade to the latest version}}
\RequirePackage{latex2pydata}
\IfPackageAtLeastTF{latex2pydata}{2026/02/25}%
 {}{\minted@fatalerror{package latex2pydata is outdated; upgrade to the latest version}}
\RequirePackage{pgfkeys}
\RequirePackage{pgfopts}
\RequirePackage{shellesc}
```

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

```
\AtEndPreamble{%
  \IfPackageLoadedTF{color}%
   {}%
   {\IfPackageLoadedTF{xcolor}{}{\RequirePackage{xcolor}}}}
```

12.3 LaTeX package version

\minted@sty@version

Store the current `minted.sty` version in `\minted@sty@version` (currently 3.8.0). This is passed to the Python side as part of compatibility checks.

```
\def\minted@def@sty@version{%
  \expandafter\let\expandafter\minted@tmp\csname ver@minted.sty\endcsname
  \edef\minted@tmp{\detokenize\expandafter{\minted@tmp}}%
  \expandafter\minted@def@sty@Version@i\minted@tmp\relax}
```

```
\begingroup
\catcode`\.=12
\catcode`\v=12
\gdef\minted@def@sty@Version@i#1v#2.#3.#4\relax{%
  \def\minted@sty@Version{#2.#3.}%
  \minted@def@sty@Version@ii#4\relax}
\endgroup
\def\minted@def@sty@Version@ii#1#2\relax{%
  \ifnum\number`#1<48\relax
    \expandafter\@secondoftwo
  \else
    \ifnum\number`#1>57\relax
      \expandafter\expandafter\expandafter\@secondoftwo
    \else
      \expandafter\expandafter\expandafter\@firstofone
    \fi
  \fi
   {\expandafter\def\expandafter\minted@sty@Version\expandafter{\minted@sty@Version#1}%
    \minted@def@sty@Version@ii#2\relax}%
   {\let\minted@sty@version\minted@sty@Version
    \let\minted@sty@Version\minted@undefined}}
\minted@def@sty@version
```

12.4 Python executable and minimum supported version

\MintedExecutable

Name of minted Python executable.

```
\edef\MintedExecutable{\detokenize{latexminted}}
```

\minted@executable@minversion

```
\edef\minted@executable@minversion{\detokenize{0.7.0}}
```

\minted@executable@minmajor
\minted@executable@minminor
\minted@executable@minpatch

```
\def\minted@executable@setminversioncomponents{%
  \expandafter\minted@executable@setminversioncomponents@i
    \minted@executable@minversion\relax}
\begingroup
\catcode`\.=12
\gdef\minted@executable@setminversioncomponents@i#1.#2.#3\relax{%
  \def\minted@executable@minmajor{#1}%
  \def\minted@executable@minminor{#2}%
  \def\minted@executable@minpatch{#3}}
\endgroup
\minted@executable@setminversioncomponents
```

\minted@executable@version

```
\let\minted@executable@version\relax
```

minted@executable@exists

```
\newbool{minted@executable@exists}
```

minted@executable@issupported

```
\newbool{minted@executable@issupported}
```

12.5 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

```
\begingroup
\catcode`\-=12
\catcode`\+=12
\catcode`\:=12
\def\minted@creationdatetotimestamp#1{%
  \expandafter\minted@creationdatetotimestamp@i#1-\relax}
\def\minted@creationdatetotimestamp@i#1:#2-#3\relax{%
  \minted@creationdatetotimestamp@ii#2+\relax}
\def\minted@creationdatetotimestamp@ii#1+#2\relax{%
  #1}
\expandafter\ifx\csname pdftexversion\endcsname\relax
\else
  \xdef\minted@timestamp{\minted@creationdatetotimestamp{\pdfcreationdate}}
\fi
\expandafter\ifx\csname XeTeXrevision\endcsname\relax
\else
  \xdef\minted@timestamp{\minted@creationdatetotimestamp{\creationdate}}
\fi
\expandafter\ifx\csname directlua\endcsname\relax
\else
  \begingroup
  \outputmode=1
  \xdef\minted@timestamp{\minted@creationdatetotimestamp{\pdffeedback creationdate}}%
  \endgroup
\fi
\endgroup
\ifcsname minted@timestamp\endcsname
\else
  \begingroup
  \newcounter{minted@timestamp@hr}
  \newcounter{minted@timestamp@min}
  \setcounter{minted@timestamp@min}{\number\time}
  \loop\unless\ifnum\value{minted@timestamp@min}<60\relax
    \addtocounter{minted@timestamp@hr}{1}
    \addtocounter{minted@timestamp@min}{-60}
  \repeat
  \xdef\minted@timestamp{%
    \the\year
    \ifnum\month<10 0\fi\the\month
    \ifnum\day<10 0\fi\the\day
    \ifnum\value{minted@timestamp@hr}<10 0\fi\theminted@timestamp@hr
    \ifnum\value{minted@timestamp@min}<10 0\fi\theminted@timestamp@min}
  \endgroup
\fi
\xdef\minted@timestamp{\detokenize\expandafter{\minted@timestamp}}
```

12.6 Jobname MD5 and derived file names

\minted@mdfivehash
\minted@filemdfivehash

Wrappers for MD5 macros.

```
\expandafter\let\expandafter\minted@mdfivehash\csname str_mdfive_hash:e\endcsname
\expandafter\let\expandafter\minted@filemdfivehash\csname file_mdfive_hash:n\endcsname
```

\MintedJobnameMdfive

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

`\MintedJobnameMdfive` is used for creating temp files rather than `\jobname` to avoid shell escaping issues. Under restricted shell escape, shell commands are quoted and escaped by LaTeX itself, so using `\jobname` would work correctly in most cases. However, when full shell escape is enabled, no command escaping is performed by LaTeX, so minted would have to quote/escape `\jobname` in a platform-specific manner. (See for example `web2c.info` and `texmfmp.c` in the TeX Live source for shell escape implementation details.) It is simpler to avoid escaping issues altogether, including edge cases in the restricted shell escape scenario, by using an MD5 hash that is guaranteed to consist only of ASCII alphanumeric characters.

```
\begingroup
\catcode`\"=12
\catcode`\'=12
\gdef\minted@setjobnamemdfive#1#2\FV@Sentinel{%
  \ifx#1"\relax
    \let\minted@next\minted@setjobnamemdfive@dquoted
  \else\ifx#1'\relax
    \let\minted@next\minted@setjobnamemdfive@squoted
  \else
    \let\minted@next\minted@setjobnamemdfive@uquoted
  \fi\fi
  \minted@next#1#2\FV@Sentinel}
\gdef\minted@setjobnamemdfive@dquoted#1#2\FV@Sentinel{%
  \minted@setjobnamemdfive@dquoted@i#2"\FV@Sentinel}
\gdef\minted@setjobnamemdfive@dquoted@i#1"#2\FV@Sentinel{%
  \if\relax\detokenize{#2}\relax
    \edef\MintedJobnameMdfive{\minted@mdfivehash{\jobname}}%
  \else\if\relax\detokenize\expandafter{\@gobble#2}\relax
    \edef\MintedJobnameMdfive{\minted@mdfivehash{#1}}%
  \else
    \edef\MintedJobnameMdfive{\minted@mdfivehash{\jobname}}%
  \fi\fi}
\gdef\minted@setjobnamemdfive@squoted#1#2\FV@Sentinel{%
  \minted@setjobnamemdfive@squoted@i#2'\FV@Sentinel}
\gdef\minted@setjobnamemdfive@squoted@i#1'#2\FV@Sentinel{%
  \if\relax\detokenize{#2}\relax
    \edef\MintedJobnameMdfive{\minted@mdfivehash{\jobname}}%
  \else\if\relax\detokenize\expandafter{\@gobble#2}\relax
    \edef\MintedJobnameMdfive{\minted@mdfivehash{#1}}%
  \else
    \edef\MintedJobnameMdfive{\minted@mdfivehash{\jobname}}%
  \fi\fi}
\gdef\minted@setjobnamemdfive@uquoted#1\FV@Sentinel{%
  \edef\MintedJobnameMdfive{\minted@mdfivehash{#1}}}
\endgroup
\expandafter\minted@setjobnamemdfive\jobname\FV@Sentinel
```

\MintedCacheIndexFilename

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

```
\edef\MintedCacheIndexFilename{%
  \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.

```
\edef\MintedConfigFilename{%
  \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.

```
\edef\MintedDataFilename{%
  \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.

```
\edef\MintedErrlogFilename{%
  \detokenize{_}\MintedJobnameMdfive\detokenize{.errlog.minted}}
```

\MintedMessageFilename

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

```
\edef\MintedMessageFilename{%
  \detokenize{_}\MintedJobnameMdfive\detokenize{.message.minted}}
```

12.7 Package options

\minted@pgfopts

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

12.7.1 Package option definitions

\minted@float@within

Control the section numbering of the `listing` float.

```
\minted@pgfopts{
  chapter/.code=\def\minted@float@within{chapter},
  chapter/.value forbidden,
  section/.code=\def\minted@float@within{section},
  section/.value forbidden,
}
```

minted@newfloat

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

```
\newbool{minted@newfloat}
```

```
\minted@pgfopts{
  newfloat/.is if=minted@newfloat,
}
```

minted@debug

Keep temp files for aid in debugging.

```
\newbool{minted@debug}
\minted@pgfopts{
  debug/.is if=minted@debug,
}
```

minted@cache

Determine whether highlighted content is cached.

```
\newbool{minted@cache}
\booltrue{minted@cache}
\minted@pgfopts{
  cache/.is if=minted@cache,
}
```

\minted@cachedir

Set the directory in which cached content is saved.

```
\edef\minted@cachedir{\detokenize{_minted}}
\minted@pgfopts{
  cachedir/.estore in=\minted@cachedir,
}
```

minted@cacheignoresfilecontents

When caching highlighted versions of external files, ignore file contents. This allows frozencache to function without external files being accessible, but at the cost of not being able to verify file contents.

```
\newbool{minted@cacheignoresfilecontents}
\minted@pgfopts{
  cacheignoresfilecontents/.is if=minted@cacheignoresfilecontents,
}
```

minted@frozencache

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.

```
\newbool{minted@frozencache}
\minted@pgfopts{
  frozencache/.is if=minted@frozencache,
}
```

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.

```
\newbool{minted@lexerlinenos}
\minted@pgfopts{
  lexerlinenos/.is if=minted@lexerlinenos,
  langlinenos/.is if=minted@lexerlinenos,
}
```

minted@inputlexerlinenos

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

```
\newbool{minted@inputlexerlinenos}
\minted@pgfopts{
  inputlexerlinenos/.is if=minted@inputlexerlinenos,
  inputlanglinenos/.is if=minted@inputlexerlinenos,
}
```

`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/Ti*k*Z externalization, when all non-PGF/Ti*k*Z features should be disabled.

```
\newbool{minted@placeholder}
\minted@pgfopts{
  placeholder/.is if=minted@placeholder,
}
\gdef\minted@insertplaceholder{%
  \ifbool{minted@isinline}%
   {\begingroup
    \fvset{extra=true}\Verb[formatcom=\color{red}\bfseries]{<MINTED>}%
    \endgroup}%
   {\begingroup
    \par\noindent
    \fvset{extra=true}\Verb[formatcom=\color{red}\bfseries]{<MINTED>}%
    \par
    \endgroup}}%
```

`minted@verbatim`

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

```
\newbool{minted@verbatim}
\minted@pgfopts{
  verbatim/.is if=minted@verbatim,
}
```

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

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

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

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

```
\newbool{minted@fasthighlightmode}
\newbool{minted@fasthighlightmode@open}
\minted@pgfopts{
  highlightmode/.is choice,
  highlightmode/fast/.code=
    \let\minted@highlightmode@init\minted@highlightmode@init@fast,
  highlightmode/fastfirst/.code=
    \let\minted@highlightmode@init\minted@highlightmode@init@fastfirst,
  highlightmode/immediate/.code=
    \let\minted@highlightmode@init\minted@highlightmode@init@immediate,
}
\def\minted@highlightmode@init@fast{%
  \global\booltrue{minted@fasthighlightmode}}
\def\minted@highlightmode@init@fastfirst{%
  \IfFileExists{\minted@cachepath\MintedCacheIndexFilename}%
   {\global\boolfalse{minted@fasthighlightmode}}
   {\global\booltrue{minted@fasthighlightmode}}}
\def\minted@highlightmode@init@immediate{%
  \global\boolfalse{minted@fasthighlightmode}}
\let\minted@highlightmode@init\minted@highlightmode@init@fastfirst
\def\minted@fasthighlightmode@checkstart{%
  \ifbool{minted@fasthighlightmode}%
   {\pydatawritelistopen
    \global\booltrue{minted@fasthighlightmode@open}}%
   {}%
  \global\let\minted@fasthighlightmode@checkstart\relax}
\def\minted@fasthighlightmode@checkend{%
  \ifbool{minted@fasthighlightmode@open}%
   {\pydatasetfilename{\MintedDataFilename}%
    \pydatawritelistclose
    \pydataclosefilename{\MintedDataFilename}%
    \global\boolfalse{minted@fasthighlightmode@open}%
    \global\boolfalse{minted@fasthighlightmode}%
    \begingroup
    \minted@exec@batch
    \ifx\minted@exec@warning\relax
    \else
      \expandafter\minted@exec@warning
    \fi
    \ifx\minted@exec@error\relax
    \else
      \expandafter\minted@exec@error
    \fi
    \endgroup
    \global\boolfalse{minted@canexec}}%
   {}%
  \global\let\minted@fasthighlightmode@checkend\relax}
```

12.7.2 Package options that are no longer supported or deprecated

finalizecache Old, no longer needed option from minted v2.

```
\minted@pgfopts{
  finalizecache/.code=\minted@error{%
    Package option "finalizecache" is no longer needed with minted v3+},
}
```

kpsewhich Old, no longer needed option from minted v2.

```
\minted@pgfopts{
  kpsewhich/.code=\minted@error{%
```

```
    Package option "kpsewhich" is no longer needed with minted v3+},
}
```

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.

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

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

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.

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

12.7.3 Package option processing

```
\ProcessPgfOptions{/minted/pkg}
\ifbool{minted@cache}
 {}
 {\minted@pgfopts{
    cachedir=,
    cacheignoresfilecontents=false,
    highlightmode=immediate,
    frozencache=false,
 }}
\ifbool{minted@newfloat}{\RequirePackage{newfloat}}{\RequirePackage{float}}
\ifcsname tikzifexternalizing\endcsname
  \ifx\tikzifexternalizing\relax
  \else
    \tikzifexternalizing{\booltrue{minted@placeholder}}{}
  \fi
\fi
```

12.8 Util

`\minted@styleprefix`

Prefix for generating Pygments style names.

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

\minted@tempindex

Temp index for looping.

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

\minted@forcsvlist

Wrapper for etoolbox `\forcsvlist`. Syntax: `\minted@forcsvlist{`⟨*handler*⟩`}{`⟨*listmacro*⟩`}`.

```
\def\minted@forcsvlist#1#2{%
  \if\relax\detokenize\expandafter{\@gobble#2}\relax
    \expandafter\minted@forcsvlist@exp
  \else
    \expandafter\minted@forcsvlist@i
  \fi
  {#2}{#1}}
\def\minted@forcsvlist@exp#1#2{%
  \expandafter\minted@forcsvlist@i\expandafter{#1}{#2}}
\def\minted@forcsvlist@i#1#2{%
  \forcsvlist{#2}{#1}}
```

\minted@apptoprovidecs

```
\def\minted@apptoprovidecs#1#2{%
  \ifcsname#1\endcsname
  \else
    \expandafter\def\csname#1\endcsname{}%
  \fi
  \expandafter\let\expandafter\minted@tmp\csname#1\endcsname
  \expandafter\def\expandafter\minted@tmp\expandafter{\minted@tmp#2}%
  \expandafter\let\csname#1\endcsname\minted@tmp}
```

\minted@const@pgfkeysnovalue

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

\minted@ensureatletter

```
\def\minted@ensureatletter#1{%
  \edef\minted@tmpatcat{\the\catcode`\@}%
  \catcode`\@=11\relax
  #1%
  \catcode`\@=\minted@tmpatcat\relax}
```

12.8.1 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.

```
% [0-9]
\def\minted@tempindex{48}
\loop\unless\ifnum\minted@tempindex>57\relax
  \expandafter\let\csname minted@isnum\minted@tempindex\endcsname\relax
  \expandafter\let\csname minted@isalphanum\minted@tempindex\endcsname\relax
  \expandafter\let
    \csname minted@isalphanumhyphenunderscore\minted@tempindex\endcsname\relax
  \edef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}
\repeat
% [A-Z]
\def\minted@tempindex{65}
\loop\unless\ifnum\minted@tempindex>90\relax
  \expandafter\let\csname minted@isalpha\minted@tempindex\endcsname\relax
  \expandafter\let\csname minted@isalphanum\minted@tempindex\endcsname\relax
  \expandafter\let
    \csname minted@isalphanumhyphenunderscore\minted@tempindex\endcsname\relax
  \edef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}
\repeat
% [a-z]
\def\minted@tempindex{97}
\loop\unless\ifnum\minted@tempindex>122\relax
  \expandafter\let\csname minted@isalpha\minted@tempindex\endcsname\relax
  \expandafter\let\csname minted@isalphanum\minted@tempindex\endcsname\relax
  \expandafter\let
    \csname minted@isalphanumhyphenunderscore\minted@tempindex\endcsname\relax
  \edef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}
\repeat
% [-]
\expandafter\let\csname minted@isalphanumhyphenunderscore45\endcsname\relax
% [_]
\expandafter\let\csname minted@isalphanumhyphenunderscore95\endcsname\relax
```

\minted@ifalphanumhyphenunderscore

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

```
\def\minted@ifalphanumhyphenunderscore#1#2#3{%
  \if\relax\detokenize{#1}\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
  {#3}%
  {\expandafter\minted@ifalphanumhyphenunderscore@i\detokenize{#1}\FV@Sentinel{#2}{#3}}}
\def\minted@ifalphanumhyphenunderscore@i#1#2\FV@Sentinel{%
  \if\relax#2\relax
    \expandafter\minted@ifalphanumhyphenunderscore@iii
  \else
    \expandafter\minted@ifalphanumhyphenunderscore@ii
  \fi
  #1#2\FV@Sentinel}
\def\minted@ifalphanumhyphenunderscore@ii#1#2\FV@Sentinel{%
  \ifcsname minted@isalphanumhyphenunderscore\number`#1\endcsname
    \expandafter\minted@ifalphanumhyphenunderscore@i
  \else
    \expandafter\minted@ifalphanumhyphenunderscore@false
  \fi
  #2\FV@Sentinel}
\def\minted@ifalphanumhyphenunderscore@iii#1\FV@Sentinel{%
  \ifcsname minted@isalphanumhyphenunderscore\number`#1\endcsname
    \expandafter\minted@ifalphanumhyphenunderscore@true
  \else
```

```
    \expandafter\minted@ifalphanumhyphenunderscore@false
  \fi
  \FV@Sentinel}
\def\minted@ifalphanumhyphenunderscore@true\FV@Sentinel#1#2{#1}
\def\minted@ifalphanumhyphenunderscore@false#1\FV@Sentinel#2#3{#3}
```

12.9 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.

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

minted@isinline

Whether in command or environment for typesetting. This is not related to whether a command or environment initiates typesetting (see minted@iscmd); for example, \inputminted is a command that causes typesetting within an environment.

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

minted@iscmd

Whether a command or environment initiates typesetting.

```
\newbool{minted@iscmd}
```

minted@tmpcodebufferlength

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

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

12.10 Calling minted executable

minted@canexec

```
\newbool{minted@canexec}
\booltrue{minted@canexec}
\ifnum\csname c_sys_shell_escape_int\endcsname=0\relax
  \boolfalse{minted@canexec}
\fi
\ifbool{minted@frozencache}{\boolfalse{minted@canexec}}{}
\ifbool{minted@placeholder}{\boolfalse{minted@canexec}}{}
\ifbool{minted@verbatim}{\boolfalse{minted@canexec}}{}
```

\minted@ShellEscapeMaybeMessages
\minted@ShellEscapeNoMessages

```
\def\minted@ShellEscapeMaybeMessages#1{%
  \ifbool{minted@debug}%
   {\immediate\typeout{%
      minted debug: shell escape at
      \ifx\CurrentFile\@empty\else\CurrentFile\space\fi line \the\inputlineno: #1}}%
   {}%
  \let\minted@exec@warning\relax
  \let\minted@exec@error\relax
  \ifbool{minted@canexec}{\ShellEscape{#1}\minted@inputexecmessages}{}}
\def\minted@ShellEscapeNoMessages#1{%
  \ifbool{minted@debug}%
   {\immediate\typeout{%
```

```
      minted debug: shell escape at
      \ifx\CurrentFile\@empty\else\CurrentFile\space\fi line \the\inputlineno: #1}}%
   {}%
  \ifbool{minted@canexec}{\ShellEscape{#1}}{}}
```

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

```
\def\minted@execarg@debug{%
  \ifbool{minted@debug}{\detokenize{ --debug }}{}}
\def\minted@execarg@timestamp{%
  \detokenize{ --timestamp }\minted@timestamp\detokenize{ }}
```

\minted@inputexecmessages

If temp file containing warning and/or error messages exists, \input and then delete.

```
\def\minted@inputexecmessages{%
  \minted@ensureatletter{\InputIfFileExists{\MintedMessageFilename}{}{}}}
```

\minted@exec@batch

Run in batch mode, for highlightmode=fast or highlightmode=fastfirst.

```
\def\minted@exec@batch{%
  \minted@ShellEscapeMaybeMessages{%
    \MintedExecutable
    \detokenize{ batch }\minted@execarg@timestamp\minted@execarg@debug
    \MintedJobnameMdfive}}
```

\minted@exec@config

Detect configuration.

```
\def\minted@exec@config{%
  \minted@ShellEscapeMaybeMessages{%
    \MintedExecutable
    \detokenize{ config }\minted@execarg@timestamp\minted@execarg@debug
    \MintedJobnameMdfive}}
```

\minted@exec@styledef

Create style definition.

```
\def\minted@exec@styledef{%
  \minted@ShellEscapeMaybeMessages{%
    \MintedExecutable
    \detokenize{ styledef }\minted@execarg@timestamp\minted@execarg@debug
    \MintedJobnameMdfive}}
```

\minted@exec@highlight

Highlight code.

```
\def\minted@exec@highlight{%
  \minted@ShellEscapeMaybeMessages{%
    \MintedExecutable
    \detokenize{ highlight }\minted@execarg@timestamp\minted@execarg@debug
    \MintedJobnameMdfive}}
```

\minted@exec@clean

Clean temp files and cache.

```
\def\minted@exec@clean{%
  \minted@ShellEscapeNoMessages{%
    \MintedExecutable
    \detokenize{ clean }\minted@execarg@timestamp\minted@execarg@debug
    \MintedJobnameMdfive}}
```

\minted@exec@cleanconfig

Clean config temp file.

```
\def\minted@exec@cleanconfig{%
  \minted@ShellEscapeNoMessages{%
    \MintedExecutable
    \detokenize{ cleanconfig }\minted@execarg@timestamp\minted@execarg@debug
    \MintedJobnameMdfive}}
```

\minted@exec@cleantemp

Clean all temp files.

```
\def\minted@exec@cleantemp{%
  \minted@ShellEscapeNoMessages{%
    \MintedExecutable
    \detokenize{ cleantemp }\minted@execarg@timestamp\minted@execarg@debug
    \MintedJobnameMdfive}}
```

12.11 Config detection

minted@diddetectconfig

```
\newbool{minted@diddetectconfig}
```

\minted@detectconfig

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

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

```
\def\minted@detectconfig{%
  \ifbool{minted@diddetectconfig}%
   {}%
   {\ifx\minted@cachedir\@empty
      \gdef\minted@cachepath{}%
    \else
      \gdef\minted@cachepath{\minted@cachedir/}%
    \fi
    \ifbool{minted@canexec}{\begingroup\minted@detectconfig@i\endgroup}{}%
    \global\booltrue{minted@diddetectconfig}}}
\def\minted@detectconfig@i{%
  \global\let\minted@executable@version\relax
  \global\let\minted@executable@timestamp\relax
  \global\let\minted@config@timestamp\relax
  \pydatasetfilename{\MintedDataFilename}%
  \pydatawritedictopen
  \pydatawritekeyvalue{command}{config}%
  \pydatawritekeyedefvalue{mintedversion}{\minted@sty@version}%
  \pydatawritekeyedefvalue{jobname}{\jobname}%
  \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
  \pydatawritekeyedefvalue{cachedir}{\minted@cachedir}%
  \pydatawritedictclose
  \pydataclosefilename{\MintedDataFilename}%
  \minted@exec@config
  \minted@ensureatletter{%
    \InputIfFileExists{\MintedConfigFilename}{}{}}%
  \ifx\minted@executable@version\relax
    \expandafter\minted@detectconfig@noexecutableorerrlog
  \else
    \expandafter\minted@detectconfig@ii
  \fi}
\def\minted@detectconfig@noexecutableorerrlog{%
  \global\boolfalse{minted@canexec}%
  \ifnum\csname c_sys_shell_escape_int\endcsname=1\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
   {\IfFileExists{\MintedErrlogFilename}%
     {\minted@error{minted v3+ executable is not installed or is not added to PATH;
        or MiKTeX is being used with \detokenize{-aux-directory} or
        \detokenize{-output-directory} without setting a
        \detokenize{TEXMF_OUTPUT_DIRECTORY} environment variable;
        or there was an unexpected error (check "\MintedErrlogFilename")}}%
     {\minted@error{minted v3+ executable is not installed or is not added to PATH;
        or MiKTeX is being used with \detokenize{-aux-directory} or
        \detokenize{-output-directory} without setting a
        \detokenize{TEXMF_OUTPUT_DIRECTORY} environment variable}}}%
   {\IfFileExists{\MintedErrlogFilename}%
     {\minted@error{minted v3+ executable is not installed, is not added to PATH,
        or is not permitted with restricted shell escape;
        or MiKTeX is being used with \detokenize{-aux-directory} or
        \detokenize{-output-directory} without setting a
        \detokenize{TEXMF_OUTPUT_DIRECTORY} environment variable;
        or there was an unexpected error (check "\MintedErrlogFilename")}}%
     {\minted@error{minted v3+ executable is not installed, is not added to PATH,
        or is not permitted with restricted shell escape;
        or MiKTeX is being used with \detokenize{-aux-directory} or
        \detokenize{-output-directory} without setting a
        \detokenize{TEXMF_OUTPUT_DIRECTORY} environment variable}}}}
\def\minted@detectconfig@ii{%
  \ifx\minted@timestamp\minted@config@timestamp
    \expandafter\minted@detectconfig@iii
  \else
    \expandafter\minted@detectconfig@wrongtimestamp
  \fi}
\def\minted@detectconfig@wrongtimestamp{%
  \ifx\minted@timestamp\minted@executable@timestamp
    \minted@exec@cleanconfig
    \global\boolfalse{minted@canexec}%
    \minted@error{minted v3 Python executable could not find output directory;
```

```
      upgrade to TeX distribution that supports \detokenize{TEXMF_OUTPUT_DIRECTORY}
      or set environment variable \detokenize{TEXMF_OUTPUT_DIRECTORY} manually)}%
  \else
    \expandafter\minted@detectconfig@noexecutableorerrlog
  \fi}
\def\minted@detectconfig@iii{%
  \minted@exec@cleanconfig
  \ifx\minted@exec@warning\relax
  \else
    \expandafter\minted@exec@warning
  \fi
  \ifx\minted@exec@error\relax
    \expandafter\minted@detectconfig@iv
  \else
    \expandafter\minted@detectconfig@error
  \fi}
\def\minted@detectconfig@error{%
  \global\boolfalse{minted@canexec}%
  \minted@exec@error}
\def\minted@detectconfig@iv{%
  \expandafter\minted@detectconfig@v\minted@executable@version\relax}
\begingroup
\catcode`\.=12
\gdef\minted@detectconfig@v#1.#2.#3\relax{%
  \def\minted@executable@major{#1}%
  \def\minted@executable@minor{#2}%
  \def\minted@executable@patch{#3}%
  \minted@detectconfig@vi}
\endgroup
\def\minted@detectconfig@vi{%
  \ifnum\minted@executable@major>\minted@executable@minmajor\relax
    \global\booltrue{minted@executable@issupported}%
  \else\ifnum\minted@executable@major=\minted@executable@minmajor\relax
    \ifnum\minted@executable@minor>\minted@executable@minminor\relax
      \global\booltrue{minted@executable@issupported}%
    \else\ifnum\minted@executable@minor=\minted@executable@minminor\relax
      \ifnum\minted@executable@patch<\minted@executable@minpatch\relax
      \else
        \global\booltrue{minted@executable@issupported}%
      \fi
    \fi\fi
  \fi\fi
  \ifbool{minted@executable@issupported}%
   {\ifx\minted@config@cachepath\relax
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi
    {\global\boolfalse{minted@canexec}%
     \minted@error{minted Python executable returned incomplete configuration data;
       this may indicate a bug in minted or file corruption}}%
    {\global\let\minted@cachepath\minted@config@cachepath
     \minted@highlightmode@init
     \ifbool{minted@fasthighlightmode}{\newread\minted@intempfile}{}}}%
   {\global\boolfalse{minted@canexec}%
    \minted@error{minted Python executable is version \minted@executable@version,
      but version \minted@executable@minversion+ is required}}}
```

12.12 Options

12.12.1 Option processing

\minted@optcats
\minted@optkeyslist@<optcat>

Option categories, along with lists of keys for each.

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

```
\begingroup
\catcode`\,=12
\gdef\minted@optcats{fv,py,tex}
\endgroup
\def\minted@do#1{\expandafter\def\csname minted@optkeyslist@#1\endcsname{}}
\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.

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

\minted@iflexerscope

```
\let\minted@iflexerscope@lexer\relax
\let\minted@iflexerscope@lexerinline\relax
\def\minted@iflexerscope#1#2#3{%
  \ifcsname minted@iflexerscope@#1\endcsname
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
  {#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` (LaTeX 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.

```
\def\minted@addoptkey#1#2{%
  \ifcsname minted@optkeyslist@#1\endcsname
  \else
    \minted@fatalerror{Defining options under category "#1" is not supported}%
  \fi
  \expandafter\let\expandafter\minted@tmp\csname minted@optkeyslist@#1\endcsname
  \ifx\minted@tmp\@empty
    \def\minted@tmp{#2}%
  \else
    \expandafter\def\expandafter\minted@tmp\expandafter{\minted@tmp,#2}%
  \fi
  \expandafter\let\csname minted@optkeyslist@#1\endcsname\minted@tmp}
\newcommand*{\mintedpgfkeyscreate}[3][]{%
  \mintedpgfkeyscreate@i{#1}{#2}{#3}}
\begingroup
\catcode`\==12
\gdef\mintedpgfkeyscreate@i#1#2#3{%
  \def\minted@do##1{%
    \minted@do@i##1=\FV@Sentinel}%
  \def\minted@do@i##1=##2\FV@Sentinel{%
    \minted@do@ii##1<>\FV@Sentinel}%
  \def\minted@do@ii##1<##2>##3\FV@Sentinel{%
    \minted@addoptkey{#2}{##1}}%
  \minted@forcsvlist{\minted@do}{#3}%
  \csname minted@pgfkeyscreate@#2\endcsname{#1}{#3}}
\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.

```
\def\minted@pgfkeyscreate@fv#1#2{%
  \if\relax\detokenize{#1}\relax
  \else
    \minted@fatalerror{Processor macros are not supported in defining fancyvrb options}%
  \fi
  \minted@forcsvlist{\minted@pgfkeycreate@fv}{#2}}
\begingroup
\catcode`\==12
\gdef\minted@pgfkeycreate@fv#1{%
  \minted@pgfkeycreate@fv@i#1=\FV@Sentinel}
\gdef\minted@pgfkeycreate@fv@i#1=#2\FV@Sentinel{%
  \if\relax\detokenize{#2}\relax
    \expandafter\minted@pgfkeycreate@fv@ii
  \else
    \expandafter\minted@pgfkeycreate@fv@iii
  \fi
  {#1}#2\FV@Sentinel}
\gdef\minted@pgfkeycreate@fv@ii#1\FV@Sentinel{%
  \minted@pgfkeycreate@fv@iv{#1}{\minted@const@pgfkeysnovalue}}
\gdef\minted@pgfkeycreate@fv@iii#1#2=\FV@Sentinel{%
  \minted@pgfkeycreate@fv@iv{#1}{#2}}
\endgroup
\def\minted@pgfkeycreate@fv@iv#1#2{%
  \def\minted@do##1{%
    \minted@iflexerscope{##1}%
     {\minted@do@i{##1}{@\minted@lexer}}%
     {\minted@do@i{##1}{}}}%
  \def\minted@do@i##1##2{%
    \pgfkeys{%
      /minted/##1/.cd,
      #1/.code={%
        \def\minted@tmp{####1}%
        \ifx\minted@tmp\minted@const@pgfkeysnovalue
          \begingroup\fvset{#1}\endgroup
          \minted@apptoprovidecs{minted@fvoptlist@##1##2}{#1,}%
        \else
          \begingroup\fvset{#1={####1}}\endgroup
          \minted@apptoprovidecs{minted@fvoptlist@##1##2}{#1={####1},}%
        \fi},
    }%
  }%
  \minted@forcsvlist{\minted@do}{\minted@optscopes}%
  \ifx\minted@const@pgfkeysnovalue#2\relax
  \else
    \pgfkeys{%
      /minted/global/.cd,
      #1={#2},
    }%
  \fi}
\def\minted@usefvopts{%
  \ifbool{minted@isinline}%
   {\minted@forcsvlist{\minted@usefvopts@do}{\minted@optscopes}}%
   {\minted@forcsvlist{\minted@usefvopts@do}{\minted@optscopes@onlyblock}}}
\def\minted@usefvopts@do#1{%
  \minted@iflexerscope{#1}%
   {\ifcsname minted@fvoptlist@#1@\minted@lexer\endcsname
    \expandafter
      \let\expandafter\minted@tmp\csname minted@fvoptlist@#1@\minted@lexer\endcsname
    \expandafter\fvset\expandafter{\minted@tmp}%
    \fi}%
   {\ifcsname minted@fvoptlist@#1\endcsname
    \expandafter
      \let\expandafter\minted@tmp\csname minted@fvoptlist@#1\endcsname
    \expandafter\fvset\expandafter{\minted@tmp}%
    \fi}}
\def\minted@useadditionalfvoptsnopy{%
  \edef\minted@tmp{\mintedpyoptvalueof{gobble}}%
  \ifx\minted@tmp\minted@const@pgfkeysnovalue
  \else
    \expandafter\minted@useadditionalfvoptsnopy@fvsetvk
      \expandafter{\minted@tmp}{gobble}%
  \fi
  \edef\minted@tmp{\mintedpyoptvalueof{mathescape}}%
  \ifx\minted@tmp\minted@const@pgfkeysnovalue
  \else
    \expandafter\minted@useadditionalfvoptsnopy@fvsetvk
      \expandafter{\minted@tmp}{mathescape}%
  \fi}
\def\minted@useadditionalfvoptsnopy@fvsetvk#1#2{%
  \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.

```
\def\minted@pgfkeyscreate@py#1#2{%
  \minted@forcsvlist{\minted@pgfkeycreate@py{#1}}{#2}}
\begingroup
```

```
\catcode`\==12
\catcode`\<=12
\catcode`\>=12
\gdef\minted@pgfkeycreate@py#1#2{%
  \minted@pgfkeycreate@py@i{#1}#2=\FV@Sentinel}
\gdef\minted@pgfkeycreate@py@i#1#2=#3\FV@Sentinel{%
  \if\relax\detokenize{#3}\relax
    \expandafter\minted@pgfkeycreate@py@ii
  \else
    \expandafter\minted@pgfkeycreate@py@iii
  \fi
  {#1}{#2}#3\FV@Sentinel}
\gdef\minted@pgfkeycreate@py@ii#1#2\FV@Sentinel{%
  \minted@pgfkeycreate@py@iv{#1}{\pgfkeysnovalue}#2<>\FV@Sentinel}
\gdef\minted@pgfkeycreate@py@iii#1#2#3=\FV@Sentinel{%
  \minted@pgfkeycreate@py@iv{#1}{#3}#2<>\FV@Sentinel}
\gdef\minted@pgfkeycreate@py@iv#1#2#3<#4>#5\FV@Sentinel{%
  \if\relax\detokenize{#4}\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
  {\minted@pgfkeycreate@py@v{#1}{#3}{#2}{\minted@const@pgfkeysnovalue}}%
  {\minted@pgfkeycreate@py@v{#1}{#3}{#2}{#4}}}
\endgroup
\def\minted@pgfkeycreate@py@v#1#2#3#4{%
  \def\minted@do##1{%
    \minted@iflexerscope{##1}%
     {\minted@do@i{##1}{@\minted@lexer}}%
     {\minted@do@i{##1}{}}}
  \def\minted@do@i##1##2{%
    \if\relax\detokenize{#1}\relax
      \pgfkeys{%
        /minted/##1/.cd,
        #2/.code={\expandafter\def\csname minted@pyopt@##1##2@#2\endcsname{####1}},
      }%
    \else
      \pgfkeys{%
        /minted/##1/.cd,
        #2/.code={%
          \def\minted@tmp{####1}%
          \ifx\minted@tmp\minted@const@pgfkeysnovalue
            \expandafter\let\csname minted@pyopt@##1##2@#2\endcsname\minted@tmp
          \else\ifcsname minted@opthandler@immediate@\string#1\endcsname
            #1{minted@pyopt@##1##2@#2}{####1}%
          \else
            \expandafter\def\csname minted@pyopt@##1##2@#2\endcsname{#1{####1}}%
          \fi\fi},
      }%
    \fi
    \ifx\minted@const@pgfkeysnovalue#4\relax
      \pgfkeys{%
        /minted/##1/.cd,
        #2/.value required,
      }%
    \else
      \pgfkeys{%
        /minted/##1/.cd,
        #2/.default={#4},
      }%
    \fi
  }%
  \minted@forcsvlist{\minted@do}{\minted@optscopes}%
  \pgfkeys{%
    /minted/global/.cd,
    #2={#3},
  }}
\def\mintedpyoptvalueof#1{%
  \ifbool{minted@isinline}%
   {\minted@pyoptvalueof@inline{#1}}%
   {\minted@pyoptvalueof@block{#1}}}
\def\minted@pyoptvalueof@inline#1{%
  \ifcsname minted@pyopt@cmd@#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@pyopt@cmd@#1\endcsname}%
  \else\ifcsname minted@pyopt@lexerinline@\minted@lexer @#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@pyopt@lexerinline@\minted@lexer @#1\endcsname}%
  \else\ifcsname minted@pyopt@globalinline@#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@pyopt@globalinline@#1\endcsname}%
  \else\ifcsname minted@pyopt@lexer@\minted@lexer @#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@pyopt@lexer@\minted@lexer @#1\endcsname}%
  \else
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@pyopt@global@#1\endcsname}%
  \fi\fi\fi\fi}
\def\minted@pyoptvalueof@block#1{%
  \ifcsname minted@pyopt@cmd@#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@pyopt@cmd@#1\endcsname}%
  \else\ifcsname minted@pyopt@lexer@\minted@lexer @#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@pyopt@lexer@\minted@lexer @#1\endcsname}%
  \else
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@pyopt@global@#1\endcsname}%
  \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`.

```
\def\minted@pgfkeyscreate@tex#1#2{%
  \minted@forcsvlist{\minted@pgfkeycreate@tex{#1}}{#2}}
\begingroup
\catcode`\==12
\gdef\minted@pgfkeycreate@tex#1#2{%
  \minted@pgfkeycreate@tex@i{#1}#2=\FV@Sentinel}
\gdef\minted@pgfkeycreate@tex@i#1#2=#3\FV@Sentinel{%
  \if\relax\detokenize{#3}\relax
    \expandafter\minted@pgfkeycreate@tex@ii
  \else
    \expandafter\minted@pgfkeycreate@tex@iii
  \fi
  {#1}{#2}#3\FV@Sentinel}
\gdef\minted@pgfkeycreate@tex@ii#1#2\FV@Sentinel{%
  \minted@pgfkeycreate@tex@iv{#1}{#2}{\pgfkeysnovalue}}
\gdef\minted@pgfkeycreate@tex@iii#1#2#3=\FV@Sentinel{%
  \minted@pgfkeycreate@tex@iv{#1}{#2}{#3}}
\endgroup
\def\minted@pgfkeycreate@tex@iv#1#2#3{%
  \def\minted@do##1{%
    \minted@iflexerscope{##1}%
     {\minted@do@i{##1}{@\minted@lexer}}%
     {\minted@do@i{##1}{}}}
  \def\minted@do@i##1##2{%
    \if\relax\detokenize{#1}\relax
      \pgfkeys{%
        /minted/##1/.cd,
        #2/.code={\expandafter\def\csname minted@texopt@##1##2@#2\endcsname{####1}},
        #2/.value required,
      }%
    \else
      \pgfkeys{%
        /minted/##1/.cd,
        #2/.code={%
          \def\minted@tmp{####1}%
          \ifx\minted@tmp\minted@const@pgfkeysnovalue
            \expandafter\let\csname minted@texopt@##1##2@#2\endcsname\minted@tmp
          \else\ifcsname minted@opthandler@immediate@\string#1\endcsname
            #1{minted@texopt@##1##2@#2}{####1}%
          \else
            \expandafter\def\csname minted@texopt@##1##2@#2\endcsname{#1{####1}}%
          \fi\fi},
        #2/.value required,
      }%
    \fi
  }%
  \minted@forcsvlist{\minted@do}{\minted@optscopes}%
  \pgfkeys{%
```

```
    /minted/global/.cd,
    #2={#3},
  }}
\def\mintedtexoptvalueof#1{%
  \ifbool{minted@isinline}%
   {\minted@texoptvalueof@inline{#1}}%
   {\minted@texoptvalueof@block{#1}}}
\def\minted@texoptvalueof@inline#1{%
  \ifcsname minted@texopt@cmd@#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@texopt@cmd@#1\endcsname}%
  \else\ifcsname minted@texopt@lexerinline@\minted@lexer @#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@texopt@lexerinline@\minted@lexer @#1\endcsname}%
  \else\ifcsname minted@texopt@globalinline@#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@texopt@globalinline@#1\endcsname}%
  \else\ifcsname minted@texopt@lexer@\minted@lexer @#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@texopt@lexer@\minted@lexer @#1\endcsname}%
  \else
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@texopt@global@#1\endcsname}%
  \fi\fi\fi\fi}
\def\minted@texoptvalueof@block#1{%
  \ifcsname minted@texopt@cmd@#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@texopt@cmd@#1\endcsname}%
  \else\ifcsname minted@texopt@lexer@\minted@lexer @#1\endcsname
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@texopt@lexer@\minted@lexer @#1\endcsname}%
  \else
    \unexpanded\expandafter\expandafter\expandafter{%
      \csname minted@texopt@global@#1\endcsname}%
  \fi\fi}
\def\minted@usetexoptsnonpygments{}
```

12.12.2 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.

```
\def\minted@opthandler@deforrestrictedescape#1#2{%
  \if\relax\detokenize{#2}\relax
```

```
    \expandafter\def\csname#1\endcsname{#2}%
  \else\if\relax\detokenize\expandafter{\@gobble#2}\relax
    \ifcat\relax\noexpand#2%
      \expandafter\expandafter\expandafter\minted@opthandler@deforrestrictedescape@i
        \expandafter\@gobble\string#2\FV@Sentinel{#1}{#2}%
    \else
      \FVExtraDetokenizeREscVArg{\expandafter\def\csname#1\endcsname}{#2}%
    \fi
  \else
    \FVExtraDetokenizeREscVArg{\expandafter\def\csname#1\endcsname}{#2}%
  \fi\fi}
\def\minted@opthandler@deforrestrictedescape@i#1#2\FV@Sentinel#3#4{%
  \ifcsname minted@isalpha\number`#1\endcsname
    \expandafter\def\csname#3\endcsname{#4}%
  \else
    \FVExtraDetokenizeREscVArg{\expandafter\def\csname#3\endcsname}{#4}%
  \fi}
\expandafter\let\csname
  minted@opthandler@immediate@\string\minted@opthandler@deforrestrictedescape
  \endcsname\relax
```

12.12.3 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.

```
\mintedpgfkeyscreate{fv}{
  baseline,
  baselinestretch,
  beameroverlays,
  backgroundcolor,
  backgroundcolorvphantom,
  bgcolor,
  bgcolorpadding,
  bgcolorvphantom,
  breakafter,
  breakafterinrun,
  breakaftersymbolpost,
  breakaftersymbolpre,
  breakanywhere,
  breakanywhereinlinestretch,
  breakanywheresymbolpost,
  breakanywheresymbolpre,
  breakautoindent,
  breakbefore,
  breakbeforeinrun,
  breakbeforesymbolpost,
  breakbeforesymbolpre,
  breakbytoken,
  breakbytokenanywhere,
  breakindent,
  breakindentnchars,
  breaklines,
  breaksymbol,
  breaksymbolindent,
  breaksymbolindentleft,
  breaksymbolindentleftnchars,
  breaksymbolindentnchars,
  breaksymbolindentright,
  breaksymbolindentrightnchars,
  breaksymbolleft,
  breaksymbolright,
  breaksymbolsep,
  breaksymbolsepleft,
  breaksymbolsepleftnchars,
  breaksymbolsepnchars,
  breaksymbolsepright,
  breaksymbolseprightnchars,
  curlyquotes,
  fillcolor,
  firstline,
  firstnumber,
  fontencoding,
  fontfamily,
  fontseries,
  fontshape,
  fontsize,
  formatcom,
  frame,
  framerule,
  framesep,
  highlightcolor,
  highlightlines,
  label,
  labelposition,
  lastline,
  linenos,
  listparameters,
  numberblanklines,
  numberfirstline,
  numbers,
  numbersep,
  obeytabs,
  reflabel,
  resetmargins,
  rulecolor,
  samepage,
  showspaces,
  showtabs,
  space,
  spacecolor,
  stepnumber,
  stepnumberfromfirst,
  stepnumberoffsetvalues,
  tab,
  tabcolor=black,
  tabsize,
  vspace,
  xleftmargin,
  xrightmargin,
}
```

minted (passed to Python)

- PHP should use startinline for \mintinline.

```
\mintedpgfkeyscreate{py}{
  autogobble<true>=false,
  encoding=utf8,
  extrakeywords=,
  extrakeywordsconstant=,
  extrakeywordsdeclaration=,
  extrakeywordsnamespace=,
  extrakeywordspseudo=,
  extrakeywordsreserved=,
  extrakeywordstype=,
  funcnamehighlighting<true>=true,
  gobble=0,
  gobblefilter=0,
  keywordcase=none,
  literalenvname=MintedVerbatim,
  mathescape<true>=false,
  python3<true>=true,
  rangeregexmatchnumber=1,
  rangeregexdotall<true>=false,
  rangeregexmultiline<true>=false,
  startinline<true>=false,
  stripall<true>=false,
  stripnl<true>=false,
  texcl<true>=false,
  texcomments<true>=false,
  tokenmerge<true>=true,
}
\mintedpgfkeyscreate[\minted@opthandler@deforrestrictedescape]{py}{
  codetagify=,
  escapeinside=,
  literatecomment=,
  rangestartstring=,
  rangestartstringline=,
  rangestartafterstring=,
  rangestartafterstringline=,
  rangestopstring=,
  rangestopstringline=,
  rangestopbeforestring=,
  rangestopbeforestringline=,
  rangeregex=,
}
\let\minted@tmplexer\minted@lexer
\def\minted@lexer{php}
\pgfkeys{
  /minted/lexerinline/.cd,
  startinline=true,
}
\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.

```
\mintedpgfkeyscreate{tex}{
  envname=Verbatim,
  ignorelexererrors=false,
  style=default,
}
\pgfkeys{
  /minted/globalinline/.cd,
  envname=VerbEnv,
}
\expandafter\def\expandafter\minted@usetexoptsnonpygments\expandafter{%
  \minted@usetexoptsnonpygments
  \edef\minted@literalenvname{\mintedpyoptvalueof{literalenvname}}%
  \edef\minted@envname{\mintedtexoptvalueof{envname}}%
  \expandafter\def\expandafter\minted@literalenv\expandafter{%
    \csname \minted@literalenvname\endcsname}%
  \expandafter\def\expandafter\minted@endliteralenv\expandafter{%
    \csname end\minted@literalenvname\endcsname}%
  \expandafter\expandafter\expandafter
    \let\expandafter\minted@literalenv\csname \minted@envname\endcsname
  \expandafter\expandafter\expandafter
    \let\expandafter\minted@endliteralenv\csname end\minted@envname\endcsname}%
\ifcsname minted@def@optcl\endcsname
  \ifx\minted@def@optcl\relax
    \let\minted@def@optcl\minted@undefined
  \fi
\fi
\providecommand{\minted@def@optcl}[4][]{%
  \minted@warning{Macro \string\minted@def@optcl\space is deprecated with minted v3
    and no longer has any effect}}
```

12.13 Caching, styles, and highlighting

12.13.1 State

minted@didcreatefiles

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

```
\newbool{minted@didcreatefiles}
```

12.13.2 Cache management

\minted@addcachefilename
\minted@numcachefiles

\minted@cachefile<n>
\minted@cachechecksum
\mintedoldcachechecksum

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

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

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

```
\def\minted@numcachefiles{0}
\def\minted@addcachefilename#1{%
  \ifbool{minted@canexec}%
   {\xdef\minted@numcachefiles{\the\numexpr\minted@numcachefiles+1\relax}%
    \expandafter\xdef\csname minted@cachefile\minted@numcachefiles\endcsname{#1}%
    \edef\minted@tmp{\minted@mdfivehash{#1}}%
    \expandafter\minted@cachechecksum@update\expandafter{\minted@tmp}}%
   {}}
\let\minted@cachechecksum\relax
\ifcsname mintedoldcachechecksum\endcsname
\else
  \let\mintedoldcachechecksum\relax
\fi
\edef\minted@cachechecksum@files{\minted@mdfivehash{}}
\expandafter\let\expandafter\minted@intfromhex\csname int_from_hex:n\endcsname
\expandafter\let\expandafter\minted@inttoHex\csname int_to_Hex:n\endcsname
\expandafter\let\expandafter\minted@intmod\csname int_mod:nn\endcsname
\def\minted@intmodsixteen#1{\minted@intmod{#1}{16}}
\def\minted@cachechecksum@update#1{%
  \xdef\minted@cachechecksum@files{%
    \expandafter\minted@cachechecksum@files@calc
      \minted@cachechecksum@files\FV@Sentinel#1\FV@Sentinel}%
  \xdef\minted@cachechecksum{%
    \detokenize\expandafter{\minted@cachechecksum@files}%
    \detokenize{:}%
    \detokenize\expandafter{\minted@numcachefiles}}}
\def\minted@cachechecksum@files@calc#1#2\FV@Sentinel#3#4\FV@Sentinel{%
  \minted@inttoHex{%
    \the\numexpr
      \minted@intmodsixteen{\minted@intfromhex{#1}+\minted@intfromhex{#3}}%
    \relax}%
  \if\relax\detokenize{#2}\relax
    \expandafter\@gobble
  \else
    \expandafter\@firstofone
  \fi
   {\minted@cachechecksum@files@calc#2\FV@Sentinel#4\FV@Sentinel}}
```

\minted@clean

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

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

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

```
\def\minted@clean{%
  \ifbool{minted@canexec}%
   {\ifbool{minted@didcreatefiles}%
     {\minted@clean@i}%
     {\ifbool{minted@fasthighlightmode}%
       {\minted@clean@i}%
       {\ifbool{minted@cache}%
         {\ifx\minted@cachechecksum\mintedoldcachechecksum
          \else
            \expandafter\minted@clean@i
          \fi}%
         {}}}%
    \ifbool{minted@fasthighlightmode}{}{\global\boolfalse{minted@canexec}}}%
   {}}
\def\minted@clean@i{%
  \ifnum\minted@numcachefiles>0\relax
    \expandafter\minted@savecachelist
  \fi
  \ifbool{minted@fasthighlightmode}%
   {}%
   {\ifnum\minted@numcachefiles>0\relax
      \expandafter\minted@exec@clean
    \else
      \expandafter\minted@exec@cleantemp
    \fi}%
  \gdef\minted@numcachefiles{0}}
\def\minted@savecachelist{%
  \pydatasetfilename{\MintedDataFilename}%
  \minted@fasthighlightmode@checkstart
  \pydatawritedictopen
  \pydatawritekeyvalue{command}{clean}%
  \pydatawritekeyedefvalue{jobname}{\jobname}%
  \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
  \pydatawritekeyedefvalue{cachepath}{\minted@cachepath}%
  \pydatawritekey{cachefiles}%
  \pydatawritemlvaluestart
  \pydatawritemlvalueline{[}%
  \gdef\minted@tempindex{1}%
  \loop\unless\ifnum\minted@tempindex>\minted@numcachefiles\relax
    \expandafter\minted@savecachelist@writecachefile\expandafter{%
      \csname minted@cachefile\minted@tempindex\endcsname}%
    \expandafter\global\expandafter
      \let\csname minted@cachefile\minted@tempindex\endcsname\minted@undefined
    \xdef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}%
  \repeat
  \pydatawritemlvalueline{]}%
  \pydatawritemlvalueend
  \pydatawritedictclose
  \ifbool{minted@fasthighlightmode}{}{\pydataclosefilename{\MintedDataFilename}}}
\begingroup
\catcode`\"=12
\catcode`\,=12
```

```
\gdef\minted@savecachelist@writecachefile#1{%
  \expandafter\pydatawritemlvalueline\expandafter{\expandafter"#1",}}
\endgroup
\AddToHook{enddocument/afterlastpage}{%
  \minted@clean
  \minted@fasthighlightmode@checkend
  \ifbool{minted@cache}%
   {\immediate\write\@auxout{%
      \xdef\string\mintedoldcachechecksum{\string\detokenize{\minted@cachechecksum}}}}%
   {}}
```

12.13.3 Style definitions

\minted@patch@PygmentsStyledef

The macros generated by Pygments must be patched.

Redefine the single quote macro for upquote compatibility.

```
\def\minted@patch@PygmentsZsq{%
  \ifcsname\minted@styleprefix Zsq\endcsname
    \ifcsstring{\minted@styleprefix Zsq}{\char`\'}{\minted@patch@PygmentsZsq@i}{}%
  \fi}
\begingroup
\catcode`\'=\active
\gdef\minted@patch@PygmentsZsq@i{\def\PYGZsq{'}}
\endgroup
```

Redefine the hyphen to prevent unintended line breaks under LuaTeX.

```
\def\minted@patch@PygmentsZhy{%
  \ifcsname\minted@styleprefix Zhy\endcsname
    \ifcsstring{\minted@styleprefix Zhy}{\char`\-}{\def\PYGZhy{\mbox{-}}}{}%
  \fi}
```

Redefine the error token if ignorelexererrors.

```
\def\minted@patch@ignorelexererrors{%
  \edef\minted@tmp{\mintedtexoptvalueof{ignorelexererrors}}%
  \ifdefstring{\minted@tmp}{true}%
   {\expandafter\let\csname\minted@styleprefix @tok@err\endcsname\relax}%
   {}}
```

Patch the Pygments token macros.

```
\def\minted@patch@PygmentsPYG{%
  \edef\minted@tmp{\mintedpyoptvalueof{texcomments}}%
  \ifdefstring{\minted@tmp}{true}{\fvset{texcomments}}{}%
  \expandafter\minted@patch@PygmentsPYG@i\expandafter{%
    \csname\minted@styleprefix\endcsname}}
\def\minted@patch@PygmentsPYG@i#1{%
  \VerbatimPygments{#1}{#1}}%
```

Apply all patches.

```
\def\minted@patch@PygmentsStyledef{%
  \minted@patch@PygmentsZsq
  \minted@patch@PygmentsZhy
  \minted@patch@ignorelexererrors
  \minted@patch@PygmentsPYG}
```

\minted@standardcatcodes

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

```
\def\minted@standardcatcodes{%
  \catcode`\\=0
  \catcode`\{=1
  \catcode`\}=2
  \catcode`\#=6
  \catcode`\ =10
  \catcode`\@=11
  \catcode`\`=12
  \catcode`\==12
  \catcode`\+=12
  \catcode`\.=12
  \catcode`\,=12
  \catcode`\[=12
  \catcode`\]=12
  \catcode`\%=14}
```

`\minted@defstyle`

Define highlighting style macros.

```
\def\minted@defstyle{%
  \edef\minted@tmp{\mintedtexoptvalueof{style}}%
  \expandafter\minted@defstyle@i\expandafter{\minted@tmp}}
\def\minted@defstyle@i#1{%
  \minted@ifalphanumhyphenunderscore{#1}%
   {\minted@defstyle@ii{#1}}%
   {\minted@error{Highlighting style is set to "#1" but only style names with
      alphanumeric characters, hyphens, and underscores are supported;
      falling back to default style}%
    \minted@defstyle@ii{default}}}
\def\minted@defstyle@ii#1{%
  \ifcsname minted@styledef@#1\endcsname
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
  {\csname minted@styledef@#1\endcsname
   \minted@patch@PygmentsStyledef}%
  {\minted@defstyle@load{#1}}}
```

`\minted@defstyle@load`

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

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

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

The \begingroup...\endgroup around \minted@exec@styledef and associated messages is necessary to prevent errors related to the message file. If a style does not exist, then the Python executable will create a _<hash>.message.minted file, which is brought in via \InputIfFileExists and generates an error message. After this, there is an attempt to load the default style. If the default style needs to be generated, then \InputIfFileExists will attempt to bring in a _<hash>.message.minted file regardless of whether it exists, unless it is wrapped in the \begingroup...\endgroup.

```
\def\minted@catchfiledef#1#2{%
  \CatchFileDef{#1}{#2}{\minted@standardcatcodes\endlinechar=-1}}
\def\minted@defstyle@load#1{%
  \minted@detectconfig
  \ifbool{minted@cache}%
   {\edef\minted@styledeffilename{#1\detokenize{.style.minted}}%
    \edef\minted@styledeffilepath{\minted@cachepath\minted@styledeffilename}%
    \IfFileExists{\minted@styledeffilepath}%
     {\minted@defstyle@input{#1}}%
     {\ifbool{minted@canexec}%
      {\minted@defstyle@generate{#1}}%
      {\minted@error{Missing definition for highlighting style "#1" (minted executable
         is unavailable or disabled); attempting to substitute fallback style}%
       \minted@defstyle@fallback{#1}}}}%
   {\edef\minted@styledeffilename{%
      \detokenize{_}\MintedJobnameMdfive\detokenize{.style.minted}}%
    \let\minted@styledeffilepath\minted@styledeffilename
    \ifbool{minted@canexec}%
     {\minted@defstyle@generate{#1}}%
     {\minted@error{Missing definition for highlighting style "#1" (minted executable
        is unavailable or disabled); attempting to substitute fallback style}%
      \minted@defstyle@fallback{#1}}}}
\def\minted@defstyle@input#1{%
  \begingroup
  \minted@catchfiledef{\minted@tmp}{\minted@styledeffilepath}%
  \minted@tmp
  \ifcsname\minted@styleprefix\endcsname
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
  {\expandafter\global\expandafter\let\csname minted@styledef@#1\endcsname\minted@tmp
   \endgroup
   \ifbool{minted@cache}{\minted@addcachefilename{\minted@styledeffilename}}{}%
   \csname minted@styledef@#1\endcsname
   \minted@patch@PygmentsStyledef}%
  {\endgroup
   \ifbool{minted@canexec}%
    {\minted@warning{Invalid or corrupted style definition file
       "\minted@styledeffilename"; attempting to regenerate}%
     \minted@defstyle@generate{#1}}%
    {\minted@error{Invalid or corrupted style definition file
       "\minted@styledeffilename"; attempting to substitute fallback style
       (minted executable is unavailable or disabled)}%
     \minted@defstyle@fallback{#1}}}}
\def\minted@defstyle@generate#1{%
```

```
  \pydatasetfilename{\MintedDataFilename}%
  \minted@fasthighlightmode@checkstart
  \pydatawritedictopen
  \pydatawritekeyvalue{command}{styledef}%
  \pydatawritekeyedefvalue{jobname}{\jobname}%
  \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
  \pydatawritekeyedefvalue{currentfilepath}{\CurrentFilePath}%
  \pydatawritekeyedefvalue{currentfile}{\CurrentFile}%
  \pydatawritekeyedefvalue{inputlineno}{\minted@inputlineno}%
  \pydatawritekeyedefvalue{cachepath}{\minted@cachepath}%
  \pydatawritekeyedefvalue{styledeffilename}{\minted@styledeffilename}%
  \pydatawritekeyvalue{style}{#1}%
  \pydatawritekeyedefvalue{commandprefix}{\minted@styleprefix}%
  \pydatawritedictclose
  \ifbool{minted@fasthighlightmode}%
   {\minted@defstyle@fallback{#1}}%
   {\pydataclosefilename{\MintedDataFilename}%
    \begingroup
    \minted@exec@styledef
    \global\booltrue{minted@didcreatefiles}%
    \ifx\minted@exec@warning\relax
    \else
      \expandafter\minted@exec@warning
    \fi
    \ifx\minted@exec@error\relax
      \expandafter\minted@defstyle@generate@i
    \else
      \expandafter\minted@defstyle@generate@error
    \fi
    {#1}}}
\def\minted@defstyle@generate@i#1{%
  \endgroup
  \begingroup
  \minted@catchfiledef{\minted@tmp}{\minted@styledeffilepath}%
  \minted@tmp
  \ifcsname\minted@styleprefix\endcsname
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
  {\expandafter\global\expandafter\let\csname minted@styledef@#1\endcsname\minted@tmp
   \endgroup
   \ifbool{minted@cache}{\minted@addcachefilename{\minted@styledeffilename}}{}%
   \csname minted@styledef@#1\endcsname
   \minted@patch@PygmentsStyledef}%
  {\endgroup
   \minted@error{Failed to create style definition file "\minted@styledeffilename"
     (no error message, see "\MintedErrlogFilename" if it exists);
     attempting to substitute fallback style}%
   \minted@defstyle@fallback{#1}}}
\def\minted@defstyle@generate@error#1{%
  \minted@exec@error
  \endgroup
  \minted@defstyle@fallback{#1}}
```

```
\def\minted@defstyle@fallback#1{%
  \ifstrequal{#1}{default}%
   {\expandafter\global\expandafter
      \let\csname minted@styledef@default\endcsname\minted@styledeffallback}%
   {\ifcsname minted@styledef@default\endcsname
    \else
      \minted@defstyle@load{default}%
    \fi
    \expandafter\let\expandafter\minted@tmp\csname minted@styledef@default\endcsname
    \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.

```
\def\minted@styledeffallback{%
  \expandafter\def\csname\minted@styleprefix\endcsname##1##2{##2}%
  \expandafter\def\csname\minted@styleprefix Zbs\endcsname{\char`\\}%
  \expandafter\def\csname\minted@styleprefix Zus\endcsname{\char`\_}%
  \expandafter\def\csname\minted@styleprefix Zob\endcsname{\char`\{}%
  \expandafter\def\csname\minted@styleprefix Zcb\endcsname{\char`\}}%
  \expandafter\def\csname\minted@styleprefix Zca\endcsname{\char`\^}%
  \expandafter\def\csname\minted@styleprefix Zam\endcsname{\char`\&}%
  \expandafter\def\csname\minted@styleprefix Zlt\endcsname{\char`\<}%
  \expandafter\def\csname\minted@styleprefix Zgt\endcsname{\char`\>}%
  \expandafter\def\csname\minted@styleprefix Zsh\endcsname{\char`\#}%
  \expandafter\def\csname\minted@styleprefix Zpc\endcsname{\char`\%}%
  \expandafter\def\csname\minted@styleprefix Zdl\endcsname{\char`\$}%
  \expandafter\def\csname\minted@styleprefix Zhy\endcsname{\char`\-}%
  \expandafter\def\csname\minted@styleprefix Zsq\endcsname{\char`\'}%
  \expandafter\def\csname\minted@styleprefix Zdq\endcsname{\char`\"}%
  \expandafter\def\csname\minted@styleprefix Zti\endcsname{\char`\~}%
  \minted@patch@PygmentsStyledef}
```

12.13.4 Lexer-specific line numbering

minted@FancyVerbLineTemp

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

```
\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.

```
\def\minted@lexerlinenoson{%
  \ifcsname c@minted@lexer\minted@lexer\endcsname
  \else
    \newcounter{minted@lexer\minted@lexer}%
  \fi
  \setcounter{minted@FancyVerbLineTemp}{\value{FancyVerbLine}}%
  \setcounter{FancyVerbLine}{\value{minted@lexer\minted@lexer}}}
\def\minted@lexerlinenosoff{%
  \setcounter{minted@lexer\minted@lexer}{\value{FancyVerbLine}}%
  \setcounter{FancyVerbLine}{\value{minted@FancyVerbLineTemp}}}
\ifbool{minted@inputlexerlinenos}%
 {\let\minted@inputlexerlinenoson\minted@lexerlinenoson
  \let\minted@inputlexerlinenosoff\minted@lexerlinenosoff}%
 {\let\minted@inputlexerlinenoson\relax
  \let\minted@inputlexerlinenosoff\relax
  \ifbool{minted@lexerlinenos}
   {}%
   {\let\minted@lexerlinenoson\relax
    \let\minted@lexerlinenosoff\relax}}
```

\minted@codewrapper

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

```
\def\minted@codewrapper#1{%
  \ifcsname minted@inputfilepath\endcsname
    \minted@inputlexerlinenoson
  \else
    \minted@lexerlinenoson
  \fi
  #1%
  \ifcsname minted@inputfilepath\endcsname
    \minted@inputlexerlinenosoff
  \else
    \minted@lexerlinenosoff
  \fi}
```

12.13.5 Highlighting code

\minted@highlight
\minted@highlightinputfile

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

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

The default \minted@highlight@fallback inserts a placeholder. Typically commands/environments will redefine the fallback locally to inserted 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.

cacheignoresfilecontents requires modifications. There is no longer an error if an external file cannot be located for file contents verification. The settings buffer no longer includes the MD5 hash of file contents, so the buffer hash (which is used to name cache files) no longer depends on file contents. When a cache file does not exist but can be created, the MD5 hash of file contents is still passed to the Python side and is still verified like normal.

```
\def\minted@debug@input{%
  \ifbool{minted@debug}%
   {\immediate\typeout{%
     minted debug: \string\input\space at
     \ifx\CurrentFile\@empty\else\CurrentFile\space\fi line \the\inputlineno}}%
   {}}
\def\minted@highlight{%
  \ifbool{minted@iscmd}%
   {\edef\minted@inputlineno{\the\inputlineno}}%
   {\edef\minted@inputlineno{%
      \the\numexpr\the\inputlineno-\minted@tmpcodebufferlength-1\relax}}%
  \minted@defstyle
  \pydatasetbuffername{minted@tmpdatabuffer}%
  \pydatabufferkeyvalue{command}{highlight}%
  \pydatabufferkey{code}%
  \pydatabuffermlvaluestart
  \gdef\minted@tempindex{1}%
  \loop\unless\ifnum\minted@tempindex>\minted@tmpcodebufferlength\relax
    \expandafter\let\expandafter
      \minted@tmp\csname minted@tmpcodebufferline\minted@tempindex\endcsname
    \expandafter\pydatabuffermlvalueline\expandafter{\minted@tmp}%
    \xdef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}%
  \repeat
  \pydatabuffermlvalueend
  \minted@highlight@i}
\def\minted@highlightinputfile{%
  \edef\minted@inputlineno{\the\inputlineno}%
  \minted@defstyle
  \edef\minted@inputfilemdfivesum{\minted@filemdfivehash{\minted@inputfilepath}}%
  \ifx\minted@inputfilemdfivesum\@empty
    \ifminted@cacheignoresfilecontents
      \expandafter\expandafter\expandafter\@secondoftwo
    \else
      \expandafter\expandafter\expandafter\@firstoftwo
    \fi
  \else
    \expandafter\@secondoftwo
  \fi
  {\minted@error{Cannot find input file "\minted@inputfilepath"; inserting placeholder}%
   \minted@insertplaceholder}%
  {\pydatasetbuffername{minted@tmpdatabuffer}%
   \pydatabufferkeyvalue{command}{highlight}%
   \pydatabufferkeyedefvalue{inputfilepath}{\minted@inputfilepath}%
   \ifbool{minted@cacheignoresfilecontents}%
    {}%
    {\pydatabufferkeyedefvalue{inputfilemdfivesum}{\minted@inputfilemdfivesum}}%
   \minted@highlight@i}}
\def\minted@def@FV@GetKeyValues@standardcatcodes{%
  \let\minted@FV@GetKeyValues@orig\FV@GetKeyValues
  \def\FV@GetKeyValues##1{%
    \begingroup
    \minted@standardcatcodes
    \minted@FV@GetKeyValues@i{##1}}%
  \def\minted@FV@GetKeyValues@i##1[##2]{%
    \endgroup
    \let\FV@GetKeyValues\minted@FV@GetKeyValues@orig
    \let\minted@FV@GetKeyValues@i\minted@undefined
    \FV@GetKeyValues{##1}[##2]}}
\def\minted@highlight@i{%
  \pydatabufferkeyedefvalue{pyopt.lexer}{\minted@lexer}%
  \pydatabufferkeyedefvalue{pyopt.commandprefix}{\minted@styleprefix}%
  \minted@forcsvlist{\minted@highlight@bufferpykeys}{\minted@optkeyslist@py}%
  \ifbool{minted@cache}%
   {\edef\minted@highlightfilename{\pydatabuffermdfivesum\detokenize{.highlight.minted}}%
    \edef\minted@highlightfilepath{\minted@cachepath\minted@highlightfilename}%
    \IfFileExists{\minted@highlightfilepath}%
     {\minted@codewrapper{%
        \minted@def@FV@GetKeyValues@standardcatcodes
        \minted@debug@input
        \input{\minted@highlightfilepath}}%
      \minted@addcachefilename{\minted@highlightfilename}}%
     {\ifbool{minted@canexec}%
       {\ifbool{minted@fasthighlightmode}%
         {\ifcsname minted@processedfilename@\minted@highlightfilename\endcsname
            \expandafter\@firstoftwo
          \else
            \expandafter\@secondoftwo
          \fi
           {\minted@insertplaceholder}%
           {\expandafter\global\expandafter\let
              \csname minted@processedfilename@\minted@highlightfilename\endcsname\relax
            \minted@iffasthighlightmode@buffertempfile
            \minted@highlight@create}}%
         {\minted@iffasthighlightmode@buffertempfile
          \minted@highlight@create}}%
       {\ifbool{minted@frozencache}%
         {\minted@error{Cannot highlight code (frozencache=true);
            attempting to typeset without highlighting}}%
         {\minted@error{Cannot highlight code (minted executable is unavailable or
            disabled); attempting to typeset without highlighting}}%
        \minted@highlight@fallback}}}%
   {\edef\minted@highlightfilename{%
      \detokenize{_}\MintedJobnameMdfive\detokenize{.highlight.minted}}%
    \let\minted@highlightfilepath\minted@highlightfilename
    \ifbool{minted@canexec}%
     {\minted@highlight@create}%
     {\minted@error{Cannot highlight code (minted executable is unavailable or
        disabled); attempting to typeset without highlighting}%
      \minted@highlight@fallback}}%
  \pydataclearbuffername{minted@tmpdatabuffer}}
\def\minted@highlight@bufferpykeys#1{%
  \edef\minted@tmp{\mintedpyoptvalueof{#1}}%
  \ifx\minted@tmp\minted@const@pgfkeysnovalue
  \else
    \pydatabufferkeyedefvalue{pyopt.#1}{\minted@tmp}%
  \fi}
```

```
\def\minted@highlight@create{%
  \pydatasetfilename{\MintedDataFilename}%
  \minted@fasthighlightmode@checkstart
  \pydatawritedictopen
  \pydatawritebuffer
  \pydatawritekeyedefvalue{jobname}{\jobname}%
  \pydatawritekeyedefvalue{timestamp}{\minted@timestamp}%
  \pydatawritekeyedefvalue{currentfilepath}{\CurrentFilePath}%
  \pydatawritekeyedefvalue{currentfile}{\CurrentFile}%
  \pydatawritekeyedefvalue{inputlineno}{\minted@inputlineno}%
  \pydatawritekeyedefvalue{cachepath}{\minted@cachepath}%
  \pydatawritekeyedefvalue{highlightfilename}{\minted@highlightfilename}%
  \ifbool{minted@cacheignoresfilecontents}%
   {\pydatawritekeyedefvalue{inputfilemdfivesum}{\minted@inputfilemdfivesum}}%
   {}%
  \pydatawritedictclose
  \ifbool{minted@fasthighlightmode}%
   {\minted@insertplaceholder}%
   {\pydataclosefilename{\MintedDataFilename}%
    \begingroup
    \minted@exec@highlight
    \global\booltrue{minted@didcreatefiles}%
    \IfFileExists{\minted@highlightfilepath}%
     {\ifx\minted@exec@warning\relax
      \else
        \expandafter\minted@exec@warning
      \fi
      \ifx\minted@exec@error\relax
      \else
        \expandafter\minted@exec@error
      \fi
      \endgroup
      \minted@codewrapper{%
        \minted@def@FV@GetKeyValues@standardcatcodes
        \minted@debug@input
        \input{\minted@highlightfilepath}}%
      \ifbool{minted@cache}{\minted@addcachefilename{\minted@highlightfilename}}{}}%
     {\ifx\minted@exec@warning\relax
      \else
        \expandafter\minted@exec@warning
      \fi
      \ifx\minted@exec@error\relax
        \minted@error{Minted executable failed during syntax highlighting
          but returned no error message (see if "\MintedErrlogFilename" exists)}%
      \else
        \expandafter\minted@exec@error
      \fi
      \endgroup
      \minted@highlight@fallback}}}
\def\minted@highlight@fallback{%
  \minted@insertplaceholder}
```

\minted@iffasthighlightmode@buffertempfile
\MintedRegisterTempFileExtension

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.

```
\def\MintedRegisterTempFileExtension#1{%
  \ifstrempty{#1}%
   {}%
   {\expandafter
      \let\csname minted@buffertempfileextension@\detokenize{#1}\endcsname\relax}}
\minted@forcsvlist{\MintedRegisterTempFileExtension}{
  .listing,
  .out,
  .outfile,
  .output,
  .tcbtemp,
  .temp,
  .tempfile,
  .tmp,
  .verb,
  .vrb,
}
\def\minted@iffasthighlightmode@buffertempfile{%
  \ifbool{minted@fasthighlightmode}%
   {\ifcsname minted@inputfilepath\endcsname
      \expandafter\minted@iffasthighlightmode@buffertempfile@i
    \fi}%
   {}}
\def\minted@iffasthighlightmode@buffertempfile@i{%
  \csname file_parse_full_name:nNNN\endcsname{\minted@inputfilepath}%
    \minted@inputfileparent\minted@inputfilestem\minted@inputfilesuffix
  \ifcsname minted@buffertempfileextension@\minted@inputfilesuffix\endcsname
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
   {\minted@iffasthighlightmode@buffertempfile@ii}%
   {\csname file_parse_full_name:nNNN\endcsname{\minted@inputfilestem}%
      \minted@inputfileparent\minted@inputfilestem\minted@inputfilesuffix
    \ifcsname minted@buffertempfileextension@\minted@inputfilesuffix\endcsname
      \expandafter\minted@iffasthighlightmode@buffertempfile@ii
    \fi}}
\def\minted@iffasthighlightmode@buffertempfile@ii{%
  \begingroup
  \gdef\minted@tmpcodebufferlength{0}%
  \openin\minted@intempfile=\minted@inputfilepath
  \endlinechar=-1%
  \let\do\@makeother\FVExtraDoSpecials
  \catcode`\^^I=12%
  \loop\unless\ifeof\minted@intempfile
    \read\minted@intempfile to\minted@intempfileline
    \xdef\minted@tmpcodebufferlength{\the\numexpr\minted@tmpcodebufferlength+1\relax}%
    \expandafter\global\expandafter\let\csname
      minted@tmpcodebufferline\minted@tmpcodebufferlength
      \endcsname\minted@intempfileline
  \repeat
  \closein\minted@intempfile
  \expandafter\ifx\csname
      minted@tmpcodebufferline\minted@tmpcodebufferlength\endcsname\@empty
    \expandafter\global\expandafter\let\csname
      minted@tmpcodebufferline\minted@tmpcodebufferlength\endcsname\minted@undefined
    \xdef\minted@tmpcodebufferlength{\the\numexpr\minted@tmpcodebufferlength-1\relax}%
  \fi
  \endgroup
  \edef\minted@inputfilemdfivesum@check{\minted@filemdfivehash{\minted@inputfilepath}}%
  \ifx\minted@inputfilemdfivesum@check\minted@inputfilemdfivesum
    \expandafter\@gobble
  \else
    \expandafter\@firstofone
  \fi
   {\VerbatimClearBuffer[buffername=minted@tmpcodebuffer]}%
  \ifnum\minted@tmpcodebufferlength>0\relax
    \expandafter\@firstofone
  \else
    \expandafter\@gobble
  \fi
   {\minted@iffasthighlightmode@buffertempfile@iii}}
\def\minted@iffasthighlightmode@buffertempfile@iii{%
  \pydatabufferkey{code}%
  \pydatabuffermlvaluestart
  \gdef\minted@tempindex{1}%
  \loop\unless\ifnum\minted@tempindex>\minted@tmpcodebufferlength\relax
    \expandafter\let\expandafter
      \minted@tmp\csname minted@tmpcodebufferline\minted@tempindex\endcsname
    \expandafter\pydatabuffermlvalueline\expandafter{\minted@tmp}%
    \xdef\minted@tempindex{\the\numexpr\minted@tempindex+1\relax}%
  \repeat
  \pydatabuffermlvalueend
  \VerbatimClearBuffer[buffername=minted@tmpcodebuffer]}
```

12.14 Public API

\setminted

Set global or lexer-level options.

```
\newcommand{\setminted}[2][]{%
  \ifstrempty{#1}%
   {\pgfkeys{/minted/global/.cd,#2}}%
   {\let\minted@tmplexer\minted@lexer
    \edef\minted@lexer{#1}%
    \pgfkeys{/minted/lexer/.cd,#2}%
    \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.

```
\newcommand{\setmintedinline}[2][]{%
  \ifstrempty{#1}%
   {\pgfkeys{/minted/globalinline/.cd,#2}}%
   {\let\minted@tmplexer\minted@lexer
    \edef\minted@lexer{#1}%
    \pgfkeys{/minted/lexerinline/.cd,#2}%
    \let\minted@lexer\minted@tmplexer}}
```

\usemintedstyle

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

```
\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.

```
\def\mintinline{%
  \FVExtraRobustCommand\RobustMintInline\FVExtraUnexpandedReadStarOArgMArgBVArg}
\FVExtrapdfstringdefDisableCommands{%
  \def\RobustMintInline{}}
\newrobustcmd{\RobustMintInline}[2][]{%
  \ifbool{FVExtraRobustCommandExpanded}%
   {\@ifnextchar\bgroup
     {\FVExtraReadVArg{\RobustMintInlineProcess{#1}{#2}}}%
     {\minted@error{Inline delimiters must be paired curly braces in this context}}}%
   {\FVExtraReadVArg{\RobustMintInlineProcess{#1}{#2}}}}
\def\RobustMintInlineProcess@highlight#1#2#3{%
  \begingroup
  \booltrue{minted@iscmd}%
  \booltrue{minted@isinline}%
  \ifstrempty{#1}{}{\pgfkeys{/minted/cmd/.cd,#1}}%
  \edef\minted@lexer{#2}%
  \minted@usefvopts
  \minted@usetexoptsnonpygments
  \FVExtraDetokenizeVArg{%
     \FVExtraRetokenizeVArg{\RobustMintInlineProcess@highlight@i}{\FV@CatCodes}}{#3}}
\def\RobustMintInlineProcess@highlight@i#1{%
  \expandafter\def\csname minted@tmpcodebufferline1\endcsname{#1}%
  \gdef\minted@tmpcodebufferlength{1}%
  \let\minted@highlight@fallback\RobustMintInlineProcess@highlight@fallback
  \minted@highlight
  \gdef\minted@tmpcodebufferlength{0}%
  \endgroup}
```

```
\def\RobustMintInlineProcess@highlight@fallback{%
  \minted@useadditionalfvoptsnopy
  \fvset{extra=true}%
  \minted@codewrapper{%
    \expandafter\let\expandafter\minted@tmp\csname minted@tmpcodebufferline1\endcsname
    \expandafter\Verb\expandafter{\minted@tmp}}}
\def\RobustMintInlineProcess@placeholder#1#2#3{%
  \begingroup
  \booltrue{minted@iscmd}%
  \booltrue{minted@isinline}%
  \minted@insertplaceholder
  \endgroup}
\def\RobustMintInlineProcess@verbatim#1#2#3{%
  \begingroup
  \booltrue{minted@iscmd}%
  \booltrue{minted@isinline}%
  \ifstrempty{#1}{}{\pgfkeys{/minted/cmd/.cd,#1}}%
  \edef\minted@lexer{#2}%
  \minted@usefvopts
  \minted@useadditionalfvoptsnopy
  \minted@usetexoptsnonpygments
  \fvset{extra=true}%
  \minted@codewrapper{\Verb{#3}}%
  \endgroup}
\ifbool{minted@placeholder}%
 {\let\RobustMintInlineProcess\RobustMintInlineProcess@placeholder}%
 {\ifbool{minted@verbatim}%
   {\let\RobustMintInlineProcess\RobustMintInlineProcess@verbatim}%
   {\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).

```
\def\mint{%
  \FVExtraRobustCommand\RobustMint\FVExtraUnexpandedReadStarOArgMArgBVArg}
\FVExtrapdfstringdefDisableCommands{%
  \def\RobustMint{}}
\newrobustcmd{\RobustMint}[2][]{%
  \ifbool{FVExtraRobustCommandExpanded}%
   {\@ifnextchar\bgroup
     {\FVExtraReadVArg{\RobustMintProcess{#1}{#2}}}%
     {\minted@error{Delimiters must be paired curly braces in this context}}}%
   {\FVExtraReadVArg{\RobustMintProcess{#1}{#2}}}}
\def\RobustMintProcess@highlight#1#2#3{%
  \begingroup
  \ifstrempty{#1}{}{\pgfkeys{/minted/cmd/.cd,#1}}%
  \edef\minted@lexer{#2}%
  \minted@usefvopts
  \minted@usetexoptsnonpygments
  \FVExtraDetokenizeVArg{%
```

```
    \FVExtraRetokenizeVArg{\RobustMintProcess@highlight@i}{\FV@CatCodes}}{#3}}
\def\RobustMintProcess@highlight@i#1{%
  \expandafter\def\csname minted@tmpcodebufferline1\endcsname{#1}%
  \gdef\minted@tmpcodebufferlength{1}%
  \let\minted@highlight@fallback\RobustMintProcess@highlight@fallback
  \minted@highlight
  \gdef\minted@tmpcodebufferlength{0}%
  \endgroup}
\def\RobustMintProcess@highlight@fallback{%
  \minted@useadditionalfvoptsnopy
  \minted@codewrapper{%
    \VerbatimInsertBuffer[buffername=minted@tmpcodebuffer,insertenvname=\minted@envname]}}
\def\RobustMintProcess@placeholder#1#2#3{%
  \minted@insertplaceholder}
\def\RobustMintProcess@verbatim#1#2#3{%
  \begingroup
  \ifstrempty{#1}{}{\pgfkeys{/minted/cmd/.cd,#1}}%
  \edef\minted@lexer{#2}%
  \minted@usefvopts
  \minted@useadditionalfvoptsnopy
  \minted@usetexoptsnonpygments
  \FVExtraDetokenizeVArg{%
    \FVExtraRetokenizeVArg{\RobustMintProcess@verbatim@i}{\FV@CatCodes}}{#3}}
\def\RobustMintProcess@verbatim@i#1{%
  \expandafter\def\csname minted@tmpcodebufferline1\endcsname{#1}%
  \gdef\minted@tmpcodebufferlength{1}%
  \minted@codewrapper{%
    \VerbatimInsertBuffer[buffername=minted@tmpcodebuffer,insertenvname=\minted@envname]}%
  \gdef\minted@tmpcodebufferlength{0}%
  \endgroup}
\ifbool{minted@placeholder}%
 {\let\RobustMintProcess\RobustMintProcess@placeholder}%
 {\ifbool{minted@verbatim}%
   {\let\RobustMintProcess\RobustMintProcess@verbatim}%
   {\let\RobustMintProcess\RobustMintProcess@highlight}}
```

`minted` *(env.)*

Highlight a longer piece of code inside a verbatim environment.

```
\newenvironment{minted}[2][]%
 {\VerbatimEnvironment
  \MintedBegin{#1}{#2}}%
 {\MintedEnd}
\def\MintedBegin@highlight#1#2{%
  \ifstrempty{#1}{}{\pgfkeys{/minted/cmd/.cd,#1}}%
  \edef\minted@lexer{#2}%
  \minted@usefvopts
  \minted@usetexoptsnonpygments
  \begin{VerbatimBuffer}[buffername=minted@tmpcodebuffer,globalbuffer=true]}
\def\MintedEnd@highlight{%
  \end{VerbatimBuffer}%
  \let\minted@highlight@fallback\MintedEnv@highlight@fallback
  \minted@highlight
  \VerbatimClearBuffer[buffername=minted@tmpcodebuffer]}
\def\MintedEnv@highlight@fallback{%
```

```
  \minted@useadditionalfvoptsnopy
  \minted@codewrapper{%
    \VerbatimInsertBuffer[buffername=minted@tmpcodebuffer,insertenvname=\minted@envname]}}
\def\MintedBegin@placeholder#1#2{%
  \begin{VerbatimBuffer}[buffername=minted@tmpcodebuffer]}
\def\MintedEnd@placeholder{%
  \end{VerbatimBuffer}%
  \minted@insertplaceholder}
\def\MintedBegin@verbatim#1#2{%
  \ifstrempty{#1}{}{\pgfkeys{/minted/cmd/.cd,#1}}%
  \edef\minted@lexer{#2}%
  \minted@usefvopts
  \minted@useadditionalfvoptsnopy
  \minted@usetexoptsnonpygments
  \begin{\minted@envname}}
\def\MintedEnd@verbatim{%
  \end{\minted@envname}}
\ifbool{minted@placeholder}%
 {\let\MintedBegin\MintedBegin@placeholder
  \let\MintedEnd\MintedEnd@placeholder}%
 {\ifbool{minted@verbatim}%
   {\let\MintedBegin\MintedBegin@verbatim
    \let\MintedEnd\MintedEnd@verbatim}%
   {\let\MintedBegin\MintedBegin@highlight
    \let\MintedEnd\MintedEnd@highlight}}
```

\inputminted

Highlight an external source file.

```
\def\minted@readinputmintedargs#1#{%
  \minted@readinputmintedargs@i{#1}}
\def\minted@readinputmintedargs@i#1#2#3{%
  \FVExtraAlwaysUnexpanded{\minted@readinputmintedargs#1{#2}{#3}}}
\FVExtrapdfstringdefDisableCommands{%
  \makeatletter
  \def\minted@readinputmintedargs@i#1#2#3{%
    \detokenize{<input from file "}#3\detokenize{">}}%
  \makeatother}
\def\inputminted{%
  \FVExtraRobustCommand\RobustInputMinted\minted@readinputmintedargs}
\FVExtrapdfstringdefDisableCommands{%
  \def\RobustInputMinted{}}
\newrobustcmd{\RobustInputMinted}[3][]{%
  \RobustInputMintedProcess{#1}{#2}{#3}}
\def\minted@define@inputfilepath#1{%
  \IfPackageLoadedTF{import}%
   {\edef\minted@inputfilepath{\import@path#1}}%
   {\edef\minted@inputfilepath{#1}}}
\def\RobustInputMintedProcess@highlight#1#2#3{%
  \begingroup
  \booltrue{minted@iscmd}%
  \ifstrempty{#1}{}{\pgfkeys{/minted/cmd/.cd,#1}}%
  \edef\minted@lexer{#2}%
  \minted@define@inputfilepath{#3}%
  \minted@usefvopts
  \minted@usetexoptsnonpygments
  \let\minted@highlight@fallback\RobustInputMintedProcess@highlight@fallback
  \minted@highlightinputfile
  \endgroup}
\def\RobustInputMintedProcess@highlight@fallback{%
  \minted@useadditionalfvoptsnopy
  \minted@codewrapper{%
    \csname\minted@envname Input\endcsname{\minted@inputfilepath}}}
\def\RobustInputMintedProcess@placeholder#1#2#3{%
  \begingroup
  \booltrue{minted@iscmd}%
  \minted@insertplaceholder
  \endgroup}
\def\RobustInputMintedProcess@verbatim#1#2#3{%
  \begingroup
  \booltrue{minted@iscmd}%
  \ifstrempty{#1}{}{\pgfkeys{/minted/cmd/.cd,#1}}%
  \edef\minted@lexer{#2}%
  \minted@define@inputfilepath{#3}%
  \minted@usefvopts
  \minted@useadditionalfvoptsnopy
  \minted@usetexoptsnonpygments
  \minted@codewrapper{%
    \csname\minted@envname Input\endcsname{\minted@inputfilepath}}%
  \endgroup}
\ifbool{minted@placeholder}%
 {\let\RobustInputMintedProcess\RobustInputMintedProcess@placeholder}%
 {\ifbool{minted@verbatim}%
   {\let\RobustInputMintedProcess\RobustInputMintedProcess@verbatim}%
   {\let\RobustInputMintedProcess\RobustInputMintedProcess@highlight}}
```

12.15 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.

```
\newcommand{\newminted}[3][]{%
  \ifstrempty{#1}%
   {\newminted@i{#2code}{#2}{#3}}%
   {\newminted@i{#1}{#2}{#3}}}
\begingroup
\catcode`\^^M=\active%
\gdef\newminted@i#1#2#3{%
  \expandafter\def\csname#1@i\endcsname##1{%
    \begin{minted}[#3,##1]{#2}^^M}%
  \newenvironment{#1}%
```

```
   {\VerbatimEnvironment%
    \FVExtraReadOArgBeforeVEnv{\csname#1@i\endcsname}}%
   {\end{minted}}%
  \newenvironment{#1*}[1]%
   {\VerbatimEnvironment%
    \begin{minted}[#3,##1]{#2}}%
   {\end{minted}}}%
\endgroup
```

\newmint

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

```
\newcommand{\newmint}[3][]{%
  \ifstrempty{#1}%
   {\edef\minted@tmp{#2}}%
   {\edef\minted@tmp{#1}}
  \expandafter\newmint@i\expandafter{\minted@tmp}{#2}{#3}}
\def\newmint@i#1#2#3{%
  \expandafter\newcommand\csname#1\endcsname{%
    \expandafter\FVExtraRobustCommand\csname RobustNewMint#1\endcsname
    \FVExtraUnexpandedReadStarOArgBVArg}%
  \FVExtrapdfstringdefDisableCommands{%
    \expandafter\def\csname RobustNewMint#1\endcsname{}}%
  \expandafter\newrobustcmd\csname RobustNewMint#1\endcsname{%
    \FVExtraReadOArgBeforeVArg{\csname RobustNewMint#1@i\endcsname}}%
  \expandafter\def\csname RobustNewMint#1@i\endcsname##1{%
    \ifbool{FVExtraRobustCommandExpanded}%
     {\@ifnextchar\bgroup
       {\FVExtraReadVArg{\csname RobustNewMint#1@ii\endcsname{##1}}}%
       {\minted@error{Delimiters must be paired curly braces in this context}}}%
     {\FVExtraReadVArg{\csname RobustNewMint#1@ii\endcsname{##1}}}}
  \expandafter\def\csname RobustNewMint#1@ii\endcsname##1##2{%
    \RobustMintProcess{#3,##1}{#2}{##2}}}
```

\newmintedfile

Define a new language-specific alias for \inputminted.

```
\def\minted@readnewmintedfileargs#1#{%
  \minted@readnewmintedfileargs@i{#1}}
\def\minted@readnewmintedfileargs@i#1#2{%
  \FVExtraAlwaysUnexpanded{\minted@readnewmintedfileargs#1{#2}}}
\FVExtrapdfstringdefDisableCommands{%
  \makeatletter
  \def\minted@readnewmintedfileargs@i#1#2{%
    \detokenize{<input from file "}#2\detokenize{">}}%
  \makeatother}
\newcommand{\newmintedfile}[3][]{%
  \ifstrempty{#1}%
   {\edef\minted@tmp{#2file}}%
   {\edef\minted@tmp{#1}}%
  \expandafter\newmintedfile@i\expandafter{\minted@tmp}{#2}{#3}}
\def\newmintedfile@i#1#2#3{%
  \expandafter\newcommand\csname#1\endcsname{%
    \expandafter\FVExtraRobustCommand\csname RobustNewMintedFile#1\endcsname
    \minted@readnewmintedfileargs}%
  \FVExtrapdfstringdefDisableCommands{%
    \expandafter\def\csname RobustNewMintedFile#1\endcsname{}}%
  \expandafter\newrobustcmd\csname RobustNewMintedFile#1\endcsname[2][]{%
    \RobustInputMintedProcess{#3,##1}{#2}{##2}}}
```

\newmintinline

Define an alias for \mintinline.

```
\newcommand{\newmintinline}[3][]{%
  \ifstrempty{#1}%
   {\edef\minted@tmp{#2inline}}%
   {\edef\minted@tmp{#1}}%
  \expandafter\newmintinline@i\expandafter{\minted@tmp}{#2}{#3}}
\def\newmintinline@i#1#2#3{%
  \expandafter\newcommand\csname#1\endcsname{%
    \expandafter\FVExtraRobustCommand\csname RobustNewMintInline#1\endcsname
    \FVExtraUnexpandedReadStarOArgBVArg}%
  \FVExtrapdfstringdefDisableCommands{%
    \expandafter\def\csname RobustNewMintInline#1\endcsname{}}%
  \expandafter\newrobustcmd\csname RobustNewMintInline#1\endcsname{%
    \FVExtraReadOArgBeforeVArg{\csname RobustNewMintInline#1@i\endcsname}}%
  \expandafter\def\csname RobustNewMintInline#1@i\endcsname##1{%
    \ifbool{FVExtraRobustCommandExpanded}%
     {\@ifnextchar\bgroup
       {\FVExtraReadVArg{\csname RobustNewMintInline#1@ii\endcsname{##1}}}%
       {\minted@error{Inline delimiters must be paired curly braces in this context}}}%
     {\FVExtraReadVArg{\csname RobustNewMintInline#1@ii\endcsname{##1}}}}
  \expandafter\def\csname RobustNewMintInline#1@ii\endcsname##1##2{%
    \RobustMintInlineProcess{#3,##1}{#2}{##2}}}
```

12.16 Float support

listing *(env.)*

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

```
\ifbool{minted@newfloat}%
 {\@ifundefined{minted@float@within}%
   {\DeclareFloatingEnvironment[fileext=lol,placement=tbp]{listing}}%
   {\def\minted@tmp#1{%
      \DeclareFloatingEnvironment[fileext=lol,placement=tbp,within=#1]{listing}}%
    \expandafter\minted@tmp\expandafter{\minted@float@within}}}%
 {\@ifundefined{minted@float@within}%
   {\newfloat{listing}{tbp}{lol}}%
   {\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.

```
\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.

```
\newcommand{\listingscaption}{Listing}
```

The following definition should not be changed by the user.

```
\floatname{listing}{\listingscaption}
```

`\listoflistingscaption`

The caption that is displayed for the list of listings.

```
\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.

```
\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.

```
\fi
```