

The `minted` package: Highlighted source code in L^AT_EX

Geoffrey M. Poore

`gpoore@gmail.com`

`github.com/gpoore/minted`

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

v2.1 from 2015/09/09

Abstract

`minted` is a package that facilitates expressive syntax highlighting using the powerful `Pygments` library. The package also provides options to customize the highlighted source code output.

License

LaTeX Project Public License (LPPL) version 1.3.

Additionally, the project may be distributed under the terms of the 3-Clause (“New”) BSD license: <http://opensource.org/licenses/BSD-3-Clause>.

Contents

1	Introduction	4
2	Installation	4
2.1	Prerequisites	4
2.2	Required packages	5
2.3	Installing <code>minted</code>	5
3	Transitioning to version 2	6
4	Basic usage	6
4.1	Preliminary	6
4.2	A minimal complete example	7
4.3	Formatting source code	8
4.4	Using different styles	9
4.5	Supported languages	9
5	Floating listings	9
6	Options	11
6.1	Package options	11
6.2	Macro option usage	14
6.3	Available options	15
7	Defining shortcuts	23
8	FAQ and Troubleshooting	25
	Version History	28
9	Implementation	32
9.1	Required packages	32
9.2	Package options	33
9.3	Input, caching, and temp files	35
9.4	OS interaction	37
9.5	Option processing	39
9.6	Additions to <code>fancyvrb</code>	54
9.6.1	Setup	54

9.6.2	Line breaking	55
9.7	linenos	67
9.8	Cleanup	67
9.9	Internal helpers	67
9.10	Public API	75
9.11	Command shortcuts	79
9.12	Float support	81
9.13	Epilogue	82
9.14	Final cleanup	82
10	Implementation of compatibility package	83

1 Introduction

minted is a package that allows formatting source code in L^AT_EX. For example:

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

will highlight a piece of code in a chosen language. The appearance can be customized with a number of options and color schemes.

Unlike some other packages, most notably `listings`, `minted` requires the installation of additional software, `Pygments`. This may seem like a disadvantage, but there are also significant advantages.

`Pygments` provides superior syntax highlighting compared to conventional packages. For example, `listings` basically only highlights strings, comments and keywords. `Pygments`, on the other hand, can be completely customized to highlight any kind of token the source language might support. This might include special formatting sequences inside strings, numbers, different kinds of identifiers and exotic constructs such as HTML tags.

Some languages make this especially desirable. Consider the following Ruby code as an extreme, but at the same time typical, example:

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

Here we have four different colors for identifiers (five, if you count keywords) and escapes from inside strings, none of which pose a problem for `Pygments`.

Additionally, installing `Pygments` is actually incredibly easy (see the next section).

2 Installation

2.1 Prerequisites

`Pygments` is written in Python, so make sure that you have Python 2.6 or later installed on your system. This may be easily checked from the command line:

```
$ python --version
Python 2.7.5
```

If you don't have Python installed, you can download it from the [Python website](#) or use your operating system's package manager.

Some Python distributions include `Pygments` (see some of the options under "Alternative Implementations" on the Python site). Otherwise, you will need to install `Pygments` manually. This may be done by installing `setuptools`, which facilitates the distribution of Python applications. You can then install `Pygments` using the following command:

```
$ sudo easy_install Pygments
```

Under Windows, you will not need the `sudo`, but may need to run the command prompt as administrator. `Pygments` may also be installed with `pip`:

```
$ pip install Pygments
```

If you already have `Pygments` installed, be aware that the latest version is recommended (at least 1.4 or later). Some features, such as `escapeinside`, will only work with 2.0+. `minted` may work with versions as early as 1.2, but there are no guarantees.

2.2 Required packages

`minted` requires that the following packages be available and reasonably up to date on your system. All of these ship with recent `TeX` distributions.

- `keyval`
- `ifthen`
- `etoolbox`
- `kvoptions`
- `calc`
- `xstring`
- `fancyvrb`
- `ifplatform`
- `xcolor`
- `float`
- `pdftexcmds`
- `lineno`

2.3 Installing `minted`

You can probably install `minted` with your `TeX` distribution's package manager. Otherwise, or if you want the absolute latest version, you can install it manually by following the directions below.

You may download `minted.sty` from the [project's homepage](#). We have to install the file so that `TeX` is able to find it. In order to do that, please refer to the [TeX FAQ](#). If you just want to experiment with the latest version, you could locate your current `minted.sty` in your `TeX` installation and replace it with the latest version. Or you could just put the latest `minted.sty` in the same directory as the file you wish to use it with.

3 Transitioning to version 2

Transitioning from `minted` 1.7 to 2.0+ should require no changes in almost all cases. Version 2 provides the same interface and all of the same features.

In cases when custom code was used to hook into the `minted` internals, it may still be desirable to use the old `minted` 1.7. For those cases, the new package `minted1` is provided. Simply load this before any other package attempts to load `minted`, and you will have the code from 1.7.

A brief summary of new features in version 2.0 is provided below. More detail is available in the [Version History](#).

- New inline command `\mintinline`.
- Support for caching highlighted code with new package option `cache`. This drastically reduces package overhead. Caching is on by default. A cache directory called `_minted-(document name)` will be created in the document root directory. This may be modified with the `cachedir` package option.
- Automatic line breaking for all commands and environments with new option `breaklines`. Many additional options for customizing line breaking.
- Support for Unicode under the pdfTeX engine.
- Set document-wide options using `\setminted{<opts>}`. Set language-specific options using `\setminted[<lang>]{<opts>}`. Similarly, set inline-specific options using `\setmintedinline`.
- Package option `langlinenos`: do line numbering by language.
- Many new options, including `encoding`, `autogobble`, and `escapeinside` (requires Pygments 2.0+).
- New package option `outputdir` provides compatibility with command-line options `-output-directory` and `-aux-directory`.
- New package option `draft` disables Python use to give maximum performance.
- `\mint` can now take code delimited by matched curly braces `{}`.

4 Basic usage

4.1 Preliminary

Since `minted` makes calls to the outside world (that is, Pygments), you need to tell the L^AT_EX processor about this by passing it the `-shell-escape` option or it won't allow such calls. In effect, instead of calling the processor like this:

```
$ latex input
```

you need to call it like this:

```
$ latex -shell-escape input
```

The same holds for other processors, such as `pdflatex` or `xelatex`.

You should be aware that using `-shell-escape` allows \LaTeX to run potentially arbitrary commands on your system. It is probably best to use `-shell-escape` only when you need it, and to use it only with documents from trusted sources.

Working with OS X

If you are using `minted` with some versions/configurations of OS X, and are using caching with a large number of code blocks (> 256), you may receive an error like

```
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 the OS X commands `launchctl limit maxfiles` or `ulimit -n` to increase the number of files that may be used.

4.2 A minimal complete example

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

```
\documentclass{article}

\usepackage{minted}

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

By compiling the source file like this:

```
$ pdflatex -shell-escape minimal
```

we end up with the following output in `minimal.pdf`:

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

4.3 Formatting source code

`minted` Using `minted` is straightforward. For example, to highlight some Python source code we might use the following code snippet (result on the right):

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

Optionally, the environment accepts a number of options in `key=value` notation, which are described in more detail below.

`\mint` For a single line of source code, you can alternatively use a shorthand notation:

```
\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}delimcodedelim`, where the code delimiter can be almost any punctuation character. The *code* may also be delimited with matched curly braces {}, so long as *code* itself does not contain unmatched curly braces. Again, this command supports a number of options described below.

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:

```
X\mintinline{python}{print(x**2)}X Xprint(x**2)X
```

The syntax is `\mintinline[options]{language}delimcodedelim`. The delimiters can be a pair of characters, as for `\mint`. They can also be a matched pair of curly braces, {}.

The command has been carefully crafted so that in most cases it will function correctly when used inside other commands.¹

¹For example, `\mintinline` works in footnotes! The main exception is when the code

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

4.4 Using different styles

`\usemintedstyle` Instead of using the default style you may choose another stylesheet provided by Pygments. This may be done via the following:

```
\usemintedstyle{name}
```

The full syntax is `\usemintedstyle[<language>]{<style>}`. 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 `\setminted` and via the optional argument for each command and environment.²

To get a list of all available stylesheets, see the online demo at the [Pygments website](#) or execute the following command on the command line:

```
$ pygmentize -L styles
```

Creating your own styles is also easy. Just follow the instructions provided on the [Pygments website](#).

4.5 Supported languages

Pygments supports over 300 different programming languages, template languages, and other markup languages. To see an exhaustive list of the currently supported languages, use the command

```
$ pygmentize -L lexers
```

5 Floating listings

`listing` `minted` provides the `listing` environment to wrap around a source code block. This puts the code into a floating box. You can also provide a `\caption` and a `\label` for such a listing in the usual way (that is, as for the `table` and `figure` environments):

contains the percent `%` or hash `#` characters, or unmatched curly braces.

²Version 2.0 added the optional language argument and removed the restriction that the command be used in the preamble.

```

\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.

will yield:

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

Listing 1: Example of a listing.

Listing 1 contains an example of a listing.

`\listoflistings`

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

<code>\listoflistings</code>	List of Listings
	1 Example of a listing. 10

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`

(Only applies when package option `newfloat` is not used.) The string “Listing”

in a listing’s caption can be changed. To do this, simply redefine the macro `\listingscaption`, for example:

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

`\listoflistingscaption` (Only applies when package option `newfloat` is not used.) Likewise, the caption of the listings list, “List of Listings,” can be changed by redefining `\listoflistingscaption`:

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

6 Options

6.1 Package options

`chapter` To control how L^AT_EX counts the `listing` floats, you can pass either the `section` or `chapter` 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 the code via Pygments and saving the output to another temporary file, and inputting the output into the L^AT_EX 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-<jobname>` in the document’s root directory (this may be customized with the `cachedir` option).³ Files of highlighted code are stored in this directory, so that the code will not have to be highlighted again in the future. In most cases, caching will significantly speed up document compilation.

Cached files that are no longer in use are automatically deleted.⁴

`cachedir=<directory>` `(def: _minted-<jobname>)` This allows the directory in which cached files are stored to be specified. Paths

³The directory is actually named using a “sanitized” copy of `<jobname>`, in which spaces and asterisks have been replaced by underscores, and double quotation marks have been stripped. If the file name contains spaces, `\jobname` will contain a quote-wrapped name, except under older versions of MiKTeX which used the name with spaces replaced by asterisks. Using a “sanitized” `<jobname>` is simpler than accomodating the various escaping conventions.

⁴This depends on the main auxiliary file not being deleted or becoming corrupted. If that happens, you could simply delete the cache directory and start over.

should use forward spaces, even under Windows.

Note that this directory is relative to the `outputdir`, if an `outputdir` is specified.

```
draft={boolean}
(default: false)
```

This uses `fancyvrb` alone for all typesetting; `Pygments` is not used. This trades syntax highlighting and some other `minted` features for faster compiling. Performance should be essentially the same as using `fancyvrb` directly; no external temporary files are used. Note that if you are not changing much code between compiles, the difference in performance between caching and draft mode may be minimal. Also note that `draft` settings are typically inherited from the document class.

Draft mode does not support `autogobble`. Regular `gobble`, `linenos`, and most other options not related to syntax highlighting will still function in draft mode.

Documents can usually be compiled without shell escape in draft mode. The `ifplatform` package may issue a warning about limited functionality due to shell escape being disabled, but this may be ignored in almost all cases. (Shell escape is only really required if you have an unusual system configuration such that the `\ifwindows` macro must fall back to using shell escape to determine the system. See the `ifplatform` documentation for more details: <http://www.ctan.org/pkg/ifplatform>.)

If the `cache` option is set, then all existing cache files will be kept while draft mode is on. This allows caching to be used intermitently with draft mode without requiring that the cache be completely recreated each time. Automatic cleanup of cached files will resume as soon as draft mode is turned off. (This assumes that the auxiliary file has not been deleted in the meantime; it contains the cache history and allows automatic cleanup of unused files.)

```
final={boolean}
(default: true)
```

This is the opposite of `draft`; it is equivalent to `draft=false`. Again, note that `draft` and `final` settings are typically inherited from the document class.

```
kpsewhich={boolean}
(default: false)
```

This option uses `kpsewhich` to locate files that are to be highlighted. Some build tools such as `texi2pdf` function by modifying `TEXINPUTS`; in some cases, users may customize `TEXINPUTS` as well. The `kpsewhich` option allows `minted` to work with such configurations.

This option may add a noticeable amount of overhead on some systems, or with some system configurations.

This option does *not* make `minted` work with the `-output-directory` and `-aux-directory` command-line options for \LaTeX . For those, see the `outputdir` package option.

Under Windows, this option currently requires that PowerShell be installed. It may need to be installed in versions of Windows prior to Windows 7.

```
langlinenos={boolean}
(default: false)
```

`minted` uses the `fancyvrb` package behind the scenes for the code typesetting. `fancyvrb` provides an option `firstnumber` that allows the starting line number of an environment to be specified. For convenience, there is an option `firstnumber=last` that allows line numbering to pick up where it left off. The `langlinenos` option

makes `firstnumber` work for each language individually with all `minted` and `\mint` usages. For example, consider the code and output below.

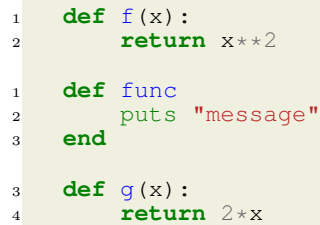
```

\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 `lanlincenos` 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.

```

outputdir=<directory>
(default: <none>)

```

The `-output-directory` and `-aux-directory` (MiKTeX) command-line options for `LATEX` causes problems for `minted`, because the `minted` temporary files are saved in `<outputdir>`, but `minted` still looks for them in the document root directory. There is no way to access the value of the command-line option so that `minted` can automatically look in the right place. But it is possible to allow the output directory to be specified manually as a package option.

The output directory should be specified using an absolute path or a path relative to the document root directory. Paths should use forward spaces, even under Windows. Paths that include spaces are not allowed.

```

section

```

To control how `LATEX` counts the `listing` floats, you can pass either the `section` or `chapter` option when loading the `minted` package.

6.2 Macro option usage

All minted highlighting commands 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 “=”). To customize the display of the line numbers further, override the `\theFancyVerbLine` command. Consult the `fancyvrb` documentation for details.

`\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]{python}
# Returns  $\sum_{i=1}^n i$ 
def sum_from_one_to(n):
    r = range(1, n + 1)
    return sum(r)
\end{minted}
# Returns  $\sum_{i=1}^n i$ 
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 these unnecessary whitespace characters from the output. There is also an `autogobble` option that detects the length of this whitespace automatically.

```
\begin{minted}[gobble=2,
showspaces]{python}
def boring(args = None):
    pass
\end{minted}
def_boring(args=_None):
    pass

versus

\begin{minted}[showspaces]{python}
def boring(args = None):
    pass
\end{minted}
def_boring(args=_None):
    pass
```

`\setminted`

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

options override language-specific options.

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

6.3 Available options

Following is a full list of available options. For more detailed option descriptions please refer to the `fancyvrb` and `Pygments` documentation.

`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]{python} ...text.
    def f(x):
        return x**2
\end{minted}
def f(x):
    return x**2
```

`baselinestretch` (auto|dimension) (default: `auto`)
Value to use as for `baselinestretch` inside the listing.

`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 `#`, `{`, `}`, `%`, `[`, `]`; the backslash `\` may be obtained via `\\`).

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

some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCould
↪ NeverFitOnOneLine'
```

`breakaftergroup` (boolean) (default: `true`)
When `breakafter` is used, group all adjacent identical characters together, and only allow a break after the last character.

`breakaftersymbolpre` (string) (default: `\, \footnotesize\ensuremath{_}\rfloor`, `_`)

The symbol inserted pre-break for breaks inserted by `breakafter`.

`breakaftersymbolpost` (string) (default: `<none>`)
The symbol inserted post-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 = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNev
↪ erFitOnOneLine'
```

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

`breakanywheresymbolpost` (string) (default: `<none>`)
The symbol inserted post-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`.

`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 is not compatible with draft mode. A complete list of Pygments tokens is available at <http://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: `0pt`)
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`.

`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`, and `breakafter` for more fine-tuned breaking. Using `escapeinside` to escape to L^AT_EX 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.)

<pre>...text. \begin{minted}[breaklines]{python} def f(x): return 'Some text ' + str(x) \end{minted}</pre>	<pre>...text. def f(x): return 'Some text ' + ↪ str(x)</pre>
--	--

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.

Automatic line breaks will not work with `showspaces=true` unless you use `breakanywhere`. You may be able to change the definition of `\FV@Space` if you need this; see the `fancyvrb` implementation for details.

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

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

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

Does not apply to `\mintinline`.

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

`breaksymbolindent` (dimension) (default: `breaksymbolindentleft`)
Alias for `breaksymbolindentleft`.

`breaksymbolindentleft` (dimension) (default: width of 4 characters in teletype font at default point size)
The extra left indentation that is provided to make room for `breaksymbolleft`. This indentation is only applied when there is a `breaksymbolleft`.

This may be set to the width of a specific number of (fixed-width) characters by using an approach such as

```
\newdimen\temporarydimen
\settowidth{\temporarydimen}{\ttfamily aaaa}
```

and then using `breaksymbolindentleft=\temporarydimen`.

Does not apply to `\mintinline`.

`breaksymbolindentrigh` (dimension) (default: width of 4 characters in teletype font at default point size)

The extra right indentation that is provided to make room for `breaksymbolright`. This indentation is only applied when there is a `breaksymbolright`.

<code>breaksymbolsep</code>	(dimension)	(default: <code>breaksymbolsepleft</code>)
	Alias for <code>breaksymbolsepleft</code>	
<code>breaksymbolsepleft</code>	(dimension)	(default: <code>1em</code>)
	The separation between the <code>breaksymbolleft</code> and the adjacent code. Does not apply to <code>\mintinline</code> .	
<code>breaksymbolsepright</code>	(dimension)	(default: <code>1em</code>)
	The separation between the <code>breaksymbolright</code> and the adjacent code.	
<code>bgcolor</code>	(string)	(default: <code><none></code>)
	Background color of the listing. Notice that the value of this option must <i>not</i> be a color command. Instead, it must be a color <i>name</i> , given as a string, of a previously-defined color:	

```
\definecolor{bg}{rgb}{0.95,0.95,0.95}
\begin{minted}[bgcolor=bg]{php}
<?php
  echo "Hello, $x";
?>
\end{minted}
```

This option puts `minted` environments and `\mint` commands in a `minipage` with a colored background. It puts `\mintinline` inside a `\colorbox`. If you want to use `\setminted` to set background colors, and only want background colors on `minted` and `\mint`, you may use `\setmintedinline{bgcolor={}}` to turn off the coloring for inline commands.

This option will prevent `breaklines` from working with `\mintinline`. A `\colorbox` cannot break across lines.

This option will prevent environments from breaking across pages. If you want support for page breaks and advanced options, you should consider a framing package such as `framed`, `mdframed`, or `tcolorbox`. It is easy to add framing to `minted` commands and environments using the `etoolbox` package. For example, using `mdframed`:

```
\BeforeBeginEnvironment{minted}{\begin{mdframed}}
\AfterEndEnvironment{minted}{\end{mdframed}}
```

Some framing packages also provide built-in commands for such purposes. For example, `mdframed` provides a `\surroundwithmdframed` command, which could be used to add a frame to all `minted` environments:

```
\surroundwithmdframed{minted}
```

`tcolorbox` even provides a built-in framing environment with `minted` support.

<code>codetagify</code>	(list of strings)	(default: highlight XXX, TODO, BUG, and NOTE)
	Highlight special code tags in comments and docstrings.	
<code>encoding</code>	(string)	(default: system-specific)
	Sets the file encoding that Pygments expects. See also <code>outencoding</code> .	
<code>escapeinside</code>	(string)	(default: <code><none></code>)
	Escape to \LaTeX between the two characters specified in (string). 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. Special \LaTeX characters must be escaped when they are used as the escape characters (for example, <code>escapeinside=\#\%</code>). Requires Pygments 2.0+.	

Escaping does not work inside strings and comments (for comments, there is `texcomments`). As of Pygments 2.0.2, this means that 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](#).

```

\begin{minted}[escapeinside=||]{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, in a few cases ligature handling may need to be modified. The single-quote character (') is normally a shortcut for \prime in math mode, but this is disabled in verbatim content as a byproduct of ligatures being disabled. For the same reason, any package that relies on 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 <http://tex.stackexchange.com/questions/223876>. `minted` currently does not attempt to patch `\@noligs` due to the potential for package conflicts.

<code>firstline</code>	(integer)	(default: 1)
	The first line to be shown. All lines before that line are ignored and do not appear in the output.	
<code>firstnumber</code>	(auto integer)	(default: auto = 1)
	Line number of the first line.	
<code>fontfamily</code>	(family name)	(default: <code>tt</code>)
	The font family to use. <code>tt</code> , <code>courier</code> and <code>helvetica</code> are pre-defined.	
<code>fontseries</code>	(series name)	(default: auto – the same as the current font)
	The font series to use.	
<code>fontsize</code>	(font size)	(default: auto – the same as the current font)

	The size of the font to use, as a size command, e.g. <code>\footnotesize</code> .	
<code>fontshape</code>	(font shape) (default: <code>auto</code> – the same as the current font) The font shape to use.	
<code>formatcom</code>	(command) (default: <code><none></code>) A format to execute before printing verbatim text.	
<code>frame</code>	(<code>none leftline topline bottomline lines single</code>) (default: <code>none</code>) The type of frame to put around the source code listing.	
<code>framerule</code>	(dimension) (default: <code>0.4pt</code>) Width of the frame.	
<code>framesep</code>	(dimension) (default: <code>\fboxsep</code>) Distance between frame and content.	
<code>funcnamehighlighting</code>	(boolean) (default: <code>true</code>) [For PHP only] If <code>true</code> , highlights built-in function names.	
<code>gobble</code>	(integer) (default: <code>0</code>) Remove the first n characters from each input line.	
<code>keywordcase</code>	(string) (default: <code>'lower'</code>) Changes capitalization of keywords. Takes <code>'lower'</code> , <code>'upper'</code> , or <code>'capitalize'</code> .	
<code>label</code>	(string) (default: <code>empty</code>) Add a label to the top, the bottom or both of the frames around the code. See the <code>fancyvrb</code> documentation for more information and examples. <i>Note:</i> This does <i>not</i> add a <code>\label</code> to the current listing. To achieve that, use a floating environment (section 5) instead.	
<code>labelposition</code>	(<code>none topline bottomline all</code>) (default: <code>topline</code> , <code>all</code> or <code>none</code>) Position where to print the label (see above; default: <code>topline</code> if one label is defined, <code>all</code> if two are defined, <code>none</code> else). See the <code>fancyvrb</code> documentation for more information.	
<code>lastline</code>	(integer) (default: <i>last line of input</i>) The last line to be shown.	
<code>linenos</code>	(boolean) (default: <code>false</code>) Enables line numbers. In order to customize the display style of line numbers, you need to redefine the <code>\theFancyVerbLine</code> macro:	

```

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

\begin{minted}[linenos,                11 def all(iterable):
firstnumber=11]{python}              12     for i in iterable:
def all(iterable):                    13         if not i:
    for i in iterable:                14             return False
        if not i:                      15     return True
            return False
    return True
\end{minted}

```

<code>numbers</code>	(left right) (default: <i>none</i>) Essentially the same as <code>linenos</code> , except the side on which the numbers appear may be specified.
<code>mathescape</code>	(boolean) (default: <i>false</i>) Enable L ^A T _E X math mode inside comments. Usage as in package <code>listings</code> . See the note under <code>escapeinside</code> regarding math and ligatures.
<code>numberblanklines</code>	(boolean) (default: <i>true</i>) Enables or disables numbering of blank lines.
<code>numbersep</code>	(dimension) (default: 12pt) Gap between numbers and start of line.
<code>obeytabs</code>	(boolean) (default: <i>false</i>) Treat tabs as tabs instead of converting them to spaces.
<code>outencoding</code>	(string) (default: system-specific) Sets the file encoding that <code>Pygments</code> uses for highlighted output. Overrides any encoding previously set via <code>encoding</code> .
<code>python3</code>	(boolean) (default: <i>false</i>) [For <code>PythonConsoleLexer</code> only] Specifies whether Python 3 highlighting is applied.
<code>resetmargins</code>	(boolean) (default: <i>false</i>) Resets the left margin inside other environments.
<code>rulecolor</code>	(color command) (default: <i>black</i>) The color of the frame.
<code>samepage</code>	(boolean) (default: <i>false</i>) Forces the whole listing to appear on the same page, even if it doesn't fit.
<code>showspaces</code>	(boolean) (default: <i>false</i>) Enables visible spaces: <code>visible_spaces</code> .
<code>showtabs</code>	(boolean) (default: <i>false</i>) Enables visible tabs—only works in combination with <code>obeytabs</code> .
<code>startinline</code>	(boolean) (default: <i>false</i>) [For <code>PHP</code> only] Specifies that the code starts in <code>PHP</code> mode, i.e., leading <code><?php</code> is omitted.

<code>style</code>	(string)	(default: <i>default</i>)
	Sets the stylesheet used by Pygments.	
<code>stepnumber</code>	(integer)	(default: 1)
	Interval at which line numbers appear.	
<code>stripall</code>	(boolean)	(default: <i>false</i>)
	Strip all leading and trailing whitespace from the input.	
<code>stripnl</code>	(boolean)	(default: <i>true</i>)
	Strip leading and trailing newlines from the input.	
<code>tabsize</code>	(integer)	(default: 8)
	The number of spaces a tab is equivalent to. If <code>obeytabs</code> is <i>not</i> active, tabs will be converted into this number of spaces. If <code>obeytabs</code> is active, tab stops will be set this number of space characters apart.	
<code>texcl</code>	(boolean)	(default: <i>false</i>)
	Enables \LaTeX code inside comments. Usage as in package <code>listings</code> . See the note under <code>escapeinside</code> regarding math and ligatures.	
<code>texcomments</code>	(boolean)	(default: <i>false</i>)
	Enables \LaTeX code inside comments. The newer name for <code>texcl</code> . See the note under <code>escapeinside</code> regarding math and ligatures.	
	As of Pygments 2.0.2, <code>texcomments</code> fails with multiline C/C++ preprocessor directives, and may fail in some other circumstances. This is because preprocessor directives are <i>tokenized as <code>Comment.Preproc</code></i> , so <code>texcomments</code> causes preprocessor directives to be treated as literal \LaTeX code. <i>An issue has been opened at the Pygments site</i> ; additional details are also available on the <i>minted GitHub site</i> .	
<code>xleftmargin</code>	(dimension)	(default: 0)
	Indentation to add before the listing.	
<code>xrightmargin</code>	(dimension)	(default: 0)
	Indentation to add after the listing.	

7 Defining shortcuts

Large documents with a lot of listings will nonetheless use the same source language and the same set of options for most listings. Always specifying all options is redundant, a lot to type and makes performing changes hard.

One option is to use `\setminted`, but even then you must still specify the language each time.

`minted` therefore defines a set of commands that lets you define shortcuts for the highlighting commands. Each shortcut is specific for one programming language.

`\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}
1 template <typename T>
2 T id(T value) {
3     return value;
4 }

```

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;

```

Notice the star “*” behind the environment name—due to restrictions in fancyvrb’s handling of options, it is necessary to provide a *separate* environment that accepts options, and the options are *not* optional on the starred version of the environment.

The default name of the environment is `<language>code`. If this name clashes with another environment or if you want to choose an own name for another reason, you may do so by specifying it as the first argument: `\newminted[<environment name>]{<language>}{<options>}`.

`\newmint`

The above macro only defines shortcuts for the minted environment. The main reason is that the short command form `\mint` often needs different options—at the very least, it will generally not use the `gobble` option. A shortcut for `\mint` is defined using `\newmint[<macro name>]{<language>}{<options>}`. The arguments and usage are identical to `\newminted`. If no `<macro name>` is specified, `<language>` 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>]{<language>}{<options>}`. If a `<macro name>` is not specified, then the created macro is called `\<language>inline`.

```

\newmintinline{perl}{showspaces}
X\perlinline/my $foo = $bar;/X
Xmy_$foo=_$bar;X

```

`\newmintedfile`

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

```

\newmintedfile[<macro name>]{<language>}{<options>}

```

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

8 FAQ and Troubleshooting

In some cases, `minted` may not give the desired result due to other document settings that it cannot control. 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 receive a “Too many open files” error under OS X when using caching.** See the note on OS X under Section 4.1.
- **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`.
- **Quotation marks and backticks don't look right. Backtick characters ``` are appearing as left quotes. Single quotes are appearing as curly right quotes.** This is due to how Pygments outputs L^AT_EX code, combined with how L^AT_EX deals with verbatim content. Try `\usepackage{upquote}`.
- **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.
- **Tabs are eaten by Beamer.** This is due to a [bug in Beamer's treatment of verbatim content](#). Upgrade Beamer or use the linked patch. Otherwise, try `fragile=singleslide` if you don't need overlays, or consider using `\inputminted` or converting the tabs into spaces.

- **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 `keyval` works (`minted` uses it to manage options). Arguments are not expanded. See [this](#) and [this](#) for more information. It is still possible to do what you want; you just need to expand the options macro before passing it to the commands that create the new commands/environments. An example is shown below. The `\expandafter` is the vital part.

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

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

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

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

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

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

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

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

- **Tabs are being turned into the character sequence `^^I`.** This happens when you use XeLaTeX. You need to use the `-8bit` command-line option so that tabs may be written correctly to temporary files. See <http://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 <http://tex.stackexchange.com/a/53540/10742> for more on listing environments with page breaks.

- **I want to use a custom script/executable to access Pygments, rather than `pygmentize`.** Redefine `\MintedPygmentize`:

```
\renewcommand{\MintedPygmentize}{...}
```

- **I want to use the command-line option `-output-directory`, or MiKTeX's `-aux-directory`, but am getting errors.** Use the package option `outputdir` to specify the location of the output directory. Unfortunately, there is no way for `minted` to detect the output directory automatically.
- **I want extended characters in frame labels, but am getting errors.** This can happen with `minted` <2.0 and Python 2.7, due to a [terminal encoding issue with Pygments](#). It should work with any version of Python with `minted` 2.0+, which processes labels internally and does not send them to Python.
- **`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.

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.

Version History

v2.1 (2015/09/09)

- Changing the highlighting style now no longer involves re-highlighting code. Style may be changed with almost no overhead.
- Improved control of automatic line breaks. New option `breakanywhere` allows line breaks anywhere when `breaklines=true`. The pre-break and post-break symbols for these types of breaks may be set with `breakanywheresymbolpre` and `breakanywheresymbolpost` (#79). New option `breakafter` allows specifying characters after which line breaks are allowed. Breaks between adjacent, identical characters may be controlled with `breakaftergroup`. The pre-break and post-break symbols for these types of breaks may be set with `breakaftersymbolpre` and `breakaftersymbolpost`.
- `breakbytoken` now only breaks lines between tokens that are separated by spaces, matching the documentation. The new option `breakbytokenanywhere` allows for breaking between tokens that are immediately adjacent. Fixed a bug in `\mintinline` that produced a following linebreak when `\mintinline` was the first thing in a paragraph and `breakbytoken` was true (#77).
- Fixed a bug in draft mode option handling for `\inputminted` (#75).
- Fixed a bug with `\MintedPygmentize` when a custom `pygmentize` was specified and there was no `pygmentize` on the default path (#62).
- Added note to docs on caching large numbers of code blocks under OS X (#78).
- Added discussion of current limitations of `texcomments` (#66) and `escapeinside` (#70).
- PGF/TikZ externalization is automatically detected and supported (#73).
- The package is now compatible with L^AT_EX files whose names contain spaces (#85).

v2.0 (2015/01/31)

- Added the compatibility package `minted1`, which provides the `minted` 1.7 code. This may be loaded when 1.7 compatibility is required. This package works with other packages that `\RequirePackage{minted}`, so long as it is loaded first.
- Moved all old `\changes` into `changelog`.

Development releases for 2.0 (2014–January 2015)

- Caching is now on by default.
- Fixed a bug that prevented compiling under Windows when file names contained commas.
- Added `breaksymbolleft`, `breaksymbolsepleft`, `breaksymbolindentleft`, `breaksymbolright`, `breaksymbolsepright`, and `breaksymbolindentright` options. `breaksymbol`, `breaksymbolsep`, and `breaksymbolindent` are now aliases for the correspondent `*left` options.
- Added `kpsewhich` package option. This uses `kpsewhich` to locate the files that are to be highlighted. This provides compatibility with build tools like `texi2pdf` that function by modifying `TEXINPUTS` (#25).
- Fixed a bug that prevented `\inputminted` from working with `outputdir`.
- Added informative error messages when Pygments output is missing.
- Added `final` package option (opposite of `draft`).
- Renamed the default cache directory to `_minted-<jobname>` (replaced leading period with underscore). The leading period caused the cache directory to be hidden on many systems, which was a potential source of confusion.
- `breaklines` and `breakbytoken` now work with `\mintinline` (#31).
- `bgcolor` may now be set through `\setminted` and `\setmintedinline`.
- When math is enabled via `texcomments`, `mathescape`, or `escapeinside`, space characters now behave as in normal math by vanishing, instead of appearing as literal spaces. Math need no longer be specially formatted to avoid undesired spaces.
- In default value of `\listoflistingscaption`, capitalized “Listings” so that capitalization is consistent with default values for other lists (figures, tables, algorithms, etc.).
- Added `newfloat` package option that creates the `listing` environment using `newfloat` rather than `float`, thus providing better compatibility with the `caption` package (#12).
- Added support for Pygments option `stripall`.
- Added `breakbytoken` option that prevents `breaklines` from breaking lines within Pygments tokens.
- `\mintinline` uses a `\colorbox` when `bgcolor` is set, to give more reasonable behavior (#57).
- For PHP, `\mintinline` automatically begins with `startinline=true` (#23).
- Fixed a bug that threw off line numbering in `minted` when `langlinenos=false` and `firstnumber=last`. Fixed a bug in `\mintinline` that threw off subsequent line numbering when `langlinenos=false` and `firstnumber=last`.

- Improved behavior of `\mint` and `\mintinline` in draft mode.
- The `\mint` command now has the additional capability to take code delimited by paired curly braces `{}`.
- It is now possible to set options only for `\mintinline` using the new `\setmintedinline` command. Inline options override options specified via `\setminted`.
- Completely rewrote option handling. `fancyvrb` options are now handled on the \LaTeX side directly, rather than being passed to Pygments and then returned. This makes caching more efficient, since code is no longer rehighlighted just because `fancyvrb` options changed.
- Fixed buffer size error caused by using `cache` with a very large number of files (#61).
- Fixed `autogobble` bug that caused failure under some operating systems.
- Added support for `escapeinside` (requires Pygments 2.0+; #38).
- Fixed issues with XeTeX and caching (#40).
- The `upquote` package now works correctly with single quotes when using Pygments 1.6+ (#34).
- Fixed caching incompatibility with Linux and OS X under `xelatex` (#18 and #42).
- Fixed `autogobble` incompatibility with Linux and OS X.
- `\mintinline` and derived commands are now robust, via `\newrobustcmd` from `etoolbox`.
- Unused styles are now cleaned up when caching.
- Fixed a bug that could interfere with caching (#24).
- Added `draft` package option (#39). This typesets all code using `fancyvrb`; Pygments is not used. This trades syntax highlighting for maximum speed in compiling.
- Added automatic line breaking with `breaklines` and related options (#1).
- Fixed a bug with boolean options that needed a `False` argument to cooperate with `\setminted` (#48).

v2.0-alpha3 (2013/12/21)

- Added `autogobble` option. This sends code through Python's `textwrap.dedent()` to remove common leading whitespace.
- Added package option `cachedir`. This allows the directory in which cached content is saved to be specified.
- Added package option `outputdir`. This allows an output directory for temporary files to be specified, so that the package can work with LaTeX's `-output-directory` command-line option.

- The `kvoptions` package is now required. It is needed to process key-value package options, such as the new `cachedir` option.
- Many small improvements, including better handling of paths under Windows and improved key system.

v2.0-alpha2 (2013/08/21)

- `\DeleteFile` now only deletes files if they do indeed exist. This eliminates warning messages due to missing files.
- Fixed a bug in the definition of `\DeleteFile` for non-Windows systems.
- Added support for Pygments option `stripnl`.
- Settings macros that were previously defined globally are now defined locally, so that `\setminted` may be confined by `\begingroup... \endgroup` as expected.
- Macro definitions for a given style are now loaded only once per document, rather than once per command/environment. This works even without caching.
- A custom script/executable may now be substituted for `pygmentize` by redefining `\MintedPygmentize`.

v2.0alpha (2013/07/30)

- Added the package option `cache`. This significantly increases compilation speed by caching old output. For example, compiling the documentation is around 5x faster.
- New inline command `\mintinline`. Custom versions can be created via `\newmintinline`. The command works inside other commands (for example, footnotes) in most situations, so long as the percent and hash characters are avoided.
- The new `\setminted` command allows options to be specified at the document and language levels.
- All extended characters (Unicode, etc.) supported by `inputenc` now work under the `pdfTeX` engine. This involved using `\detokenize` on everything prior to saving.
- New package option `langlinenos` allows line numbering to pick up where it left off for a given language when `firstnumber=last`.
- New options, including `style`, `encoding`, `outencoding`, `codetagify`, `keywordcase`, `texcomments` (same as `texcl`), `python3` (for the `PythonConsoleLexer`), and `numbers`.
- `\usemintedstyle` now takes an optional argument to specify the style for a particular language, and works anywhere in the document.
- `xcolor` is only loaded if `color` isn't, preventing potential package clashes.

1.7 (2011/09/17)

- Options for float placement added [2011/09/12]
- Fixed `tabsize` option [2011/08/30]
- More robust detection of the `-shell-escape` option [2011/01/21]
- Added the `label` option [2011/01/04]
- Installation instructions added [2010/03/16]
- Minimal working example added [2010/03/16]
- Added PHP-specific options [2010/03/14]
- Removed unportable flag from Unix shell command [2010/02/16]

1.6 (2010/01/31)

- Added font-related options [2010/01/27]
- Windows support added [2010/01/27]
- Added command shortcuts [2010/01/22]
- Simpler versioning scheme [2010/01/22]

0.1.5 (2010/01/13)

- Added `fillcolor` option [2010/01/10]
- Added float support [2010/01/10]
- Fixed `firstnumber` option [2010/01/10]
- Removed `caption` option [2010/01/10]

0.0.4 (2010/01/08)

- Initial version [2010/01/08]

9 Implementation

9.1 Required packages

Load required packages. For compatibility reasons, most old functionality should be supported with the original set of packages. More recently added packages, such as `etoolbox` and `xstring`, should only be used for new features when possible.

```
1 \RequirePackage{keyval}
2 \RequirePackage{kvoptions}
3 \RequirePackage{fancyvrb}
4 \RequirePackage{float}
5 \RequirePackage{ifthen}
6 \RequirePackage{calc}
```



```

7 \RequirePackage{ifplatform}
8 \RequirePackage{pdftexcmds}
9 \RequirePackage{etoolbox}
10 \RequirePackage{xstring}
11 \RequirePackage{lineno}

```

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

```

12 \AtBeginDocument{%
13   \@ifpackageloaded{color}{}{%
14     \@ifpackageloaded{xcolor}{}{\RequirePackage{xcolor}}}%
15 }

```

9.2 Package options

`\minted@float@within` Define an option that controls the section numbering of the listing float.

```

16 \DeclareVoidOption{chapter}{\def\minted@float@within{chapter}}
17 \DeclareVoidOption{section}{\def\minted@float@within{section}}

```

`newfloat` Define an option to use `newfloat` rather than `float` to create a floated listing environment.

```

18 \DeclareBoolOption{newfloat}

```

`cache` Define an option that determines whether highlighted content is cached. We use a boolean to keep track of its state.

```

19 \DeclareBoolOption[true]{cache}

```

`\minted@jobname` At various points, temporary files and directories will need to be named after the main `.tex` file. The typical way to do this is to use `\jobname`. However, if the file name contains spaces, then `\jobname` will contain the name wrapped in quotes (older versions of MiKTeX replace spaces with asterisks instead, and XeTeX apparently [allows double quotes within file names](#), in which case names are wrapped in single quotes). While that is perfectly fine for working with L^AT_EX internally, it causes problems with `\write18`, since quotes will end up in unwanted locations in shell commands. It would be possible to strip the wrapping quotation marks when they are present, and maintain any spaces in the file name. But it is simplest to create a “sanitized” version of `\jobname` in which spaces and asterisks are replaced by underscores, and double quotes are stripped.

```

20 \StrSubstitute{\jobname}{ }{ }[_]{\minted@jobname}
21 \StrSubstitute{\minted@jobname}{*}{ }[_]{\minted@jobname}
22 \StrSubstitute{\minted@jobname}{"}{ }[_]{\minted@jobname}

```

`\minted@cachedir` Set the directory in which cached content is saved. The default uses a `minted-` prefix followed by the sanitized `\minted@jobname`.

```

23 \newcommand{\minted@cachedir}{\detokenize{_\minted-\minted@jobname}
24 \let\minted@cachedir@windows\minted@cachedir
25 \define@key{minted}{cachedir}{%
26   \@namedef{minted@cachedir}{#1}%
27   \StrSubstitute{\minted@cachedir}{/}{\@backslashchar}[\minted@cachedir@windows]}

```

`\minted@outputdir` The `-output-directory` command-line option for L^AT_EX causes problems for `minted`, because the `minted` temporary files are saved in the output directory, but `minted` still looks for them in the document root directory. There is no way to access the value of the command-line option. But it is possible to allow the output directory to be specified manually as a package option. A trailing slash is automatically appended to the `outputdir`, so that it may be directly joined to `cachedir`. This may be redundant if the user-supplied value already ends with a slash, but doubled slashes are ignored under *nix and Windows, so it isn't a problem.

```

28 \let\minted@outputdir\@empty
29 \let\minted@outputdir@windows\@empty
30 \define@key{minted}{outputdir}{%
31   \@namedef{minted@outputdir}{#1/}%
32   \StrSubstitute{\minted@outputdir}{/}{\@backslashchar}{/}%
33   {\@backslashchar}[\minted@outputdir@windows]}

```

`kpsewhich` Define an option that invokes `kpsewhich` to locate the files that are to be `pygmentized`. This isn't done by default to avoid the extra overhead, but can be useful with some build tools such as `texi2pdf` that rely on modifying `TEXINPUTS`.

```

34 \DeclareBoolOption{kpsewhich}

```

`langlinenos` Define an option that makes all `minted` environments and `\mint` commands for a given language share cumulative line numbering (if `firstnumber=last`).

```

35 \DeclareBoolOption{langlinenos}

```

`draft` Define an option that allows `fancyvrb` to do all typesetting directly, without using `Pygments`. This trades syntax highlighting for speed. Note that in many cases, the difference in performance between caching and draft mode will be minimal. Also note that draft settings may be inherited from the document class.

```

36 \DeclareBoolOption{draft}

```

`final` Define a `final` option that is the opposite of `draft`, since many packages do this.

```

37 \DeclareComplementaryOption{final}{draft}

```

Process package options. Proceed with everything that immediately relies upon them. If PGF/TikZ externalization is in use, switch on `draft` mode and turn off `cache`. Externalization involves compiling the *entire* document; all parts not related to the current image are “silently thrown away.” `minted` needs to cooperate with that by not writing any temp files or creating any directories.

```

38 \ProcessKeyvalOptions*
39 \ifthenelse{\boolean{minted@newfloat}}{\RequirePackage{newfloat}}{}
40 \ifcsname tikzexternalrealjob\endcsname
41   \minted@drafttrue
42   \minted@cachefalse
43 \else
44 \fi
45 \ifthenelse{\boolean{minted@cache}}{%
46   \AtEndOfPackage{\ProvideDirectory{\minted@outputdir\minted@cachedir}}{}}

```

9.3 Input, caching, and temp files

`\minted@input` We need a wrapper for `\input`. In most cases, `\input` failure will be due to attempts to use `\inputminted` with files that don’t exist, but we also want to give informative error messages when `outputdir` is needed or incompatible build tools are used.

```

47 \newcommand{\minted@input}[1]{%
48   \IfFileExists{#1}%
49   {\input{#1}}%
50   {\PackageError{minted}{Missing Pygments output; \string\inputminted\space
51     was^^Jprobably given a file that does not exist--otherwise, you may need
52     ^^Jthe outputdir package option, or may be using an incompatible build
53     tool\ifwindows,^^Jor may be using the kpsewhich option without having
54     PowerShell installed\fi}%
55   {This could be caused by using -output-directory or -aux-directory
56     ^^Jwithout setting minted’s outputdir, or by using a build tool that
57     ^^Jchanges paths in ways minted cannot detect\ifwindows, or by using the
58     ^^Jkpsewhich option without PowerShell\fi.}}%
59 }

```

`\minted@infile` Define a default name for files of highlighted content that are brought in. Caching will redefine this. We start out with the default, non-caching value.

```

60 \newcommand{\minted@infile}{\minted@jobname.out.pyg}

```

We need a way to track the cache files that are created, and delete those that are not in use. This is accomplished by creating a comma-delimited list of cache files and saving this list to the `.aux` file so that it may be accessed on subsequent runs. During subsequent runs, this list is compared against the cache files that are actually used, and unused files are deleted. Cache file names are created with

MD5 hashes of highlighting settings and file contents, with a `.pygtex` extension, so they never contain commas. Thus comma-delimiting the list of file names doesn't introduce a potential for errors.

`\minted@cachelist` This is a list of the current cache files.

```
61 \newcommand{\minted@cachelist}{}
```

`\minted@addcachefile` This adds a file to the list of cache files. It also creates a macro involving the hash, so that the current usage of the hash can be easily checked by seeing if the macro exists. The list of cache files must be created with built-in linebreaks, so that when it is written to the `.aux` file, it won't all be on one line and thereby risk buffer errors.

```
62 \newcommand{\minted@addcachefile}[1]{%
63   \expandafter\long\expandafter\gdef\expandafter\minted@cachelist\expandafter{%
64     \minted@cachelist,^^J%
65     \space\space#1}%
66   \expandafter\gdef\csname minted@cached@#1\endcsname{}%
67 }
```

`\minted@savecachelist` We need to be able to save the list of cache files to the `.aux` file, so that we can reload it on the next run.

```
68 \newcommand{\minted@savecachelist}{%
69   \ifdefempty{\minted@cachelist}{}{}%
70   \immediate\write\@mainaux{%
71     \string\gdef\string\minted@oldcachelist\string{%
72       \minted@cachelist\string}}%
73   }%
74 }
```

`\minted@cleancache` Clean up old cache files that are no longer in use.

```
75 \newcommand{\minted@cleancache}{%
76   \ifcsname minted@oldcachelist\endcsname
77     \def\do##1{%
78       \ifthenelse{\equal{##1}{}}{}{}%
79       \ifcsname minted@cached@##1\endcsname\else
80         \DeleteFile[\minted@outputdir\minted@cachedir]{##1}%
81       \fi
82     }%
83   }%
84   \expandafter\docsvlist\expandafter{\minted@oldcachelist}%
85 \else
86 \fi
87 }
```

At the end of the document, save the list of cache files and clean the cache. If in draft mode, don't clean up the cache and save the old cache file list for next time. This allows draft mode to be switched on and off without requiring that all highlighted content be regenerated. The saving and cleaning operations may be called without conditionals, since their definitions already contain all necessary checks for their correct operation.

```

88 \ifthenelse{\boolean{minted@draft}}{%
89   {\AtEndDocument{%
90     \ifcsname minted@oldcachelist\endcsname
91     \let\minted@cachelist\minted@oldcachelist
92     \minted@savecachelist
93     \fi}}%
94   {\AtEndDocument{%
95     \minted@savecachelist
96     \minted@cleancache}}%

```

9.4 OS interaction

We need system-dependent macros for communicating with the “outside world.”

`\DeleteFile` Delete a file. Define conditionally in case an equivalent macro has already been defined.

```

97 \ifwindows
98   \providecommand{\DeleteFile}[2][{}]{%
99     \ifthenelse{\equal{#1}{} }{%
100       {\IfFileExists{#2}{\immediate\write18{del "#2"}}{}}%
101       {\IfFileExists{#1/#2}{%
102         \StrSubstitute{#1}{/}{\@backslashchar}[\minted@windir]
103         \immediate\write18{del "\minted@windir\@backslashchar #2"}}{}}
104     \else
105       \providecommand{\DeleteFile}[2][{}]{%
106         \ifthenelse{\equal{#1}{} }{%
107           {\IfFileExists{#2}{\immediate\write18{rm "#2"}}{}}%
108           {\IfFileExists{#1/#2}{\immediate\write18{rm "#1/#2"}}{}}
109         \fi

```

`\ProvideDirectory` We need to be able to create a directory, if it doesn't already exist. This is primarily for storing cached highlighted content.

```

110 \ifwindows
111   \newcommand{\ProvideDirectory}[1]{%
112     \StrSubstitute{#1}{/}{\@backslashchar}[\minted@windir]
113     \immediate\write18{if not exist "\minted@windir" mkdir "\minted@windir"}}
114   \else
115     \newcommand{\ProvideDirectory}[1]{%

```

```

116     \immediate\write18{mkdir -p "#1"}
117 \fi

```

`\TestAppExists` Determine whether a given application exists.

Usage is a bit roundabout, but has been retained for backward compatibility. At some point, it may be worth replacing this with something using `\@input"<command>"`. That would require MiKTeX users to `--enable-pipes`, however, which would make things a little more complicated. If Windows XP compatibility is ever no longer required, the `where` command could be used instead of the approach for Windows.

To test whether an application exists, use the following code:

```

\TestAppExists{appname}
\ifthenelse{\boolean{AppExists}}{app exists}{app doesn't exist}

118 \newboolean{AppExists}
119 \newread\minted@appexistsfile
120 \newcommand{\TestAppExists}[1]{
121     \ifwindows

```

On Windows, we need to use path expansion and write the result to a file. If the application doesn't exist, the file will be empty (except for a newline); otherwise, it will contain the full path of the application.

```

122     \DeleteFile{\minted@jobname.aex}
123     \immediate\write18{for \string^\@percentchar i in (#1.exe #1.bat #1.cmd)
124         do set >"\minted@jobname.aex" <nul: /p
125         x=\string^\@percentchar \string~$PATH:i>>"\minted@jobname.aex"}
126     %$ <- balance syntax highlighting
127     \immediate\openin\minted@appexistsfile\minted@jobname.aex
128     \expandafter\def\expandafter\@tmp@cr\expandafter{\the\endlinechar}
129     \endlinechar=-1\relax
130     \readline\minted@appexistsfile to \minted@appathifexists
131     \endlinechar=\@tmp@cr
132     \ifthenelse{\equal{\minted@appathifexists}{}}
133     {\AppExistsfalse}
134     {\AppExiststrue}
135     \immediate\closein\minted@appexistsfile
136     \DeleteFile{\minted@jobname.aex}
137 \else

```

On Unix-like systems, we do a straightforward which test and create a file upon success, whose existence we can then check.

```

138     \immediate\write18{which "#1" && touch "\minted@jobname.aex"}
139     \IfFileExists{\minted@jobname.aex}
140     {\AppExiststrue}

```

```

141     \DeleteFile{\minted@jobname.aex}}
142     {\AppExistsfalse}
143   \fi
144 }

```

9.5 Option processing

Option processing is somewhat involved, because we want to be able to define options at various levels of hierarchy: individual command/environment, language, global (document). And once those options are defined, we need to go through the hierarchy in a defined order of precedence to determine which option to apply. As if that wasn't complicated enough, some options need to be sent to Pygments, some need to be sent to `fancyvrb`, and some need to be processed within `minted` itself.

To begin with, we need macros for storing lists of options that will later be passed via the command line to Pygments (`optlistcl`). These are defined at the global (`cl@g`), language (`cl@lang`), and command or environment (`cl@cmd`) levels, so that settings can be specified at various levels of hierarchy. The language macro is actually a placeholder. The current language will be tracked using `\minted@lang`. Each individual language will create a `\minted@optlistcl@lang<language>` macro. `\minted@optlistcl@lang` may be `\let` to this macro as convenient; otherwise, the general language macro merely serves as a placeholder.

The global- and language-level lists also have an `inline (i)` variant. This allows different settings to be applied in inline settings. An inline variant is not needed at the command/environment level, since at that level settings would not be present unless they were supposed to be applied.

```

\minted@optlistcl@g
145 \newcommand{\minted@optlistcl@g}{}

\minted@optlistcl@g@i
146 \newcommand{\minted@optlistcl@g@i}{}

\minted@lang
147 \let\minted@lang\@empty

\minted@optlistcl@lang
148 \newcommand{\minted@optlistcl@lang}{}

\minted@optlistcl@lang@i
149 \newcommand{\minted@optlistcl@lang@i}{}

```

```
\minted@optlistcl@cmd
```

```
150 \newcommand{\minted@optlistcl@cmd}{}
```

We also need macros for storing lists of options that will later be passed to `fancyvrb` (`optlistfv`). As before, these exist at the global (`fv@g`), language (`fv@lang`), and command or environment (`fv@cmd`) levels. `Pygments` accepts `fancyvrb` options, but in almost all cases, these options may be applied via `\fvset` rather than via running `Pygments`. This is significantly more efficient when caching is turned on, since it allows formatting changes to be applied without having to re-highlight the code.

```
\minted@optlistfv@g
```

```
151 \newcommand{\minted@optlistfv@g}{}
```

```
\minted@optlistfv@g@i
```

```
152 \newcommand{\minted@optlistfv@g@i}{}
```

```
\minted@optlistfv@lang
```

```
153 \newcommand{\minted@optlistfv@lang}{}
```

```
\minted@optlistfv@lang@i
```

```
154 \newcommand{\minted@optlistfv@lang@i}{}
```

```
\minted@optlistfv@cmd
```

```
155 \newcommand{\minted@optlistfv@cmd}{}
```

```
\minted@configlang
```

We need a way to check whether a language has had all its option list macros created. This generally occurs in a context where `\minted@lang` needs to be set. So we create a macro that does both at once. If the language list macros do not exist, we create them globally to simplify future operations.

```
156 \newcommand{\minted@configlang}[1]{%
157   \def\minted@lang{#1}%
158   \ifcsname minted@optlistcl@lang\minted@lang\endcsname\else
159     \expandafter\gdef\csname minted@optlistcl@lang\minted@lang\endcsname{}%
160   \fi
161   \ifcsname minted@optlistcl@lang\minted@lang @i\endcsname\else
162     \expandafter\gdef\csname minted@optlistcl@lang\minted@lang @i\endcsname{}%
163   \fi
164   \ifcsname minted@optlistfv@lang\minted@lang\endcsname\else
165     \expandafter\gdef\csname minted@optlistfv@lang\minted@lang\endcsname{}%
```



```

166 \fi
167 \ifcsname minted@optlistfv@lang\minted@lang @i\endcsname\else
168 \expandafter\gdef\csname minted@optlistfv@lang\minted@lang @i\endcsname{%
169 \fi
170 }

```

We need a way to define options in bulk at the global, language, and command levels. How this is done will depend on the type of option. The keys created are grouped by level: `minted@opt@g`, `minted@opt@lang`, and `minted@opt@cmd`, plus inline variants. The language-level key groupings use `\minted@lang` internally, so we don't need to duplicate the internals for different languages. The key groupings are independent of whether a given option relates to `Pygments`, `fancyvrb`, etc. Organization by level is the only thing that is important here, since keys are applied in a hierarchical fashion. Key values are stored in macros of the form `\minted@opt@<level>:<key>`, so that they may be retrieved later. In practice, these key macros will generally not be used directly (hence the colon in the name). Rather, the hierarchy of macros will be traversed until an existing macro is found.

`\minted@def@optcl` Define a generic option that will be passed to the command line. Options are given in a `{key}{value}` format that is transformed into `key=value` and then passed to `pygmentize`. This allows `value` to be easily stored in a separate macro for later access. This is useful, for example, in separately accessing the value of `encoding` for performing `autogobble`.

If a key option is specified without `=value`, the default is assumed. Options are automatically created at all levels.

Options are added to the option lists in such a way that they will be detokenized. This is necessary since they will ultimately be used in `\write18`.

```

171 \newcommand{\minted@addto@optlistcl}[2]{%
172 \expandafter\def\expandafter#1\expandafter{#1%
173 \detokenize{#2}\space}}
174 \newcommand{\minted@addto@optlistcl@lang}[2]{%
175 \expandafter\let\expandafter\minted@tmp\csname #1\endcsname
176 \expandafter\def\expandafter\minted@tmp\expandafter{\minted@tmp%
177 \detokenize{#2}\space}%
178 \expandafter\let\csname #1\endcsname\minted@tmp}
179 \newcommand{\minted@def@optcl}[4][[]]{%
180 \ifthenelse{\equal{#1}{}}{%
181 \define@key{minted@opt@g}{#2}{%
182 \minted@addto@optlistcl{\minted@optlistcl@g}{#3=#4}%
183 \@namedef{minted@opt@g:#2}{#4}}%
184 \define@key{minted@opt@g@i}{#2}{%
185 \minted@addto@optlistcl{\minted@optlistcl@g@i}{#3=#4}%
186 \@namedef{minted@opt@g@i:#2}{#4}}%
187 \define@key{minted@opt@lang}{#2}{%
188 \minted@addto@optlistcl@lang{\minted@optlistcl@lang\minted@lang}{#3=#4}%
189 \@namedef{minted@opt@lang\minted@lang:#2}{#4}}%

```

```

190     \define@key{minted@opt@lang@i}{#2}{%
191       \minted@addto@optlistcl@lang{%
192         minted@optlistcl@lang\minted@lang @i}{#3=#4}%
193       \@namedef{minted@opt@lang\minted@lang @i:#2}{#4}}%
194     \define@key{minted@opt@cmd}{#2}{%
195       \minted@addto@optlistcl{\minted@optlistcl@cmd}{#3=#4}%
196       \@namedef{minted@opt@cmd:#2}{#4}}%
197   {\define@key{minted@opt@g}{#2}[#1]{%
198     \minted@addto@optlistcl{\minted@optlistcl@g}{#3=#4}%
199     \@namedef{minted@opt@g:#2}{#4}}%
200   \define@key{minted@opt@g@i}{#2}[#1]{%
201     \minted@addto@optlistcl{\minted@optlistcl@g@i}{#3=#4}%
202     \@namedef{minted@opt@g@i:#2}{#4}}%
203   \define@key{minted@opt@lang}{#2}[#1]{%
204     \minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang}{#3=#4}%
205     \@namedef{minted@opt@lang\minted@lang:#2}{#4}}%
206   \define@key{minted@opt@lang@i}{#2}[#1]{%
207     \minted@addto@optlistcl@lang{%
208       minted@optlistcl@lang\minted@lang @i}{#3=#4}%
209     \@namedef{minted@opt@lang\minted@lang @i:#2}{#4}}%
210   \define@key{minted@opt@cmd}{#2}[#1]{%
211     \minted@addto@optlistcl{\minted@optlistcl@cmd}{#3=#4}%
212     \@namedef{minted@opt@cmd:#2}{#4}}%
213 }

```

This covers the typical options that must be passed to Pygments. But some, particularly `escapeinside`, need more work. Since their arguments may contain escaped characters, expansion rather than detokenization is needed. Getting expansion to work as desired in a `\write18` context requires the redefinition of some characters

`\minted@escchars` We need to define versions of common escaped characters that will work correctly under expansion for use in `\write18`.

```

214 \edef\minted@hashchar{\string#}
215 \edef\minted@dollarchar{\string$}
216 \edef\minted@ampchar{\string&}
217 \edef\minted@underscorechar{\string_}
218 \edef\minted@tildechar{\string~}
219 \edef\minted@leftsquarebracket{\string[}
220 \edef\minted@rightsquarebracket{\string]}
221 \newcommand{\minted@escchars}{%
222   \let\#\minted@hashchar
223   \let\%\minted@percentchar
224   \let\{\minted@charlb
225   \let\}\minted@charrb
226   \let\$\minted@dollarchar
227   \let\&\minted@ampchar
228   \let\_ \minted@underscorechar

```

```

229 \let\\@backslashchar
230 \let~\minted@tildechar
231 \let\~\minted@tildechar
232 \let\[\minted@leftsquarebracket
233 \let\]\minted@rightsquarebracket
234 } %$ <- highlighting

```

\minted@def@optcl@e Now to define options that are expanded.

```

235 \newcommand{\minted@addto@optlistcl@e}[2]{%
236 \begingroup
237 \minted@escchars
238 \xdef\minted@xtmp{#2}%
239 \endgroup
240 \expandafter\minted@addto@optlistcl@e@i\expandafter{\minted@xtmp}{#1}}
241 \def\minted@addto@optlistcl@e@i#1#2{%
242 \expandafter\def\expandafter#2\expandafter{#2#1\space}}
243 \newcommand{\minted@addto@optlistcl@lang@e}[2]{%
244 \begingroup
245 \minted@escchars
246 \xdef\minted@xtmp{#2}%
247 \endgroup
248 \expandafter\minted@addto@optlistcl@lang@e@i\expandafter{\minted@xtmp}{#1}}
249 \def\minted@addto@optlistcl@lang@e@i#1#2{%
250 \expandafter\let\expandafter\minted@tmp\csname #2\endcsname
251 \expandafter\def\expandafter\minted@tmp\expandafter{\minted@tmp#1\space}%
252 \expandafter\let\csname #2\endcsname\minted@tmp}
253 \newcommand{\minted@def@optcl@e}[4][[]]{%
254 \ifthenelse{\equal{#1}{}}{%
255   {\define@key{minted@opt@g}{#2}{%
256     \minted@addto@optlistcl@e{\minted@optlistcl@g}{#3=#4}%
257     \@namedef{minted@opt@g:#2}{#4}}}%
258   \define@key{minted@opt@g@i}{#2}{%
259     \minted@addto@optlistcl@e{\minted@optlistcl@g@i}{#3=#4}%
260     \@namedef{minted@opt@g@i:#2}{#4}}}%
261   \define@key{minted@opt@lang}{#2}{%
262     \minted@addto@optlistcl@lang@e{minted@optlistcl@lang\minted@lang}{#3=#4}%
263     \@namedef{minted@opt@lang\minted@lang:#2}{#4}}}%
264   \define@key{minted@opt@lang@i}{#2}{%
265     \minted@addto@optlistcl@lang@e{%
266       minted@optlistcl@lang\minted@lang @i}{#3=#4}%
267     \@namedef{minted@opt@lang\minted@lang @i:#2}{#4}}}%
268   \define@key{minted@opt@cmd}{#2}{%
269     \minted@addto@optlistcl@e{\minted@optlistcl@cmd}{#3=#4}%
270     \@namedef{minted@opt@cmd:#2}{#4}}}%
271   {\define@key{minted@opt@g}{#2}[#1]{%
272     \minted@addto@optlistcl@e{\minted@optlistcl@g}{#3=#4}%
273     \@namedef{minted@opt@g:#2}{#4}}}%
274   \define@key{minted@opt@g@i}{#2}[#1]{%
275     \minted@addto@optlistcl@e{\minted@optlistcl@g@i}{#3=#4}%

```

```

276     \@namedef{minted@opt@g@i:#2}{#4}}%
277 \define@key{minted@opt@lang}{#2}[#1]{%
278   \minted@addto@optlistcl@lang@e{minted@optlistcl@lang\minted@lang}{#3=#4}%
279   \@namedef{minted@opt@lang\minted@lang:#2}{#4}}%
280 \define@key{minted@opt@lang@i}{#2}[#1]{%
281   \minted@addto@optlistcl@lang@e{%
282     minted@optlistcl@lang\minted@lang @i}{#3=#4}%
283   \@namedef{minted@opt@lang\minted@lang @i:#2}{#4}}%
284 \define@key{minted@opt@cmd}{#2}[#1]{%
285   \minted@addto@optlistcl@e{\minted@optlistcl@cmd}{#3=#4}%
286   \@namedef{minted@opt@cmd:#2}{#4}}}%
287 }

```

`\minted@def@optcl@switch` Define a switch or boolean option that is passed to Pygments, which is true when no value is specified.

```

288 \newcommand{\minted@def@optcl@switch}[2]{%
289   \define@booleankey{minted@opt@g}{#1}%
290   {\minted@addto@optlistcl{\minted@optlistcl@g}{#2=True}}%
291   \@namedef{minted@opt@g:#1}{true}}
292   {\minted@addto@optlistcl{\minted@optlistcl@g}{#2=False}}%
293   \@namedef{minted@opt@g:#1}{false}}
294 \define@booleankey{minted@opt@g@i}{#1}%
295   {\minted@addto@optlistcl{\minted@optlistcl@g@i}{#2=True}}%
296   \@namedef{minted@opt@g@i:#1}{true}}
297   {\minted@addto@optlistcl{\minted@optlistcl@g@i}{#2=False}}%
298   \@namedef{minted@opt@g@i:#1}{false}}
299 \define@booleankey{minted@opt@lang}{#1}%
300   {\minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang}{#2=True}}%
301   \@namedef{minted@opt@lang\minted@lang:#1}{true}}
302   {\minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang}{#2=False}}%
303   \@namedef{minted@opt@lang\minted@lang:#1}{false}}
304 \define@booleankey{minted@opt@lang@i}{#1}%
305   {\minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang @i}{#2=True}}%
306   \@namedef{minted@opt@lang\minted@lang @i:#1}{true}}
307   {\minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang @i}{#2=False}}%
308   \@namedef{minted@opt@lang\minted@lang @i:#1}{false}}
309 \define@booleankey{minted@opt@cmd}{#1}%
310   {\minted@addto@optlistcl{\minted@optlistcl@cmd}{#2=True}}%
311   \@namedef{minted@opt@cmd:#1}{true}}
312   {\minted@addto@optlistcl{\minted@optlistcl@cmd}{#2=False}}%
313   \@namedef{minted@opt@cmd:#1}{false}}
314 }

```

Now that all the machinery for Pygments options is in place, we can move on to fancyvrb options.

`\minted@def@optfv` Define fancyvrb options.

```

315 \newcommand{\minted@def@optfv}[1]{%
316   \define@key{minted@opt@g}{#1}{%
317     \expandafter\def\expandafter\minted@optlistfv@g\expandafter{%
318       \minted@optlistfv@g#1=##1,}%
319     \@namedef{minted@opt@g:#1}{##1}}
320 \define@key{minted@opt@g@i}{#1}{%
321   \expandafter\def\expandafter\minted@optlistfv@g@i\expandafter{%
322     \minted@optlistfv@g@i#1=##1,}%
323   \@namedef{minted@opt@g@i:#1}{##1}}
324 \define@key{minted@opt@lang}{#1}{%
325   \expandafter\let\expandafter\minted@tmp%
326     \csname minted@optlistfv@lang\minted@lang\endcsname
327   \expandafter\def\expandafter\minted@tmp\expandafter{%
328     \minted@tmp#1=##1,}%
329   \expandafter\let\csname minted@optlistfv@lang\minted@lang\endcsname%
330     \minted@tmp
331   \@namedef{minted@opt@lang\minted@lang:#1}{##1}}
332 \define@key{minted@opt@lang@i}{#1}{%
333   \expandafter\let\expandafter\minted@tmp%
334     \csname minted@optlistfv@lang\minted@lang @i\endcsname
335   \expandafter\def\expandafter\minted@tmp\expandafter{%
336     \minted@tmp#1=##1,}%
337   \expandafter\let\csname minted@optlistfv@lang\minted@lang @i\endcsname%
338     \minted@tmp
339   \@namedef{minted@opt@lang\minted@lang @i:#1}{##1}}
340 \define@key{minted@opt@cmd}{#1}{%
341   \expandafter\def\expandafter\minted@optlistfv@cmd\expandafter{%
342     \minted@optlistfv@cmd#1=##1,}%
343   \@namedef{minted@opt@cmd:#1}{##1}}
344 }

```

`\minted@def@optfv@switch` Define fancyvrb boolean options.

```

345 \newcommand{\minted@def@optfv@switch}[1]{%
346   \define@booleankey{minted@opt@g}{#1}{%
347     {\expandafter\def\expandafter\minted@optlistfv@g\expandafter{%
348       \minted@optlistfv@g#1=true,}%
349     \@namedef{minted@opt@g:#1}{true}}%
350   {\expandafter\def\expandafter\minted@optlistfv@g\expandafter{%
351     \minted@optlistfv@g#1=false,}%
352   \@namedef{minted@opt@g:#1}{false}}%
353 \define@booleankey{minted@opt@g@i}{#1}{%
354   {\expandafter\def\expandafter\minted@optlistfv@g@i\expandafter{%
355     \minted@optlistfv@g@i#1=true,}%
356   \@namedef{minted@opt@g@i:#1}{true}}%
357   {\expandafter\def\expandafter\minted@optlistfv@g@i\expandafter{%
358     \minted@optlistfv@g@i#1=false,}%
359   \@namedef{minted@opt@g@i:#1}{false}}%
360 \define@booleankey{minted@opt@lang}{#1}{%
361   {\expandafter\let\expandafter\minted@tmp%

```

```

362     \csname minted@optlistfv@lang\minted@lang\endcsname
363     \expandafter\def\expandafter\minted@tmp\expandafter{%
364         \minted@tmp#1=true,}%
365     \expandafter\let\csname minted@optlistfv@lang\minted@lang\endcsname%
366         \minted@tmp
367     \@namedef{minted@opt@lang\minted@lang:#1}{true}}%
368     {\expandafter\let\expandafter\minted@tmp%
369         \csname minted@optlistfv@lang\minted@lang\endcsname
370     \expandafter\def\expandafter\minted@tmp\expandafter{%
371         \minted@tmp#1=false,}%
372     \expandafter\let\csname minted@optlistfv@lang\minted@lang\endcsname%
373         \minted@tmp
374     \@namedef{minted@opt@lang\minted@lang:#1}{false}}%
375     \define@booleankey{minted@opt@lang@i}{#1}%
376     {\expandafter\let\expandafter\minted@tmp%
377         \csname minted@optlistfv@lang\minted@lang @i\endcsname
378     \expandafter\def\expandafter\minted@tmp\expandafter{%
379         \minted@tmp#1=true,}%
380     \expandafter\let\csname minted@optlistfv@lang\minted@lang @i\endcsname%
381         \minted@tmp
382     \@namedef{minted@opt@lang\minted@lang @i:#1}{true}}%
383     {\expandafter\let\expandafter\minted@tmp%
384         \csname minted@optlistfv@lang\minted@lang @i\endcsname
385     \expandafter\def\expandafter\minted@tmp\expandafter{%
386         \minted@tmp#1=false,}%
387     \expandafter\let\csname minted@optlistfv@lang\minted@lang @i\endcsname%
388         \minted@tmp
389     \@namedef{minted@opt@lang\minted@lang @i:#1}{false}}%
390     \define@booleankey{minted@opt@cmd}{#1}%
391     {\expandafter\def\expandafter\minted@optlistfv@cmd\expandafter{%
392         \minted@optlistfv@cmd#1=true,}%
393     \@namedef{minted@opt@cmd:#1}{true}}%
394     {\expandafter\def\expandafter\minted@optlistfv@cmd\expandafter{%
395         \minted@optlistfv@cmd#1=false,}%
396     \@namedef{minted@opt@cmd:#1}{false}}%
397 }

```

`minted@isinline` In resolving value precedence when actually using values, we need a way to determine whether we are in an inline context. This is accomplished via a boolean that is set at the beginning of inline commands.

```
398 \newboolean{minted@isinline}
```

`\minted@fvset` We will need a way to actually use the lists of stored `fancyvrb` options later on.

```

399 \newcommand{\minted@fvset}{%
400     \expandafter\fvset\expandafter{\minted@optlistfv@g}%
401     \expandafter\let\expandafter\minted@tmp%
402     \csname minted@optlistfv@lang\minted@lang\endcsname

```

```

403 \expandafter\fvset\expandafter{\minted@tmp}%
404 \ifthenelse{\boolean{minted@isinline}}%
405   {\expandafter\fvset\expandafter{\minted@optlistfv@g@i}%
406    \expandafter\let\expandafter\minted@tmp%
407     \csname minted@optlistfv@lang\minted@lang @i\endcsname
408    \expandafter\fvset\expandafter{\minted@tmp}}%
409   {}%
410 \expandafter\fvset\expandafter{\minted@optlistfv@cmd}%
411 }

```

We need a way to define `minted`-specific options at multiple levels of hierarchy, as well as a way to retrieve these options. As with previous types of options, values are stored in macros of the form `\minted@opt@<level>:<key>`, since they are not meant to be accessed directly.

The order of precedence is `cmd`, `lang@i`, `g@i`, `lang`, `g`. A value specified at the command or environment level should override other settings. In its absence, a value specified for an inline command should override other settings, if we are indeed in an inline context. Otherwise, language settings take precedence over global settings.

Before actually creating the option-definition macro, we need a few helper macros.

`\minted@def@opt` Finally, on to the actual option definitions for `minted`-specific options.

Usage: `\minted@def@opt [<initial global value>] {<key name>}`

```

412 \newcommand{\minted@def@opt}[2][{}]{%
413   \define@key{minted@opt@g}{#2}{%
414     \@namedef{minted@opt@g:#2}{##1}}
415   \define@key{minted@opt@g@i}{#2}{%
416     \@namedef{minted@opt@g@i:#2}{##1}}
417   \define@key{minted@opt@lang}{#2}{%
418     \@namedef{minted@opt@lang\minted@lang:#2}{##1}}
419   \define@key{minted@opt@lang@i}{#2}{%
420     \@namedef{minted@opt@lang\minted@lang @i:#2}{##1}}
421   \define@key{minted@opt@cmd}{#2}{%
422     \@namedef{minted@opt@cmd:#2}{##1}}
423 }

```

`\minted@def@opt@style` Define an option for styles. These are defined independently because styles need slightly different handling. It is convenient to create style macros when styles are set. Otherwise, it would be necessary to check for the existence of style macros at the beginning of every command or environment.

```

424 \newcommand{\minted@def@opt@style}{%
425   \define@key{minted@opt@g}{style}{%
426     \minted@checkstyle{##1}%
427     \@namedef{minted@opt@g:style}{##1}}%

```

```

428 \define@key{minted@opt@g@i}{style}{%
429   \minted@checkstyle{##1}%
430   \@namedef{minted@opt@g@i:style}{##1}}%
431 \define@key{minted@opt@lang}{style}{%
432   \minted@checkstyle{##1}%
433   \@namedef{minted@opt@lang\minted@lang:style}{##1}}%
434 \define@key{minted@opt@lang@i}{style}{%
435   \minted@checkstyle{##1}%
436   \@namedef{minted@opt@lang\minted@lang @i:style}{##1}}%
437 \define@key{minted@opt@cmd}{style}{%
438   \minted@checkstyle{##1}%
439   \@namedef{minted@opt@cmd:style}{##1}}%
440 }

```

`\minted@checkstyle` Make sure that style macros exist.

We have to do some tricks with `\endlinechar` to prevent `\input` from inserting unwanted whitespace. That is primarily for inline commands, where it would introduce a line break. There is also the very unorthodox `\let\def\gdef` to make sure that macros are defined globally.

If a style is not given, then revert to the default style, but create macros with prefix `PYG`, and create `default-pyg-prefix.pygstyle` if caching is on. This allows a graceful fallback in the event that style is empty. It is also purposefully used to create a complete set of macros with prefix `PYG`, so that the symbol macros may be used, as described next.

The typical style macros created by `\minted@checkstyle`, which are of the form `\PYG<style>`, are used indirectly. All code is highlighted with `commandprefix=PYG`, so that it uses `\PYG`. Then `\PYG` is `\let` to `\PYG<style>` as appropriate. This way, code need not be highlighted again when the style is changed. This has the disadvantage that none of the `\PYG<symbol>` macros will be defined; rather, only `\PYG<style><symbol>` macros will be defined. It would be possible to `\let \PYG<symbol>` to `\PYG<style><symbol>`, but it is simpler to define a complete set of symbol macros using the `PYG` prefix, so that all symbol macros will be defined by default.⁵

```

441 \newcommand{\minted@checkstyle}[1]{%
442   \ifcsname minted@styleloaded@\ifstrempy{#1}{default-pyg-prefix}{#1}\endcsname\el
443   \expandafter\gdef%
444   \csname minted@styleloaded@\ifstrempy{#1}{default-pyg-prefix}{#1}\endcsname{
445   \ifthenelse{\boolean{minted@cache}}%
446     {\IfFileExists
447       {\minted@outputdir\minted@cachedir/\ifstrempy{#1}{default-pyg-prefix}{#1}.p

```

⁵It would be possible to hard-code the symbol macros in `minted` itself, but that would have the disadvantage of tying `minted` more closely to a particular version of `Pygments`. Similarly, `\let`ing symbol macros assumes a complete, fixed list of symbol macros. The current approach is harder to break than these alternatives; the worst-case scenario should be needing to purge the cache, rather than dealing with an undefined macro.


```

448 {}%
449 {%
450 \ifwindows
451 \immediate\writel8{%
452 \MintedPygmentize\space -S \ifstrempy{#1}{default}{#1} -f latex
453 -P commandprefix=PYG#1
454 > "\minted@outputdir@windows\minted@cachedir@windows\@backslashchar%
455 \ifstrempy{#1}{default-pyg-prefix}{#1}.pygstyle"%
456 \else
457 \immediate\writel8{%
458 \MintedPygmentize\space -S \ifstrempy{#1}{default}{#1} -f latex
459 -P commandprefix=PYG#1
460 > "\minted@outputdir\minted@cachedir/%
461 \ifstrempy{#1}{default-pyg-prefix}{#1}.pygstyle"%
462 \fi
463 }%
464 \begingroup
465 \let\def\gdef
466 \endlinechar=-1\relax
467 \minted@input{%
468 \minted@outputdir\minted@cachedir/\ifstrempy{#1}{default-pyg-prefix}{#1}
469 \endgroup
470 \minted@addcachefile{\ifstrempy{#1}{default-pyg-prefix}{#1}.pygstyle}}%
471 {\ifwindows
472 \immediate\writel8{%
473 \MintedPygmentize\space -S \ifstrempy{#1}{default}{#1} -f latex
474 -P commandprefix=PYG#1 > "\minted@outputdir@windows\minted@jobname.out.
475 \else
476 \immediate\writel8{%
477 \MintedPygmentize\space -S \ifstrempy{#1}{default}{#1} -f latex
478 -P commandprefix=PYG#1 > "\minted@outputdir\minted@jobname.out.pyg"%
479 \fi
480 \begingroup
481 \let\def\gdef
482 \endlinechar=-1\relax
483 \minted@input{\minted@outputdir\minted@jobname.out.pyg}%
484 \endgroup}%
485 \fi
486 }
487 \ifthenelse{\boolean{minted@draft}}{\renewcommand{\minted@checkstyle}[1]{}{}}

```

At the beginning of the document, create the symbol macros with PYG prefix. This must wait until `\AtBeginDocument`, because the existence of `pygmentize` isn't tested and may not be final until `\AtEndPreamble`.

```
488 \AtBeginDocument{\minted@checkstyle{}}
```

`\minted@patch@PYGZsq` Patch the `Pygments` single quote macro for `upquote`. The single quote macro from `Pygments` 1.6+ needs to be patched if the `upquote` package is in use. The

conditionals for the patch definition are borrowed from `upquote`. Patching is done `\AtBeginDocument`, after the macros will have been created. Patching is only attempted if the macro exists, so that there is a graceful fallback in the event of a custom `Pygments` stylesheet.

```

489 \newcommand{\minted@patch@PYGZsq}{%
490   \ifcsname PYGZsq\endcsname
491     \ifx\upquote@cmtt\minted@undefined\else
492       \ifx\encodingdefault\upquote@OTone
493         \ifx\ttdefault\upquote@cmtt
494           \expandafter\ifdefstring\expandafter{\csname PYGZsq\endcsname}{\char`\'}%
495           {\expandafter\gdef\csname PYGZsq\endcsname{\char13 }}{}%
496         \else
497           \expandafter\ifdefstring\expandafter{\csname PYGZsq\endcsname}{\char`\'}%
498           {\expandafter\gdef\csname PYGZsq\endcsname{\textquotesingle}}{}%
499         \fi
500       \else
501         \expandafter\ifdefstring\expandafter{\csname PYGZsq\endcsname}{\char`\'}%
502         {\expandafter\gdef\csname PYGZsq\endcsname{\textquotesingle}}{}%
503       \fi
504     \fi
505   \fi
506 }
507 \ifthenelse{\boolean{minted@draft}}{}{\AtBeginDocument{\minted@patch@PYGZsq}}

```

`\minted@def@opt@switch` And we need a switch version.

It would be possible to create a special version of `\minted@get@opt` to work with these, but that would be redundant. During the key processing, any values other than `true` and `false` are filtered out. So when using `\minted@get@opt` later, we know that that part has already been taken care of, and we can just use something like `\ifthenelse{\equal{\minted@get@opt{<opt>}{<default>}}{true}}{...}{...}`. Of course, there is the possibility that a default value has not been set, but `\minted@def@opt@switch` sets a global default of `false` to avoid this. And as usual, `Pygments` values shouldn't be used without considering whether `\minted@get@opt` needs a fallback value.

```

508 \newcommand{\minted@def@opt@switch}[2][false]{%
509   \define@booleankey{minted@opt@g}{#2}%
510   {\@namedef{minted@opt@g:#2}{true}}%
511   {\@namedef{minted@opt@g:#2}{false}}
512   \define@booleankey{minted@opt@g@i}{#2}%
513   {\@namedef{minted@opt@g@i:#2}{true}}%
514   {\@namedef{minted@opt@g@i:#2}{false}}
515   \define@booleankey{minted@opt@lang}{#2}%
516   {\@namedef{minted@opt@lang\minted@lang:#2}{true}}%
517   {\@namedef{minted@opt@lang\minted@lang:#2}{false}}
518   \define@booleankey{minted@opt@lang@i}{#2}%
519   {\@namedef{minted@opt@lang\minted@lang @i:#2}{true}}%

```

```

520     {\@namedef{minted@opt@lang\minted@lang @i:#2}{false}}
521 \define@booleankey{minted@opt@cmd}{#2}%
522     {\@namedef{minted@opt@cmd:#2}{true}}%
523     {\@namedef{minted@opt@cmd:#2}{false}}%
524 \@namedef{minted@opt@g:#2}{#1}%
525 }

```

`\minted@get@opt` We need a way to traverse the hierarchy of values for a given key and return the current value that has precedence. In doing this, we need to specify a default value to use if no value is found. When working with `minted`-specific values, there should generally be a default value; in those cases, an empty default may be supplied. But the macro should also work with `Pygments` settings, which are stored in macros of the same form and will sometimes need to be accessed (for example, `encoding`). In the `Pygments` case, there may very well be no default values on the \LaTeX side, because we are falling back on `Pygments`' own built-in defaults. There is no need to duplicate those when very few `Pygments` values are ever needed; it is simpler to specify the default fallback when accessing the macro value.

From a programming perspective, the default argument value needs to be mandatory, so that `\minted@get@opt` can be fully expandable. This significantly simplifies accessing options.

```

526 \def\minted@get@opt#1#2{%
527   \ifcsname minted@opt@cmd:#1\endcsname
528     \csname minted@opt@cmd:#1\endcsname
529   \else
530     \ifminted@isinline
531       \ifcsname minted@opt@lang\minted@lang @i:#1\endcsname
532         \csname minted@opt@lang\minted@lang @i:#1\endcsname
533       \else
534         \ifcsname minted@opt@g@i:#1\endcsname
535           \csname minted@opt@g@i:#1\endcsname
536         \else
537           \ifcsname minted@opt@lang\minted@lang:#1\endcsname
538             \csname minted@opt@lang\minted@lang:#1\endcsname
539           \else
540             \ifcsname minted@opt@g:#1\endcsname
541               \csname minted@opt@g:#1\endcsname
542             \else
543               #2%
544             \fi
545           \fi
546         \fi
547       \fi
548     \else
549       \ifcsname minted@opt@lang\minted@lang:#1\endcsname
550         \csname minted@opt@lang\minted@lang:#1\endcsname
551       \else
552         \ifcsname minted@opt@g:#1\endcsname

```

```

553         \csname minted@opt@g:#1\endcsname
554         \else
555             #2%
556         \fi
557     \fi
558 \fi
559 \fi
560 }%

```

Actual option definitions. Some of these must be defined conditionally depending on whether we are in draft mode; in draft mode, we need to emulate Pygments functionality with \LaTeX , particularly with `fancyvrb`, when possible. For example, gobbling must be performed by Pygments when draft is off, but when draft is on, `fancyvrb` can perform gobbling.

Lexers.

```

561 \minted@def@optcl{encoding}{-P encoding}{#1}
562 \minted@def@optcl{outencoding}{-P outencoding}{#1}
563 \minted@def@optcl@e{escapeinside}{-P "escapeinside"}{#1"}
564 \minted@def@optcl@switch{stripnl}{-P stripnl}
565 \minted@def@optcl@switch{stripall}{-P stripall}
566 % Python console
567 \minted@def@optcl@switch{python3}{-P python3}
568 % PHP
569 \minted@def@optcl@switch{funcnamehighlighting}{-P funcnamehighlighting}
570 \minted@def@optcl@switch{startinline}{-P startinline}

```

Filters.

```

571 \ifthenelse{\boolean{minted@draft}}{%
572     {\minted@def@optfv{gobble}}%
573     {\minted@def@optcl{gobble}{-F gobble:n}{#1}}
574 \minted@def@optcl{codetagify}{-F codetagify:codetags}{#1}
575 \minted@def@optcl{keywordcase}{-F keywordcase:case}{#1}

```

\LaTeX formatter.

```

576 \minted@def@optcl@switch{texcl}{-P texcomments}
577 \minted@def@optcl@switch{texcomments}{-P texcomments}
578 \minted@def@optcl@switch{mathescape}{-P mathescape}
579 \minted@def@optfv@switch{linenos}
580 \minted@def@opt@style

```

`fancyvrb` options.

```

581 \minted@def@optfv{frame}
582 \minted@def@optfv{framesep}
583 \minted@def@optfv{framerule}
584 \minted@def@optfv{rulecolor}

```

```

585 \minded@def@optfv{numbersep}
586 \minded@def@optfv{numbers}
587 \minded@def@optfv{firstnumber}
588 \minded@def@optfv{stepnumber}
589 \minded@def@optfv{firstline}
590 \minded@def@optfv{lastline}
591 \minded@def@optfv{baselinestretch}
592 \minded@def@optfv{xleftmargin}
593 \minded@def@optfv{xrightmargin}
594 \minded@def@optfv{fillcolor}
595 \minded@def@optfv{tabsize}
596 \minded@def@optfv{fontfamily}
597 \minded@def@optfv{fontsize}
598 \minded@def@optfv{fontshape}
599 \minded@def@optfv{fontseries}
600 \minded@def@optfv{formatcom}
601 \minded@def@optfv{label}
602 \minded@def@optfv@switch{numberblanklines}
603 \minded@def@optfv@switch{showspaces}
604 \minded@def@optfv@switch{resetmargins}
605 \minded@def@optfv@switch{samepage}
606 \minded@def@optfv@switch{showtabs}
607 \minded@def@optfv@switch{obeytabs}
608 % The following are patches currently added onto fancyvrb
609 \minded@def@optfv@switch{breaklines}
610 \minded@def@optfv{breakindent}
611 \minded@def@optfv@switch{breakautoindent}
612 \minded@def@optfv{breaksymbol}
613 \minded@def@optfv{breaksymbolsep}
614 \minded@def@optfv{breaksymbolindent}
615 \minded@def@optfv{breaksymbolleft}
616 \minded@def@optfv{breaksymbolsepleft}
617 \minded@def@optfv{breaksymbolindentleft}
618 \minded@def@optfv{breaksymbolright}
619 \minded@def@optfv{breaksymbolsepright}
620 \minded@def@optfv{breaksymbolindentright}
621 \minded@def@optfv{breakafter}
622 \minded@def@optfv@switch{breakaftergroup}
623 \minded@def@optfv{breakaftersymbolpre}
624 \minded@def@optfv{breakaftersymbolpost}
625 \minded@def@optfv@switch{breakanywhere}
626 \minded@def@optfv{breakanywheresymbolpre}
627 \minded@def@optfv{breakanywheresymbolpost}

```

Finally, options specific to `minded`.

An option to force `breaklines` to work at the `Pygments` token level, rather than at the character level. This is useful in keeping things like strings from being split between lines.

```
628 \minted@def@opt@switch{breakbytoken}
629 \minted@def@opt@switch{breakbytokenanywhere}
```

`bgcolor`: The old `bgcolor` is retained for compatibility. A dedicated framing package will often be preferable.

```
630 \minted@def@opt{bgcolor}
```

Autogobble. We create an option that governs when Python's `textwrap.dedent()` is used to autogobble code.

```
631 \minted@def@opt@switch{autogobble}
```

`\minted@encoding` When working with encoding, we will need access to the current encoding. That may be done via `\minted@get@opt`, but it is more convenient to go ahead and define a shortcut with an appropriate default

```
632 \newcommand{\minted@encoding}{\minted@get@opt{encoding}{UTF8}}
```

9.6 Additions to `fancyvrb`

The following code adds automatic line breaking functionality to `fancyvrb`'s `Verbatim` environment. The code is intentionally written as an extension to `fancyvrb`, rather than as part of `minted`. Once the code has received more use and been further refined, it probably should be separated out into its own package as an extension of `fancyvrb`.

The line breaking defined here is used in `minted`'s `minted` environment and `\mint` command, which use `Verbatim` internally. The `\mintinline` command implements line wrapping using a slightly different system (essentially, `BVerbatim`, with the `\vbox \let to \relax`). This is implemented separately within `minted`, rather than as an extension to `fancyvrb`, for simplicity and because `BVerbatim` wouldn't be itself without the box. Likewise, `breaklines` is not applied to `fancyvrb`'s `\Verb` or `short verb`, since their implementation is different from that of `\mintinline`. Ideally, an extension of `fancyvrb` would add line breaking to these, or (probable better) provide equivalent commands that support breaks.

9.6.1 Setup

All of the additions to `fancyvrb` should be defined conditionally. If an extension to `fancyvrb` (such as that proposed above) is loaded before `minted`, and if this extension provides `breaklines`, then we don't want to overwrite that definition and create a conflict. We assume that any extension of `fancyvrb` would use the `keyval` package, since that is what `fancyvrb` currently uses, and test for the existence of a `fancyvrb` `keyval` key `breaklines`.

```
633 \ifcsname KV@FV@breaklines\endcsname\else
```

9.6.2 Line breaking

Begin by defining keys, with associated macros, bools, and dimens.

`FV@BreakLines` Turn line breaking on or off.

```
634 \newboolean{FV@BreakLines}
635 \let\FV@ListProcessLine@Orig\FV@ListProcessLine
636 \define@booleankey{FV}{breaklines}%
637   {\FV@BreakLinestrue
638     \let\FV@ListProcessLine\FV@ListProcessLine@Break}%
639   {\FV@BreakLinesfalse
640     \let\FV@ListProcessLine\FV@ListProcessLine@Orig}
```

`\FV@BreakIndent`

```
641 \newdimen\FV@BreakIndent
642 \define@key{FV}{breakindent}{\FV@BreakIndent=#1}
643 \fvset{breakindent=0pt}
```

`FV@BreakAutoIndent`

```
644 \newboolean{FV@BreakAutoIndent}
645 \define@booleankey{FV}{breakautoindent}%
646   {\FV@BreakAutoIndentrue}{\FV@BreakAutoIndentfalse}
647 \fvset{breakautoindent=true}
```

`\FancyVerbBreakSymbolLeft` The left-hand symbol indicating a break. Since breaking is done in such a way that a left-hand symbol will often be desired while a right-hand symbol may not be, a shorthand option `breaksymbol` is supplied. This shorthand convention is continued with other options applying to the left-hand symbol.

```
648 \define@key{FV}{breaksymbolleft}{\def\FancyVerbBreakSymbolLeft{#1}}
649 \define@key{FV}{breaksymbol}{\fvset{breaksymbolleft=#1}}
650 \fvset{breaksymbolleft=\tiny\ensuremath{\hookrightarrow}}
```

`\FancyVerbBreakSymbolRight` The right-hand symbol indicating a break.

```
651 \define@key{FV}{breaksymbolright}{\def\FancyVerbBreakSymbolRight{#1}}
652 \fvset{breaksymbolright={}}
```

Separation of break symbols from the text.

`\FV@BreakSymbolSepLeft`

```
653 \newdimen\FV@BreakSymbolSepLeft
654 \define@key{FV}{breaksymbolsepleft}{\FV@BreakSymbolSepLeft=#1}
655 \define@key{FV}{breaksymbolsep}{\fvset{breaksymbolsepleft=#1}}
656 \fvset{breaksymbolsepleft=1em}
```

`\FV@BreakSymbolSepRight`

```
657 \newdimen\FV@BreakSymbolSepRight
658 \define@key{FV}{breaksymbolsepright}{\FV@BreakSymbolSepRight=#1}
659 \fvset{breaksymbolsepright=1em}
```

Additional indentation to make room for the break symbols.

`\FV@BreakSymbolIndentLeft`

```
660 \newdimen\FV@BreakSymbolIndentLeft
661 \settowidth{\FV@BreakSymbolIndentLeft}{\ttfamily xxxx}
662 \define@key{FV}{breaksymbolindentleft}{\FV@BreakSymbolIndentLeft=#1}
663 \define@key{FV}{breaksymbolindent}{\fvset{breaksymbolindentleft=#1}}
```

`\FV@BreakSymbolIndentRight`

```
664 \newdimen\FV@BreakSymbolIndentRight
665 \settowidth{\FV@BreakSymbolIndentRight}{\ttfamily xxxx}
666 \define@key{FV}{breaksymbolindentright}{\FV@BreakSymbolIndentRight=#1}
```

We need macros that contain the logic for typesetting the break symbols. By default, the symbol macros contain everything regarding the symbol and its typesetting, while these macros contain pure logic. The symbols should be wrapped in braces so that formatting commands (for example, `\tiny`) don't escape.

`FancyVerbFormatBreakSymbolLeft`

```
667 \newcommand{\FancyVerbFormatBreakSymbolLeft}[1]{%
668   \ifnum\value{linenumber}=1\relax\else{#1}\fi}
```

`FancyVerbLineBreakLast`

We need a counter for keeping track of the internal line number for the last segment of a broken line, so that we can avoid putting a right continuation symbol there.

```
669 \newcounter{FancyVerbLineBreakLast}
```

`\FV@SetLineBreakLast`

```
670 \newcommand{\FV@SetLineBreakLast}{%
671   \setcounter{FancyVerbLineBreakLast}{\value{linenumber}}}
```

`FancyVerbFormatBreakSymbolRight`

```
672 \newcommand{\FancyVerbFormatBreakSymbolRight}[1]{%
673   \ifnum\value{linenumber}=\value{FancyVerbLineBreakLast}\relax\else{#1}\fi}
```


`FV@BreakAnywhere` Allow line breaking (almost) anywhere.

```
674 \newboolean{FV@BreakAnywhere}
675 \define@booleankey{FV}{breakanywhere}{%
676   {\FV@BreakAnywheret true
677     \let\FancyVerbBreakStart\FV@Break
678     \let\FancyVerbBreakStop\FV@EndBreak
679     \let\FV@Break@Token\FV@Break@AnyToken}%
680   {\FV@BreakAnywher false
681     \let\FancyVerbBreakStart\relax
682     \let\FancyVerbBreakStop\relax}
683 \fvset{breakanywhere=false}
```

`\FancyVerbBreakStart`

```
684 \let\FancyVerbBreakStart\relax
```

`\FancyVerbBreakStop`

```
685 \let\FancyVerbBreakStop\relax
```

`\FV@EscChars` We need to define versions of common escaped characters that reduce to raw characters.

```
686 \edef\FV@hashchar{\string#}
687 \edef\FV@dollarchar{\string$}
688 \edef\FV@ampchar{\string&}
689 \edef\FV@underscorechar{\string_}
690 \edef\FV@tildechar{\string~}
691 \edef\FV@leftsquarebracket{\string[] }
692 \edef\FV@rightsquarebracket{\string[] }
693 \newcommand{\FV@EscChars}{%
694   \let\#\FV@hashchar
695   \let\%\FV@percentchar
696   \let\{\FV@charlb
697   \let\}\FV@charrb
698   \let\$\FV@dollarchar
699   \let\&\FV@ampchar
700   \let\_FV@underscorechar
701   \let\\FV@backslashchar
702   \let~FV@tildechar
703   \let~FV@tildechar
704   \let[FV@leftsquarebracket
705   \let]FV@rightsquarebracket
706 } %$ <- highlighting
```

`\FV@BreakAfter` Allow line breaking (almost) anywhere, but only after specified characters.

```
707 \define@key{FV}{breakafter}{%
```

```

708 \ifstrempy{#1}%
709   {\let\FV@BreakAfter\@empty
710    \let\FancyVerbBreakStart\relax
711    \let\FancyVerbBreakStop\relax}%
712   {\def\FV@BreakAfter{#1}%
713    \let\FancyVerbBreakStart\FV@Break
714    \let\FancyVerbBreakStop\FV@EndBreak
715    \let\FV@Break@Token\FV@Break@AfterToken}%
716   }
717 \fvset{breakafter={}}

```

`FV@BreakAfterGroup` Determine whether breaking after specified characters is always allowed after each individual character, or is only allowed after groups of identical characters.

```

718 \newboolean{FV@BreakAfterGroup}
719 \define@booleankey{FV}{breakaftergroup}%
720   {\FV@BreakAfterGrouptrue}%
721   {\FV@BreakAfterGroupfalse}%
722 \fvset{breakaftergroup=true}

```

`\FV@BreakAfterPrep` We need a way to break after characters if they have been specified as breaking characters. It would be possible to do that via a nested conditional, but that would be messy. It is much simpler to create an empty macro whose name contains the character, and test for the existence of this macro. This needs to be done inside a `\begingroup... \endgroup` so that the macros do not have to be cleaned up manually. A good place to do this is in `\FV@FormattingPrep`, which is inside a group and before processing starts.

The procedure here is a bit roundabout. We need to use `\FV@EscChars` to handle character escapes, but the character redefinitions need to be kept local, requiring that we work within a `\begingroup... \endgroup`. So we loop through the breaking tokens and assemble a macro that will itself define character macros. Only this defining macro is declared global, and it contains *expanded* characters so that there is no longer any dependence on `\FV@EscChars`.

```

723 \def\FV@BreakAfterPrep{%
724   \ifx\FV@BreakAfter\@empty\relax
725   \else
726     \gdef\FV@BreakAfter@Def{}%
727     \begingroup
728     \def\FV@BreakAfter@Process##1##2\FV@Undefined{%
729       \expandafter\FV@BreakAfter@Process@i\expandafter{##1}%
730       \expandafter\ifx\expandafter\relax\detokenize{##2}\relax
731       \else
732         \FV@BreakAfter@Process##2\FV@Undefined
733       \fi
734     }%
735     \def\FV@BreakAfter@Process@i##1{%
736       \g@addto@macro\FV@BreakAfter@Def{%

```

```

737         \@namedef{FV@BreakAfter@Token\detokenize{##1}}{}}%
738     }%
739     \FV@EscChars
740     \expandafter\FV@BreakAfter@Process\FV@BreakAfter\FV@Undefined
741     \endgroup
742     \FV@BreakAfter@Def
743     \fi
744 }
745 \expandafter\def\expandafter\FV@FormattingPrep\expandafter{%
746     \expandafter\FV@BreakAfterPrep\FV@FormattingPrep}

```

`FancyVerbBreakAnywhereSymbolPre` The pre-break symbol for breaks introduced by `breakanywhere`. That is, the symbol before breaks that occur between characters, rather than at spaces.

```

747 \define@key{FV}{breakanywheresymbolpre}{%
748     \ifstrempy{#1}%
749     {\def\FancyVerbBreakAnywhereSymbolPre{}}%
750     {\def\FancyVerbBreakAnywhereSymbolPre{\hbox{#1}}}}
751 \fvset{breakanywheresymbolpre={\,\footnotesize\ensuremath{\_}\rfloor}}

```

`FancyVerbBreakAnywhereSymbolPost` The post-break symbol for breaks introduced by `breakanywhere`.

```

752 \define@key{FV}{breakanywheresymbolpost}{%
753     \ifstrempy{#1}%
754     {\def\FancyVerbBreakAnywhereSymbolPost{}}%
755     {\def\FancyVerbBreakAnywhereSymbolPost{\hbox{#1}}}}
756 \fvset{breakanywheresymbolpost={}}

```

`FancyVerbBreakAfterSymbolPre` The pre-break symbol for breaks introduced by `breakafter`.

```

757 \define@key{FV}{breakaftersymbolpre}{%
758     \ifstrempy{#1}%
759     {\def\FancyVerbBreakAfterSymbolPre{}}%
760     {\def\FancyVerbBreakAfterSymbolPre{\hbox{#1}}}}
761 \fvset{breakaftersymbolpre={\,\footnotesize\ensuremath{\_}\rfloor}}

```

`FancyVerbBreakAfterSymbolPost` The post-break symbol for breaks introduced by `breakafter`.

```

762 \define@key{FV}{breakaftersymbolpost}{%
763     \ifstrempy{#1}%
764     {\def\FancyVerbBreakAfterSymbolPost{}}%
765     {\def\FancyVerbBreakAfterSymbolPost{\hbox{#1}}}}
766 \fvset{breakaftersymbolpost={}}

```

`FancyVerbBreakAnywhereBreak` When `breakanywhere=true`, line breaks may occur at almost any location. This is the macro that governs the breaking in those cases. By default, `\discretionary` is used. `\discretionary` takes three arguments: a character to insert before the

break, a character to insert after the break, and a character to insert if there is no break.

`\discretionary` will generally only insert breaks when breaking at spaces simply cannot make lines short enough (this may be tweaked to some extent with hyphenation settings). This can produce a somewhat ragged appearance in some cases. If you want breaks exactly at the margin (or as close as possible) regardless of whether a break at a space is an option, you may want to use `\allowbreak` instead.

```
767 \newcommand{\FancyVerbBreakAnywhereBreak}{%
768   \discretionary{\FancyVerbBreakAnywhereSymbolPre}%
769   {\FancyVerbBreakAnywhereSymbolPost}{}}
```

`\FancyVerbBreakAfterBreak` The macro governing breaking for `breakafter=true`.

```
770 \newcommand{\FancyVerbBreakAfterBreak}{%
771   \discretionary{\FancyVerbBreakAfterSymbolPre}%
772   {\FancyVerbBreakAfterSymbolPost}{}}
```

Define helper macros.

`\FV@LineBox` A box for saving a line of code, so that its dimensions may be determined and thus we may figure out if it needs line breaking.

```
773 \newsavebox{\FV@LineBox}
```

`\FV@LineIndentBox` A box for saving the indentation of code, so that its dimensions may be determined for use in autoindentation of continuation lines.

```
774 \newsavebox{\FV@LineIndentBox}
```

`\FV@LineIndentChars` A macro for storing the indentation characters, if any, of a given line. For use in autoindentation of continuation lines

```
775 \let\FV@LineIndentChars\@empty
```

`\FV@GetLineIndent` A macro that takes a line and determines the indentation, storing the indentation chars in `\FV@LineIndentChars`.

```
776 \def\FV@GetNextChar{\let\FV@NextChar=}
777 \def\FV@CleanRemainingChars#1\FV@Undefined{}
778 \def\FV@GetLineIndent{\afterassignment\FV@CheckIndentChar\FV@GetNextChar}
779 \def\FV@CheckIndentChar{%
780   \ifx\FV@NextChar\FV@Undefined
781     \let\FV@NextChar=\relax
782   \else
783     \expandafter\ifx\FV@NextChar\FV@Space
```

```

784     \g@addto@macro{\FV@LineIndentChars}{\FV@Space}%
785     \let\FV@Next=\FV@GetLineIndent
786   \else
787     \expandafter\ifx\FV@NextChar\FV@Tab
788     \g@addto@macro{\FV@LineIndentChars}{\FV@Tab}%
789     \let\FV@Next=\FV@GetLineIndent
790   \else
791     \let\FV@Next=\FV@CleanRemainingChars
792   \fi
793 \fi
794 \fi
795 \FV@Next
796 }

```

Define the macros that actually perform `breakanywhere`.

`\FV@Break` The entry macro for breaking lines, either anywhere or after specified characters. The current line (or argument) will be scanned token by token/group by group, and accumulated (with added potential breaks) in `\FV@Tmp`. After scanning is complete, `\FV@Tmp` will be inserted. It would be possible to insert each token/group into the document immediately after it is scanned, instead of accumulating them in a “buffer.” But that would interfere with macros. Even in the current approach, macros that take optional arguments are problematic.⁶

```

797 \def\FV@Break{%
798   \def\FV@Tmp{ }%
799   \FV@Break@Scan
800 }

```

`\FV@EndBreak`

```
801 \def\FV@EndBreak{\FV@Tmp}
```

`\FV@Break@Scan` Look ahead via `\@ifnextchar`. Don’t do anything if we’re at the end of the region to be scanned. Otherwise, invoke a macro to deal with what’s next based on whether it is math, or a group, or something else.

This and some following macros are defined inside of groups, to ensure proper catcodes.

```

802 \begingroup
803 \catcode`\$=3%
804 \gdef\FV@Break@Scan{%
805   \@ifnextchar\FV@EndBreak%
806   { }%

```

⁶Through a suitable definition that tracks the current state and looks for square brackets, this might be circumvented. Then again, in verbatim contexts, macro use should be minimal, so the restriction to macros without optional arguments should generally not be an issue.

```

807   {\ifx\@let@token$\relax
808     \let\FV@Break@Next\FV@Break@Math
809   \else
810     \ifx\@let@token\bgroup\relax
811       \let\FV@Break@Next\FV@Break@Group
812     \else
813       \let\FV@Break@Next\FV@Break@Token
814     \fi
815   \fi
816   \FV@Break@Next}%
817 }
818 \endgroup

```

`\FV@Break@Math` Grab an entire math span, and insert it into `\FV@Tmp`. Due to grouping, this works even when math contains things like `\text{x}`. After dealing with the math span, continue scanning.

```

819 \begingroup
820 \catcode`\$=3%
821 \gdef\FV@Break@Math$#1${%
822   \g@addto@macro{\FV@Tmp}{$#1$}%
823   \FV@Break@Scan}
824 \endgroup

```

`\FV@Break@Group` Grab the group, and insert it into `\FV@Tmp` (as a group) before continuing scanning.

```

825 \def\FV@Break@Group#1{%
826   \g@addto@macro{\FV@Tmp}{#1}}%
827   \FV@Break@Scan}

```

`\FV@Break@Token` This macro is `\let` to `\FV@Break@AnyToken` or `\FV@Break@AfterToken` by the `breakanywhere` and `breakafter` options, so it is not explicitly defined.

`\FV@Break@AnyToken` Deal with breaking around any token.

If it is ever necessary, it would be possible to create a more sophisticated version involving `catcode` checks via `\ifcat`. Something like this:

```

\begingroup
\catcode`\a=11
\catcode`\+=12
\gdef\FV@Break...
  \ifcat\noexpand#1a
    \g@addto@macro{\FV@Tmp}...
  \else
  ...
\endgroup

```

This doesn't break macros with *mandatory* arguments, because `\FancyVerbBreakAnywhereBreak` is inserted *before* the token. Groups themselves are added without any special

handling. So a macro would end up right next to its original arguments, without anything being inserted. Optional arguments will cause this approach to fail; there is currently no attempt to identify them, since that is a much harder problem.

```
828 \def\FV@Break@AnyToken#1{%
829   \g@addto@macro{\FV@Tmp}{\FancyVerbBreakAnywhereBreak#1}%
830   \FV@Break@Scan}
```

`\FV@Break@AfterToken` Deal with breaking around only specified tokens. This is a bit trickier. We only break if a macro corresponding to the token exists. We also need to check whether the specified token should be grouped, that is, whether breaks are allowed between identical characters. All of this has to be written carefully so that nothing is accidentally inserted into the stream for future scanning.

Dealing with tokens followed by empty groups (for example, `\x{}`) is particularly challenging when we want to avoid breaks between identical characters. When a token is followed by a group, we need to save the current token for later reference (`\x` in the example), then capture and save the following group, and then—only if the group was empty—see if the following token is identical to the old saved token.

```
831 \def\FV@Break@AfterToken#1{%
832   \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
833     \let\FV@Break@Next\FV@Break@AfterTokenBreak
834   \else
835     \let\FV@Break@Next\FV@Break@AfterTokenNoBreak
836   \fi
837   \FV@Break@Next{#1}%
838 }
839 \def\FV@Break@AfterTokenNoBreak#1{%
840   \g@addto@macro{\FV@Tmp}{#1}%
841   \FV@Break@Scan}
842 \def\FV@Break@AfterTokenBreak#1{%
843   \@ifnextchar\FV@Space%
844     {\g@addto@macro{\FV@Tmp}{#1}\FV@Break@Scan}%
845     {\ifthenelse{\boolean{FV@BreakAfterGroup}}%
846       {\ifx\let@token#1\relax
847         \g@addto@macro{\FV@Tmp}{#1}%
848         \let\FV@Break@Next\FV@Break@Scan
849       \else
850         \ifx\let@token\bgroup\relax
851           \g@addto@macro{\FV@Tmp}{#1}%
852           \let\FV@TmpToken#1%
853           \let\FV@Break@Next\FV@Break@AfterTokenBreak@Group
854         \else
855           \g@addto@macro{\FV@Tmp}{#1\FancyVerbBreakAfterBreak}%
856           \let\FV@Break@Next\FV@Break@Scan
857         \fi
858       \fi}%
859   {\g@addto@macro{\FV@Tmp}{#1\FancyVerbBreakAfterBreak}%}
```

```

860     \let\FV@Break@Next\FV@Break@Scan}%
861     \FV@Break@Next}%
862 }
863 \def\FV@Break@AfterTokenBreak@Group#1{%
864   \g@addto@macro{\FV@Tmp}{\#1}}%
865   \ifstrempy{\#1}%
866   {\let\FV@Break@Next\FV@Break@AfterTokenBreak@Group@i}%
867   {\let\FV@Break@Next\FV@Break@Scan}%
868   \FV@Break@Next}
869 \def\FV@Break@AfterTokenBreak@Group@i{%
870   \@ifnextchar\FV@TmpToken%
871   {\FV@Break@Scan}%
872   {\g@addto@macro{\FV@Tmp}{\FancyVerbBreakAfterBreak}}%
873   \FV@Break@Scan}}

```

And finally the really important things.

`\FV@makeLineNumber` We need a version of `lineno`'s `\makeLineNumber` that is adapted for our purposes. This is adapted directly from the example `\makeLineNumber` that is given in the `lineno` documentation under the discussion of internal line numbers. The `\FV@SetLineBreakLast` is needed to determine the internal line number of the last segment of the broken line, so that we can disable the right-hand break symbol on this segment. When a right-hand break symbol is in use, a line of code will be processed twice: once to determine the last internal line number, and once to use this information only to insert right-hand break symbols on the appropriate lines. During the second run, `\FV@SetLineBreakLast` is disabled by `\letting` it to `\relax`.

```

874 \def\FV@makeLineNumber{%
875   \hss
876   \FancyVerbFormatBreakSymbolLeft{\FancyVerbBreakSymbolLeft}%
877   \hbox to \FV@BreakSymbolSepLeft{\hfill}%
878   \rlap{\hskip\linewidth
879     \hbox to \FV@BreakSymbolSepRight{\hfill}%
880     \FancyVerbFormatBreakSymbolRight{\FancyVerbBreakSymbolRight}}%
881     \FV@SetLineBreakLast
882   }%
883 }

```

`\FV@SaveLineBox` This is the macro that does most of the work. This was inspired by Marco Daniel's code at <http://tex.stackexchange.com/a/112573/10742>.

This macro is invoked when a line is too long. We modify the `\linewidth` to take into account `breakindent` and `breakautoindent`, and insert `\hboxes` to fill the empty space. We also account for `breaksymbolindentleft` and `breaksymbolindentright`, but *only* when there are actually break symbols. The code is placed in a `\parbox`. Break symbols are inserted via `lineno`'s `internallinenumbers*`, which does internal line numbers without continuity

between environments (the `linenumber` counter is automatically reset). The beginning of the code has negative `\hspace` inserted to pull it out to the correct starting position. `\struts` are used to maintain correct line heights. The `\parbox` is followed by an empty `\hbox` that takes up the space needed for a right-hand break symbol (if any).

```

884 \def\FV@SaveLineBox#1{%
885   \savebox{\FV@LineBox}{%
886     \advance\linewidth by -\FV@BreakIndent
887     \hbox to \FV@BreakIndent{\hfill}%
888     \ifthenelse{\boolean{FV@BreakAutoIndent}}{%
889       {\let\FV@LineIndentChars\@empty
890        \FV@GetLineIndent#1\FV@Undefined
891         \savebox{\FV@LineIndentBox}{\FV@LineIndentChars}%
892         \hbox to \wd\FV@LineIndentBox{\hfill}%
893         \advance\linewidth by -\wd\FV@LineIndentBox}%
894       }%
895     \ifdefempty{\FancyVerbBreakSymbolLeft}}{%
896       {\hbox to \FV@BreakSymbolIndentLeft{\hfill}%
897        \advance\linewidth by -\FV@BreakSymbolIndentLeft}%
898     \ifdefempty{\FancyVerbBreakSymbolRight}}{%
899       {\advance\linewidth by -\FV@BreakSymbolIndentRight}%
900     \parbox[t]{\linewidth}{%
901       \raggedright
902       \leftlinenumber*
903       \begin{internallinenumber*}%
904         \let\makeLineNumber\FV@makeLineNumber
905         \noindent\hspace*{-\FV@BreakIndent}%
906         \ifdefempty{\FancyVerbBreakSymbolLeft}}{%
907           \hspace*{-\FV@BreakSymbolIndentLeft}}%
908         \ifthenelse{\boolean{FV@BreakAutoIndent}}{%
909           {\hspace*{-\wd\FV@LineIndentBox}}%
910         }%
911         \strut#1\nobreak\strut
912       \end{internallinenumber*}
913     }%
914     \ifdefempty{\FancyVerbBreakSymbolRight}}{%
915       {\hbox to \FV@BreakSymbolIndentRight{\hfill}}%
916     }%
917 }

```

`\FancyVerbFormatText` The introduction of line breaks introduces an issue for `\FancyVerbFormatLine`. Does it format the entire line (outside the `\parbox`), or only the text part of the line (inside the `\parbox`)? Since both might be desirable, `\FancyVerbFormatLine` is assigned to the entire line, and a new macro `\FancyVerbFormatText` is assigned to the text, within the `\parbox`.

```

918 \def\FancyVerbFormatText#1{#1}

```

`\FV@ListProcessLine@Break` This macro is based on `\FV@ListProcessLine` and follows it as closely as possible. The `\linewidth` is reduced by `\FV@FrameSep` and `\FV@FrameRule` so that text will not overrun frames. This is done conditionally based on which frames are in use. We save the current line in a box, and only do special things if the box is too wide. For uniformity, all text is placed in a `\parbox`, even if it doesn't need to be wrapped.

If a line is too wide, then it is passed to `\FV@SaveLineBox`. If there is no right-hand break symbol, then the saved result in `\FV@LineBox` may be used immediately. If there is a right-hand break symbol, then the line must be processed a second time, so that the right-hand break symbol may be removed from the final segment of the broken line (since it does not continue). During the first use of `\FV@SaveLineBox`, the counter `FancyVerbLineBreakLast` is set to the internal line number of the last segment of the broken line. During the second use of `\FV@SaveLineBox`, we disable this (`\let\FV@SetLineBreakLast\relax`) so that the value of `FancyVerbLineBreakLast` remains fixed and thus may be used to determine when a right-hand break symbol should be inserted.

```

919 \def\FV@ListProcessLine@Break#1{%
920   \hbox to \hsize{%
921     \kern\leftmargin
922     \hbox to \linewidth{%
923       \ifx\FV@RightListFrame\relax\else
924         \advance\linewidth by -\FV@FrameSep
925         \advance\linewidth by -\FV@FrameRule
926       \fi
927       \ifx\FV@LeftListFrame\relax\else
928         \advance\linewidth by -\FV@FrameSep
929         \advance\linewidth by -\FV@FrameRule
930       \fi
931       \sbox{\FV@LineBox}{\FancyVerbFormatLine{\FancyVerbFormatText{#1}}}%
932       \ifdim\wd\FV@LineBox>\linewidth
933         \setcounter{FancyVerbLineBreakLast}{0}%
934         \FV@SaveLineBox{\FancyVerbFormatText{%
935           \FancyVerbBreakStart#1\FancyVerbBreakStop}}%
936         \ifdefempty{\FancyVerbBreakSymbolRight}{}{%
937           \let\FV@SetLineBreakLast\relax
938           \FV@SaveLineBox{\FancyVerbFormatText{%
939             \FancyVerbBreakStart#1\FancyVerbBreakStop}}%
940           \FV@LeftListNumber
941           \FV@LeftListFrame
942           \FancyVerbFormatLine{\usebox{\FV@LineBox}}%
943           \FV@RightListFrame
944           \FV@RightListNumber
945         \else
946           \FV@LeftListNumber
947           \FV@LeftListFrame
948           \FancyVerbFormatLine{%
949             \parbox[t]{\linewidth}{\noindent\strut\FancyVerbFormatText{#1}\strut}}%

```

```

950     \FV@RightListFrame
951     \FV@RightListNumber
952     \fi}%
953     \hss}\baselineskip\z@\lineskip\z@}

```

9.7 linenos

Since `fancyvrb` currently doesn't have a `linenos` key, we create one that mimics `numbers=left` (but only after checking to make sure that another package hasn't already patched this).

```

954 \ifcsname KV@FV@linenos\endcsname\else
955 \define@booleankey{FV}{linenos}%
956   {\@nameuse{FV@Numbers@left}}{\@nameuse{FV@Numbers@none}}
957 \fi

```

9.8 Cleanup

Finally, end the conditional creation of `fancyvrb` extensions.

```

958 \fi

```

9.9 Internal helpers

`\minted@bgbox` Define an environment that may be wrapped around a `minted` environment to assign a background color. This is retained as a holdover from version 1.0. In most cases, it is probably better to use a dedicated framing package, such as `tcolorbox` or `mdframed`.

First, we need to define a new save box.

```

959 \newsavebox{\minted@bgbox}

```

Now we can define the environment that captures a code fragment inside a `minipage` and applies a background color.

```

960 \newenvironment{minted@colorbg}[1]{
961   %\setlength{\fboxsep}{-\fboxrule}
962   \def\minted@bgcol{#1}
963   \noindent
964   \begin{lrbox}{\minted@bgbox}
965   \begin{minipage}{\linewidth-2\fboxsep}}
966   {\end{minipage}
967   \end{lrbox}}%
968   \colorbox{\minted@bgcol}{\usebox{\minted@bgbox}}

```

`\minted@code` Create a file handle for saving code (and anything else that must be written to temp files).

```
969 \newwrite\minted@code
```

`\minted@savecode` Save code to be pygmentized to a file.

```
970 \newcommand{\minted@savecode}[1]{
971   \immediate\openout\minted@code\minted@jobname.pyg\relax
972   \immediate\write\minted@code{\expandafter\detokenize\expandafter{#1}}%
973   \immediate\closeout\minted@code}
```

`minted@FancyVerbLineTemp` At various points, we will need a temporary counter for storing and then restoring the value of `FancyVerbLine`. When using the `langlinenos` option, we need to store the current value of `FancyVerbLine`, then set `FancyVerbLine` to the current value of a language-specific counter, and finally restore `FancyVerbLine` to its initial value after the current chunk of code has been typeset. In patching `VerbatimOut`, we need to prevent `FancyVerbLine` from being incremented during the write process.

```
974 \newcounter{minted@FancyVerbLineTemp}
```

`\minted@FVB@VerbatimOut` We need a custom version of `fancyvrb`'s `\FVB@VerbatimOut` that supports Unicode (everything written to file is `\detokenized`). We also need to prevent the value of `FancyVerbLine` from being incorrectly incremented.

```
975 \newcommand{\minted@write@detok}[1]{%
976   \immediate\write\FV@OutFile{\detokenize{#1}}}
977 \newcommand{\minted@FVB@VerbatimOut}[1]{%
978   \setcounter{minted@FancyVerbLineTemp}{\value{FancyVerbLine}}%
979   \@bsphack
980   \begingroup
981     \FV@UseKeyValues
982     \FV@DefineWhiteSpace
983     \def\FV@Space{\space}%
984     \FV@DefineTabOut
985     \let\FV@ProcessLine\minted@write@detok
986     \immediate\openout\FV@OutFile #1\relax
987     \let\FV@FontScanPrep\relax
988     \let\@noligs\relax
989     \FV@Scan}
```

`\minted@FVE@VerbatimOut` Likewise, we need a custom version of `\FVE@VerbatimOut` that completes the protection of `FancyVerbLine` from being incremented.

```
990 \newcommand{\minted@FVE@VerbatimOut}{%
991   \immediate\closeout\FV@OutFile\endgroup\@esphack
992   \setcounter{FancyVerbLine}{\value{minted@FancyVerbLineTemp}}}%
```

`\MintedPygmentize` We need a way to customize the executable/script that is called to perform highlighting. Typically, we will want `pygmentize`. But advanced users might wish to use a custom Python script instead. The command is only defined if it does not exist. In general, the command should be `\renewcommanded` after the package is loaded, but this way, it will work if defined before `minted` is loaded.

```
993 \ifcscname MintedPygmentize\endcscname\else
994   \newcommand{\MintedPygmentize}{pygmentize}
995 \fi
```

`\minted@pygmentize` Pygmentize a file (default: `\minted@outputdir\minted@jobname.pyg`) using the options provided.

Unfortunately, the logic for caching is a little complex due to operations that are OS- and engine-dependent.

The name of cached files is the result of concatenating the md5 of the code and the md5 of the command. This results in a filename that is longer than ideal (64 characters plus path and extension). Unfortunately, this is the only robust approach that is possible using the built-in pdfTeX hashing capabilities.⁷ LuaTeX could do better, by hashing the command and code together. The Python script that provides XeTeX capabilities simply runs both the command and the code through a single sha1 hasher, but has the additional overhead of the `\write18` call and Python execution.

One potential concern is that caching should also keep track of the command from which code originates. What if identical code is highlighted with identical settings in both the `minted` environment and `\mintinline` command? In both cases, what is actually saved by Pygments is identical. The difference in final appearance is due to how the environment and command treat the Pygments output.

This macro must always be checked carefully whenever it is modified. Under no circumstances should `#1` be written to or opened by Python in write mode. When `\inputminted` is used, `#1` will be an external file that is brought in for highlighting, so it must be left intact.

```
996 \newcommand{\minted@pygmentize}[2][\minted@outputdir\minted@jobname.pyg]{%
997   \ifthenelse{\equal{\minted@get@opt{autogobble}}{false}}{true}}%
998   {\def\minted@codefile{\minted@outputdir\minted@jobname.pyg}}%
999   {\def\minted@codefile{#1}}%
1000  \ifthenelse{\boolean{minted@isinline}}%
1001    {\def\minted@optlistcl@inlines%
1002     \minted@optlistcl@g%i
1003     \cscname minted@optlistcl@lang\minted@lang @i\endcscname}}%
1004    {\let\minted@optlistcl@inlines\@empty}%
1005  \def\minted@cmd{%
```

⁷It would be possible to use only the cache of the code, but that approach breaks down as soon as the code is used multiple times with different options. While that may seem unlikely in practice, it occurs in this documentation and may be expected to occur in other docs.

```

1006 \ifminted@kpsewhich\ifwindows powershell\space\fi\fi
1007 \MintedPygmentize\space -l #2
1008 -f latex -P commandprefix=PYG -F tokenmerge
1009 \minted@optlistcl@g \csname minted@optlistcl@lang\minted@lang\endcsname
1010 \minted@optlistcl@inlines
1011 \minted@optlistcl@cmd -o "\minted@outputdir\minted@infile"
1012 \ifminted@kpsewhich
1013 \ifwindows
1014 \detokenize{$}(kpsewhich "\minted@codefile")%
1015 \else
1016 \detokenize{`}kpsewhich "\minted@codefile"
1017 \detokenize{||} "\minted@codefile"\detokenize{`}%
1018 \fi
1019 \else
1020 "\minted@codefile"
1021 \fi}%
1022 % For debugging, uncomment: %%%
1023 % \immediate\typeout{\minted@cmd}%
1024 % %%%
1025 \ifthenelse{\boolean{minted@cache}}%
1026 {%
1027 \ifx\XeTeXinterchartoks\minted@undefined
1028 \ifthenelse{\equal{\minted@get@opt{autogobble}{false}}{true}}%
1029 {\edef\minted@hash{\pdf@filemdfivesum{#1}%
1030 \pdf@mdfivesum{\minted@cmd autogobble}}}%
1031 {\edef\minted@hash{\pdf@filemdfivesum{#1}%
1032 \pdf@mdfivesum{\minted@cmd}}}%
1033 \else
1034 \immediate\openout\minted@code\minted@jobname.mintedcmd\relax
1035 \immediate\write\minted@code{\minted@cmd}%
1036 \ifthenelse{\equal{\minted@get@opt{autogobble}{false}}{true}}%
1037 {\immediate\write\minted@code{autogobble}}}%
1038 \immediate\closeout\minted@code
1039 %Cheating a little here by using ASCII codes to write `` and ``
1040 %in the Python code
1041 \def\minted@hashcmd{%
1042 \detokenize{python -c "import hashlib;
1043 hasher = hashlib.sha1();
1044 f = open("\)\minted@outputdir\minted@jobname.mintedcmd\detokenize{\", \
1045 hasher.update(f.read());
1046 f.close();
1047 f = open("\)#1\detokenize{\", \rb");
1048 hasher.update(f.read());
1049 f.close();
1050 f = open("\)\minted@outputdir\minted@jobname.mintedcmd5\detokenize{\", \
1051 macro = "\\edef\minted@hash\" + chr(123) + hasher.hexdigest() + chr(1
1052 f.write("\\makeatletter\" + macro + "\\makeatother\endinput\n");
1053 f.close();"}}%
1054 \immediate\write18{\minted@hashcmd}%
1055 \minted@input{\minted@outputdir\minted@jobname.mintedcmd5}%

```

```

1056     \fi
1057     \edef\minted@infile{\minted@cachedir/\minted@hash.pygtex}%
1058     \IfFileExists{\minted@infile}{}{%
1059         \ifthenelse{\equal{\minted@get@opt{autogobble}{false}}{true}}{%
1060             %Need a version of open() that supports encoding under Python 2
1061             \edef\minted@autogobblecmd{%
1062                 \detokenize{python -c "import sys;
1063                     import textwrap;
1064                     from io import open;
1065                     f = open("\">#1\detokenize{"\, \"r\", encoding=\"}\minted@encoding\detoken
1066                     t = f.read();
1067                     f.close();
1068                     f = open("\">\minted@outputdir\minted@jobname.pyg\detokenize{"\, \"w\", en
1069                     f.write(textwrap.dedent(t));
1070                     f.close();"}%
1071             }%
1072             \immediate\write18{\minted@autogobblecmd}}{}%
1073             \immediate\write18{\minted@cmd}}%
1074             \expandafter\minted@addcachefile\expandafter{\minted@hash.pygtex}%
1075             \minted@inputpyg}%
1076     {%
1077     \ifthenelse{\equal{\minted@get@opt{autogobble}{false}}{true}}{%
1078         %Need a version of open() that supports encoding under Python 2
1079         \edef\minted@autogobblecmd{%
1080             \detokenize{python -c "import sys;
1081                 import textwrap;
1082                 from io import open;
1083                 f = open("\">#1\detokenize{"\, \"r\", encoding=\"}\minted@encoding\detoken
1084                 t = f.read();
1085                 f.close();
1086                 f = open("\">\minted@outputdir\minted@jobname.pyg\detokenize{"\, \"w\", en
1087                 f.write(textwrap.dedent(t));
1088                 f.close();"}%
1089         }%
1090         \immediate\write18{\minted@autogobblecmd}}{}%
1091         \immediate\write18{\minted@cmd}}%
1092         \minted@inputpyg}%
1093 }

```

`\minted@inputpyg` For increased clarity, the actual `\input` process is separated out into its own macro. The `bgcolor` option needs to be dealt with in different ways depending on whether we are using `\mintinline`. It is simplest to apply this option here, so that the macro redefinitions may be local and thus do not need to be manually reset later. `\FV@Space` is also patched for math mode, so that space characters will vanish rather than appear as literal spaces within math mode. To simplify the logic, `breakbytoken` is turned on if `breakbytokenanywhere` is on.

At the last possible moment, `\PYG` is `\let` to `\PYG<style>`. All modifications to the style macro for breaking are made to `\PYG<style>` rather than `\PYG`, so that

the \leting that must ultimately take place will indeed do what is intended.

```
1094 \def\FV@SpaceMMode{ }
1095 \def\minted@BreakAfterPrep@extension{%
1096   \ifcsname FV@BreakAfter@Token\@backslashchar\endcsname
1097     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZbs}}{}%
1098   \fi
1099   \ifcsname FV@BreakAfter@Token\FV@underscorechar\endcsname
1100     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZus}}{}%
1101   \fi
1102   \ifcsname FV@BreakAfter@Token\@charlb\endcsname
1103     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZob}}{}%
1104   \fi
1105   \ifcsname FV@BreakAfter@Token\@charrb\endcsname
1106     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZcb}}{}%
1107   \fi
1108   \ifcsname FV@BreakAfter@Token\detokenize{^}\endcsname
1109     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZca}}{}%
1110   \fi
1111   \ifcsname FV@BreakAfter@Token\FV@ampchar\endcsname
1112     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZam}}{}%
1113   \fi
1114   \ifcsname FV@BreakAfter@Token\detokenize{<}\endcsname
1115     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZlt}}{}%
1116   \fi
1117   \ifcsname FV@BreakAfter@Token\detokenize{>}\endcsname
1118     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZgt}}{}%
1119   \fi
1120   \ifcsname FV@BreakAfter@Token\FV@hashchar\endcsname
1121     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZsh}}{}%
1122   \fi
1123   \ifcsname FV@BreakAfter@Token\@percentchar\endcsname
1124     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZpc}}{}%
1125   \fi
1126   \ifcsname FV@BreakAfter@Token\FV@dollarchar\endcsname
1127     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZdl}}{}%
1128   \fi
1129   \ifcsname FV@BreakAfter@Token\detokenize{-}\endcsname
1130     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZhy}}{}%
1131   \fi
1132   \ifcsname FV@BreakAfter@Token\detokenize{'}\endcsname
1133     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZsq}}{}%
1134   \fi
1135   \ifcsname FV@BreakAfter@Token\detokenize{"}\endcsname
1136     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZdq}}{}%
1137   \fi
1138   \ifcsname FV@BreakAfter@Token\FV@tildechar\endcsname
1139     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZti}}{}%
1140   \fi
1141   \ifcsname FV@BreakAfter@Token\detokenize{@}\endcsname
```



```

1142     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZat}}{}%
1143     \fi
1144     \ifcsname FV@BreakAfter@Token\detokenize{[]\endcsname
1145     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZlb}}{}%
1146     \fi
1147     \ifcsname FV@BreakAfter@Token\detokenize{[]\endcsname
1148     \@namedef{FV@BreakAfter@Token\detokenize{\PYGZrb}}{}%
1149     \fi
1150 }
1151 \newcommand{\minted@inputpyg}{%
1152 \let\FV@BreakAfterPrep@orig\FV@BreakAfterPrep
1153 \def\FV@BreakAfterPrep{%
1154 \FV@BreakAfterPrep@orig\minted@BreakAfterPrep@extension}%
1155 \everymath\expandafter{\the\everymath\let\FV@Space\FV@SpaceMMode}%
1156 \ifthenelse{\equal{\minted@get@opt{breakbytokenanywhere}}{false}}{true}}%
1157 {\setkeys{minted@opt@cmd}{breakbytoken=true}}}%
1158 \ifthenelse{\boolean{FV@BreakAnywhere}}%
1159 {\expandafter\let\expandafter\minted@orig@PYG@breakanywhere%
1160 \csname PYG\minted@get@opt{style}{default}\endcsname
1161 \expandafter\def\csname PYG\minted@get@opt{style}{default}\endcsname##1##2{%
1162 \minted@orig@PYG@breakanywhere{##1}%
1163 {\FancyVerbBreakStart##2\FancyVerbBreakStop}}}%
1164 \ifx\FV@BreakAfter\@empty
1165 \else
1166 \expandafter\let\expandafter\minted@orig@PYG@breakafter%
1167 \csname PYG\minted@get@opt{style}{default}\endcsname
1168 \expandafter\def\csname PYG\minted@get@opt{style}{default}\endcsname##1##2{%
1169 \minted@orig@PYG@breakafter{##1}%
1170 {\FancyVerbBreakStart##2\FancyVerbBreakStop}}%
1171 \fi
1172 \ifthenelse{\boolean{minted@isinline}}%
1173 {\ifthenelse{\equal{\minted@get@opt{breaklines}}{false}}{true}}%
1174 {\let\FV@BeginVBox\relax
1175 \let\FV@EndVBox\relax
1176 \def\FV@BProcessLine##1{\FancyVerbFormatLine{##1}}%
1177 \ifthenelse{\equal{\minted@get@opt{breakbytoken}}{false}}{true}}%
1178 {\minted@inputpyg@breakbytoken
1179 \minted@inputpyg@inline}%
1180 {\minted@inputpyg@inline}}%
1181 {\minted@inputpyg@inline}}%
1182 {\ifthenelse{\equal{\minted@get@opt{breaklines}}{false}}{true}}%
1183 {\ifthenelse{\equal{\minted@get@opt{breakbytoken}}{false}}{true}}%
1184 {\minted@inputpyg@breakbytoken
1185 \minted@inputpyg@block}%
1186 {\minted@inputpyg@block}}%
1187 {\minted@inputpyg@block}}%
1188 }
1189 \def\minted@inputpyg@breakbytoken{%
1190 \expandafter\let\expandafter\minted@orig@PYG@breakbytoken%
1191 \csname PYG\minted@get@opt{style}{default}\endcsname

```

```

1192 \ifthenelse{\equal{\minted@get@opt{breakbytokenanywhere}{false}}{true}}%
1193   {\let\minted@orig@allowbreak\allowbreak
1194   \def\allowbreak{\let\allowbreak\minted@orig@allowbreak}%
1195   \expandafter\def\csname PYG\minted@get@opt{style}{default}\endcsname##1##2{%
1196     \allowbreak}\leavevmode\hbox{\minted@orig@PYG@breakbytoken{##1}{##2}}}%
1197   {\expandafter\def\csname PYG\minted@get@opt{style}{default}\endcsname##1##2{%
1198     \leavevmode\hbox{\minted@orig@PYG@breakbytoken{##1}{##2}}}%
1199   }
1200 \def\minted@inputpyg@inline{%
1201   \expandafter\let\expandafter\PYG%
1202     \csname PYG\minted@get@opt{style}{default}\endcsname
1203   \ifthenelse{\equal{\minted@get@opt{bgcolor}}{}}{}%
1204   {\minted@input{\minted@outputdir\minted@infile}}%
1205   {\colorbox{\minted@get@opt{bgcolor}}{\minted@input{\minted@outputdir\minted@infile}}}%
1206   }
1207 }
1208 \def\minted@inputpyg@block{%
1209   \expandafter\let\expandafter\PYG%
1210     \csname PYG\minted@get@opt{style}{default}\endcsname
1211   \ifthenelse{\equal{\minted@get@opt{bgcolor}}{}}{}%
1212   {\minted@input{\minted@outputdir\minted@infile}}%
1213   {\begin{minted@colorbg}{\minted@get@opt{bgcolor}}{\minted@input{\minted@outputdir\minted@infile}}}%
1214   \end{minted@colorbg}}
1215 }

```

We need a way to have line counters on a per-language basis.

`\minted@langlinenoson`

```

1216 \newcommand{\minted@langlinenoson}{%
1217   \ifcsname c@minted@lang\minted@lang\endcsname\else
1218     \newcounter{minted@lang\minted@lang}%
1219     \fi
1220   \setcounter{minted@FancyVerbLineTemp}{\value{FancyVerbLine}}%
1221   \setcounter{FancyVerbLine}{\value{minted@lang\minted@lang}}%
1222 }

```

`\minted@langlinenosoff`

```

1223 \newcommand{\minted@langlinenosoff}{%
1224   \setcounter{minted@lang\minted@lang}{\value{FancyVerbLine}}%
1225   \setcounter{FancyVerbLine}{\value{minted@FancyVerbLineTemp}}%
1226 }

```

Disable the language-specific settings if the package option isn't used.

```

1227 \ifthenelse{\boolean{minted@langlinenos}}{}%
1228   \let\minted@langlinenoson\relax
1229   \let\minted@langlinenosoff\relax
1230 }

```

9.10 Public API

`\setminted` Set global or language-level options.

```
1231 \newcommand{\setminted}[2][]{%
1232   \ifthenelse{\equal{#1}{}}{%
1233     {\setkeys{minted@opt@g}{#2}}%
1234     {\minted@configlang{#1}%
1235       \setkeys{minted@opt@lang}{#2}}}
```

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

```
1236 \newcommand{\setmintedinline}[2][]{%
1237   \ifthenelse{\equal{#1}{}}{%
1238     {\setkeys{minted@opt@g@i}{#2}}%
1239     {\minted@configlang{#1}%
1240       \setkeys{minted@opt@lang@i}{#2}}}
```

Now that the settings macros exist, we go ahead and create any needed defaults.

```
1241 \setmintedinline[php]{startinline=true}
```

`\usemintedstyle` Set style. This is a holdover from version 1, since `\setminted` can now accomplish this, and a hierarchy of style settings are now possible.

```
1242 \newcommand{\usemintedstyle}[2][]{\setminted[#1]{style=#2}}
```

`\minted@defwhitespace@retok` The `\mint` and `\mintinline` commands need to be able to retokenize the code they collect, particularly in draft mode. Retokenization involves expansion combined with `\scantokens`, with active space and tab characters. The active characters need to expand to the appropriate `fancyvrb` macros, but the macros themselves should not be expanded. We need a macro that will accomplish the appropriate definitions.

```
1243 \begingroup
1244 \catcode`\ =\active
1245 \catcode`\^^I=\active
1246 \gdef\minted@defwhitespace@retok{\def {\noexpand\FV@Space}\def^^I{\noexpand\FV@Tab}
1247 \endgroup
```

`\minted@writecmdcode` The `\mintinline` and `\mint` commands will need to write the code they capture to a temporary file for highlighting. It will be convenient to be able to accomplish this via a simple macro, since that makes it simpler to deal with any expansion of what is to be written. This isn't needed for the `minted` environment, because the (patched) `VerbatimOut` is used.

```
1248 \newcommand{\minted@writecmdcode}[1]{%
```

```

1249 \immediate\openout\minted@code\minted@jobname.pyg\relax
1250 \immediate\write\minted@code{\detokenize{#1}}%
1251 \immediate\closeout\minted@code}

```

`\mintinline` Define an inline command. This requires some catcode acrobatics. The typical verbatim methods are not used. Rather, a different approach is taken that is generally more robust when used within other commands (for example, when used in footnotes).

Pygments saves code wrapped in a `Verbatim` environment. Getting the inline command to work correctly require redefining `Verbatim` to be `BVerbatim` temporarily. This approach would break if `BVerbatim` were ever redefined elsewhere.

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

In the case of draft mode, the code is captured and retokenized. Then the internals of `fancyvrb` are used to emulate `SaveVerbatim`, so that `\BUseVerbatim` may be employed.

The `FancyVerbLine` counter is altered somehow within `\minted@pygmentize`, so we protect against this.

```

1252 \newrobustcmd{\mintinline}[2][ ]{%
1253   \begingroup
1254   \setboolean{minted@isinline}{true}%
1255   \minted@configlang{#2}%
1256   \setkeys{minted@opt@cmd}{#1}%
1257   \minted@fvset
1258   \begingroup
1259   \let\do\@makeother\dospecials
1260   \catcode`\{=1
1261   \catcode`\}=2
1262   \catcode`\^^I=\active
1263   \@ifnextchar\bgroup
1264     {\minted@inline@iii}%
1265     {\catcode`\{=12\catcode`\}=12
1266       \minted@inline@i}}
1267 \def\minted@inline@i#1{%
1268   \endgroup
1269   \def\minted@inline@ii##1#1{%
1270     \minted@inline@iii{##1}}%
1271   \begingroup
1272   \let\do\@makeother\dospecials
1273   \catcode`\^^I=\active
1274   \minted@inline@ii}
1275 \ifthenelse{\boolean{minted@draft}}{%
1276   {\newcommand{\minted@inline@iii}[1]{%
1277     \endgroup
1278     \begingroup

```

```

1279 \minted@defwhitespace@retok
1280 \everyeof{\noexpand}%
1281 \endlinechar-1\relax
1282 \let\do\@makeother\dospecials
1283 \catcode`\ =\active
1284 \catcode`\^^I=\active
1285 \xdef\minted@tmp{\scantokens{#1}}%
1286 \endgroup
1287 \let\FV@Line\minted@tmp
1288 \def\FV@SV@minted@tmp{%
1289   \FV@Gobble
1290   \expandafter\FV@ProcessLine\expandafter{\FV@Line}}%
1291 \ifthenelse{\equal{\minted@get@opt{breaklines}{false}}{true}}%
1292   {\let\FV@BeginVBox\relax
1293    \let\FV@EndVBox\relax
1294    \def\FV@BProcessLine##1{\FancyVerbFormatLine{##1}}%
1295    \BUseVerbatim{minted@tmp}}%
1296   {\BUseVerbatim{minted@tmp}}%
1297 \endgroup}}%
1298 {\newcommand{\minted@inline@iii}[1]{%
1299   \endgroup
1300   \minted@writecmdcode{#1}%
1301   \RecustomVerbatimEnvironment{Verbatim}{BVerbatim}{}}%
1302 \setcounter{minted@FancyVerbLineTemp}{\value{FancyVerbLine}}%
1303 \minted@pygmentize{\minted@lang}%
1304 \setcounter{FancyVerbLine}{\value{minted@FancyVerbLineTemp}}%
1305 \endgroup}}

```

`\mint` Highlight a small piece of verbatim code (a single line).

The draft version digs into a good deal of `fancyvrb` internals. We want to employ `\UseVerbatim`, and this requires assembling a macro equivalent to what `SaveVerbatim` would have created. Actually, this is superior to what `SaveVerbatim` would yield, because line numbering is handled correctly.

```

1306 \newrobustcmd{\mint}[2][ ]{%
1307   \begingroup
1308   \minted@configlang{#2}%
1309   \setkeys{minted@opt@cmd}{#1}%
1310   \minted@fvset
1311   \begingroup
1312   \let\do\@makeother\dospecials
1313   \catcode`\{=1
1314   \catcode`\}=2
1315   \catcode`\^^I=\active
1316   \@ifnextchar\bgroup
1317     {\mint@iii}%
1318     {\catcode`\{=12\catcode`\}=12
1319      \mint@i}}
1320 \def\mint@i#1{%

```

```

1321 \endgroup
1322 \def\mint@ii##1#1{%
1323   \mint@iii{##1}}%
1324 \begingroup
1325 \let\do\@makeother\dospecials
1326 \catcode`\^^I=\active
1327 \mint@ii}
1328 \ifthenelse{\boolean{minted@draft}}{%
1329   {\newcommand{\mint@iii}[1]{%
1330     \endgroup
1331     \begingroup
1332     \minted@defwhitespace@retok
1333     \everyeof{\noexpand}%
1334     \endlinechar-1\relax
1335     \let\do\@makeother\dospecials
1336     \catcode`\ =\active
1337     \catcode`\^^I=\active
1338     \xdef\minted@tmp{\scantokens{##1}}%
1339     \endgroup
1340     \let\FV@Line\minted@tmp
1341     \def\FV@SV@minted@tmp{%
1342       \FV@CodeLineNo=1\FV@StepLineNo
1343       \FV@Gobble
1344       \expandafter\FV@ProcessLine\expandafter{\FV@Line}}%
1345     \minted@langlinenoson
1346     \UseVerbatim{minted@tmp}%
1347     \minted@langlinenosoff
1348     \endgroup}}%
1349   {\newcommand{\mint@iii}[1]{%
1350     \endgroup
1351     \minted@writecmdcode{##1}%
1352     \minted@langlinenoson
1353     \minted@pygmentize{\minted@lang}%
1354     \minted@langlinenosoff
1355     \endgroup}}

```

`minted` Highlight a longer piece of code inside a verbatim environment.

```

1356 \ifthenelse{\boolean{minted@draft}}{%
1357   {\newenvironment{minted}[2][
1358     {\VerbatimEnvironment
1359       \minted@configlang{##2}%
1360       \setkeys{minted@opt@cmd}{##1}%
1361       \minted@fvset
1362       \minted@langlinenoson
1363       \begin{Verbatim}}%
1364     {\end{Verbatim}}%
1365     \minted@langlinenosoff}}%
1366   {\newenvironment{minted}[2][
1367     {\VerbatimEnvironment

```

```

1368     \let\FVB@VerbatimOut\minted@FVB@VerbatimOut
1369     \let\FVE@VerbatimOut\minted@FVE@VerbatimOut
1370     \minted@configlang{#2}%
1371     \setkeys{minted@opt@cmd}{#1}%
1372     \minted@fvset
1373     \begin{VerbatimOut}[codes={\catcode'\^^I=12}]{\minted@jobname.pyg}}%
1374 \end{VerbatimOut}%
1375     \minted@langlinenoson
1376     \minted@pygmentize{\minted@lang}%
1377     \minted@langlinenosoff}

```

`\inputminted` Highlight an external source file.

```

1378 \ifthenelse{\boolean{minted@draft}}%
1379   {\newcommand{\inputminted}[3][]{%
1380     \begingroup
1381     \minted@configlang{#2}%
1382     \setkeys{minted@opt@cmd}{#1}%
1383     \minted@fvset
1384     \VerbatimInput{#3}%
1385     \endgroup}}%
1386   {\newcommand{\inputminted}[3][]{%
1387     \begingroup
1388     \minted@configlang{#2}%
1389     \setkeys{minted@opt@cmd}{#1}%
1390     \minted@fvset
1391     \minted@pygmentize[#3]{#2}%
1392     \endgroup}}

```

9.11 Command shortcuts

We allow the user to define shortcuts for the highlighting commands.

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

```

1393 \newcommand{\newminted}[3][]{

```

First, we look whether a custom environment name was given as the first optional argument. If that's not the case, construct it from the language name (append "code").

```

1394   \ifthenelse{\equal{#1}{}}
1395     {\def\minted@envname{#2code}}
1396     {\def\minted@envname{#1}}

```

Now, we define two environments. The first takes no further arguments. The second, starred version, takes an extra argument that specifies option overrides.

```

1397 \newenvironment{\minted@envname}
1398   {\VerbatimEnvironment
1399     \begin{minted}[#3]{#2}}
1400   {\end{minted}}
1401 \newenvironment{\minted@envname *}[1]
1402   {\VerbatimEnvironment\begin{minted}[#3,##1]{#2}}
1403   {\end{minted}}

```

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

```

1404 \newcommand{\newmint}[3][]{

```

Same as with `\newminted`, look whether an explicit name is provided. If not, take the language name as command name.

```

1405   \ifthenelse{\equal{#1}{} }
1406     {\def\minted@shortname{#2}}
1407     {\def\minted@shortname{#1}}

```

And define the macro.

```

1408   \expandafter\newcommand\csname\minted@shortname\endcsname[2][]{
1409     \mint[#3,##1]{#2}##2}

```

`\newmintedfile` Define a new language-specific alias for `\inputminted`.

```

1410 \newcommand{\newmintedfile}[3][]{

```

Here, the default macro name (if none is provided) appends “file” to the language name.

```

1411   \ifthenelse{\equal{#1}{} }
1412     {\def\minted@shortname{#2file}}
1413     {\def\minted@shortname{#1}}

```

...and define the macro.

```

1414   \expandafter\newcommand\csname\minted@shortname\endcsname[2][]{
1415     \inputminted[#3,##1]{#2}{##2}}

```

`\newmintinline` Define an alias for `\mintinline`.

As is usual with inline commands, a little catcode trickery must be employed.

```

1416 \newcommand{\newmintinline}[3][]{%
1417   \ifthenelse{\equal{#1}{} }%
1418     {\def\minted@shortname{#2inline}}%
1419     {\def\minted@shortname{#1}}%
1420   \expandafter\newrobustcmd\csname\minted@shortname\endcsname{%
1421     \begingroup

```



```

1422     \let\do\@makeother\dospecials
1423     \catcode`\{=1
1424     \catcode`\}=2
1425     \@ifnextchar[{\endgroup\minted@inliner[#3][#2]}%
1426     {\endgroup\minted@inliner[#3][#2][]}%
1427     \def\minted@inliner[##1][##2][##3]{\mintinline[##1,##3]{##2}}%
1428 }

```

9.12 Float support

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

```

1429 \ifthenelse{\boolean{minted@newfloat}}%
1430 {\@ifundefined{minted@float@within}%
1431  {\DeclareFloatingEnvironment[fileext=lol,placement=h]{listing}}%
1432  {\def\minted@tmp#1{%
1433   \DeclareFloatingEnvironment[fileext=lol,placement=h,within=#1]{listing}}%
1434   \expandafter\minted@tmp\expandafter{\minted@float@within}}%
1435 {\@ifundefined{minted@float@within}%
1436  {\newfloat{listing}{h}{lol}}%
1437  {\newfloat{listing}{h}{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`.

```

1438 \ifminted@newfloat\else

```

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

```

1439 \newcommand{\listingcaption}{Listing}

```

The following definition should not be changed by the user.

```

1440 \floatname{listing}{\listingcaption}

```

`\listoflistingscaption` The caption that is displayed for the list of listings.

```

1441 \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.

```

1442 \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.

```
1443 \fi
```

9.13 Epilogue

Check whether LaTeX was invoked with `-shell-escape` option, set the default style, and make sure `pygmentize` exists. Checking for `pygmentize` must wait until the end of the preamble, in case it is specified via `\MintedPygmentize` (which would typically be after the package is loaded).

```
1444 \AtEndOfPackage{%
1445   \ifthenelse{\boolean{minted@draft}}{ }{%
1446     \ifnum\pdf@shellescape=1\relax\else
1447       \PackageError{minted}%
1448         {You must invoke LaTeX with the
1449           -shell-escape flag}%
1450         {Pass the -shell-escape flag to LaTeX. Refer to the minted.sty
1451           documentation for more information.}%
1452     \fi
1453     \setminted{style=default}%
1454   }%
1455 }
1456 \AtEndPreamble{%
1457   \ifthenelse{\boolean{minted@draft}}{ }{%
1458     \TestAppExists{pygmentize}%
1459     \ifAppExists\else
1460       \PackageError{minted}%
1461         {You must have 'pygmentize' installed
1462           to use this package}%
1463         {Refer to the installation instructions in the minted
1464           documentation for more information.}%
1465     \fi
1466   }%
1467 }
```

9.14 Final cleanup

Clean up temp files. What actually needs to be done depends on caching and engine.

```
1468 \AtEndDocument{
1469   \ifx\XeTeXinterchartoks\minted@undefined
1470   \else
1471     \DeleteFile[\minted@outputdir]{\minted@jobname.mintedcmd}%
1472     \DeleteFile[\minted@outputdir]{\minted@jobname.mintedmd5}%

```

```

1473 \fi
1474 \DeleteFile[\minted@outputdir]{\minted@jobname.pyg}%
1475 \DeleteFile[\minted@outputdir]{\minted@jobname.out.pyg}%
1476 }

```

10 Implementation of compatibility package

minted version 2 is designed to be completely compatible with version 1.7. All of the same options and commands still exist. As far as most users are concerned, the only difference should be the new commands and options.

However, minted 2 does require some additional packages compared to minted 1.7. More importantly, since minted 2 has almost completely new internal code, user code that accessed the internals of 1.7 will generally not work with 2.0, at least not without some modification. For these reasons, a copy of minted 1.7 is supplied as the package `minted1`. This is intended *only* for compatibility cases when using the current version is too inconvenient.

The code in `minted1` is an exact copy of minted version 1.7, except for two things: (1) the package has been renamed, and (2) code has been added that allows `minted1` to act as (impersonate) `minted`, so that it can cooperate with other packages that require `minted` to be loaded.⁸ When `minted1` is used, it must be loaded *before* any other packages that would require `minted`.

All modifications to the original `minted` 1.7 source are indicated with comments. All original code that has been replaced has been commented out rather than deleted. Any future modifications of `minted1` should *only* be for the purpose of allowing it to serve better as a drop-in compatibility substitute for the current release of `minted`.

```

1 \NeedsTeXFormat{LaTeX2e}
2 %%%% Begin minted1 modification
3 %%\ProvidesPackage{minted}[2011/09/17 v1.7 Yet another Pygments shim for LaTeX]
4 \ProvidesPackage{minted1}[2015/01/31 v1.0 minted 1.7 compatibility package]
5 %%%% End minted1 modification
6 \RequirePackage{keyval}
7 \RequirePackage{fancyvrb}
8 \RequirePackage{xcolor}
9 \RequirePackage{float}
10 \RequirePackage{ifthen}
11 %%%% Begin minted1 modification
12 \newboolean{mintedone@mintedloaded}
13 \@ifpackageloaded{minted}%
14 {\setboolean{mintedone@mintedloaded}{true}}%
15 \PackageError{minted1}{The package "minted1" may not be loaded after

```

⁸The approach used for doing this is described at <http://tex.stackexchange.com/a/39418/10742>.

```

16     ^^J"minted" has already been loaded--load "minted1" only for "minted"
17     ^^Jversion 1.7 compatibility}%
18     {Load "minted1" only when "minted" version 1.7 compatibility is required}}%
19     {}
20 \ifmintedone@mintedloaded\else
21 \@namedef{ver@minted.sty}{2011/09/17 v1.7 Yet another Pygments shim for LaTeX}
22 \expandafter\let\expandafter\minted@tmp\csname opt@minted1.sty\endcsname
23 \expandafter\let\csname opt@minted.sty\endcsname\minted@tmp
24 \let\minted@tmp\relax
25 %%% End minted1 modification
26 \RequirePackage{calc}
27 \RequirePackage{ifplatform}
28 \DeclareOption{chapter}{\def\minted@float@within{chapter}}
29 \DeclareOption{section}{\def\minted@float@within{section}}
30 \ProcessOptions\relax
31 \ifwindows
32   \providecommand\DeleteFile[1]{\immediate\write18{del #1}}
33 \else
34   \providecommand\DeleteFile[1]{\immediate\write18{rm #1}}
35 \fi
36 \newboolean{AppExists}
37 \newcommand\TestAppExists[1]{
38   \ifwindows
39     \DeleteFile{\jobname.aex}
40     \immediate\write18{for \string^\@percentchar i in (#1.exe #1.bat #1.cmd)
41       do set >\jobname.aex <nul: /p x=\string^\@percentchar \string~$PATH:i>>\jobnar
42     \newread\@appexistsfile
43     \immediate\openin\@appexistsfile\jobname.aex
44     \expandafter\def\expandafter\@tmp@cr\expandafter{\the\endlinechar}
45     \endlinechar=-1\relax
46     \readline\@appexistsfile to \@apppathifexists
47     \endlinechar=\@tmp@cr
48     \ifthenelse{\equal{\@apppathifexists}{}}
49       {\AppExistsfalse}
50       {\AppExiststrue}
51     \immediate\closein\@appexistsfile
52     \DeleteFile{\jobname.aex}
53 \immediate\typeout{file deleted}
54   \else
55     \immediate\write18{which #1 && touch \jobname.aex}
56     \IfFileExists{\jobname.aex}
57       {\AppExiststrue
58         \DeleteFile{\jobname.aex}}
59       {\AppExistsfalse}
60   \fi}
61 \newcommand\minted@resetoptions{}
62 \newcommand\minted@defopt[1]{
63   \expandafter\def\expandafter\minted@resetoptions\expandafter{%
64     \minted@resetoptions
65     \@namedef{minted@opt@#1}{}}

```

```

66 \newcommand\minted@opt[1]{
67   \expandafter\detokenize%
68     \expandafter\expandafter\expandafter{\csname minted@opt@#1\endcsname}}
69 \newcommand\minted@define@opt[3][]{
70   \minted@defopt{#2}
71   \ifthenelse{\equal{#1}{}}{
72     \define@key{minted@opt}{#2}{\@namedef{minted@opt@#2}{#3}}
73     {\define@key{minted@opt}{#2}[#1]{\@namedef{minted@opt@#2}{#3}}}
74 \newcommand\minted@define@switch[3][]{
75   \minted@defopt{#2}
76   \define@booleankey{minted@opt}{#2}
77     {\@namedef{minted@opt@#2}{#3}}
78     {\@namedef{minted@opt@#2}{#1}}
79 \minted@defopt{extra}
80 \newcommand\minted@define@extra[1]{
81   \define@key{minted@opt}{#1}{
82     \expandafter\def\expandafter\minted@opt@extra\expandafter{%
83       \minted@opt@extra,#1=#1}}
84 \newcommand\minted@define@extra@switch[1]{
85   \define@booleankey{minted@opt}{#1}
86     {\expandafter\def\expandafter\minted@opt@extra\expandafter{%
87       \minted@opt@extra,#1}}
88     {\expandafter\def\expandafter\minted@opt@extra\expandafter{%
89       \minted@opt@extra,#1=false}}
90 \minted@define@switch{texcl}{-P texcomments}
91 \minted@define@switch{mathescape}{-P mathescape}
92 \minted@define@switch{linenos}{-P linenos}
93 \minted@define@switch{startinline}{-P startinline}
94 \minted@define@switch[-P funcnamehighlighting=False]%
95   {funcnamehighlighting}{-P funcnamehighlighting}
96 \minted@define@opt{gobble}{-F gobble:n=#1}
97 \minted@define@opt{bgcolor}{#1}
98 \minted@define@extra{frame}
99 \minted@define@extra{framesep}
100 \minted@define@extra{framerule}
101 \minted@define@extra{rulecolor}
102 \minted@define@extra{numbersep}
103 \minted@define@extra{firstnumber}
104 \minted@define@extra{stepnumber}
105 \minted@define@extra{firstline}
106 \minted@define@extra{lastline}
107 \minted@define@extra{baselinestretch}
108 \minted@define@extra{xleftmargin}
109 \minted@define@extra{xrightmargin}
110 \minted@define@extra{fillcolor}
111 \minted@define@extra{tabsize}
112 \minted@define@extra{fontfamily}
113 \minted@define@extra{fontsize}
114 \minted@define@extra{fontshape}
115 \minted@define@extra{fontseries}

```

```

116 \minted@define@extra{formatcom}
117 \minted@define@extra{label}
118 \minted@define@extra@switch{numberblanklines}
119 \minted@define@extra@switch{showspaces}
120 \minted@define@extra@switch{resetmargins}
121 \minted@define@extra@switch{samepage}
122 \minted@define@extra@switch{showtabs}
123 \minted@define@extra@switch{obeytabs}
124 \newsavebox{\minted@bgbox}
125 \newenvironment{minted@colorbg}[1]{
126   \def\minted@bgcol{#1}
127   \noindent
128   \begin{lrbox}{\minted@bgbox}
129   \begin{minipage}{\linewidth-2\fbboxsep}
130   {\end{minipage}
131   \end{lrbox}}%
132   \colorbox{\minted@bgcol}{\usebox{\minted@bgbox}}
133 \newwrite\minted@code
134 \newcommand\minted@savecode[1]{
135   \immediate\openout\minted@code\jobname.pyg
136   \immediate\write\minted@code{#1}
137   \immediate\closeout\minted@code}
138 \newcommand\minted@pygmentize[2][\jobname.pyg]{
139   \def\minted@cmd{pygmentize -l #2 -f latex -F tokenmerge
140     \minted@opt{gobble} \minted@opt{texcl} \minted@opt{mathescape}
141     \minted@opt{startinline} \minted@opt{funcnamehighlighting}
142     \minted@opt{linenos} -P "verboptions=\minted@opt{extra}"
143     -o \jobname.out.pyg #1}
144   \immediate\write18{\minted@cmd}
145   % For debugging, uncomment:
146   %\immediate\typeout{\minted@cmd}
147   \ifthenelse{\equal{\minted@opt@bgcolor}{}}{
148     {}
149     {\begin{minted@colorbg}{\minted@opt@bgcolor}}
150     \input{\jobname.out.pyg}
151     \ifthenelse{\equal{\minted@opt@bgcolor}{}}{
152       {}
153       {\end{minted@colorbg}}
154     \DeleteFile{\jobname.out.pyg}}
155 \newcommand\minted@usedefaultstyle{\usemintedstyle{default}}
156 \newcommand\usemintedstyle[1]{
157   \renewcommand\minted@usedefaultstyle{
158     \immediate\write18{pygmentize -S #1 -f latex > \jobname.pyg}
159     \input{\jobname.pyg}}
160 \newcommand\mint[3][]{
161   \DefineShortVerb{#3}
162   \minted@resetoptions
163   \setkeys{minted@opt}{#1}
164   \SaveVerb[aftersave={
165     \UndefineShortVerb{#3}

```

```

166     \minted@savecode{\FV@SV@minted@verb}
167     \minted@pygmentize{#2}
168     \DeleteFile{\jobname.pyg}}{\minted@verb}#3}
169 \newcommand\minted@proglang[1]{}
170 \newenvironment{minted}[2]{}
171 {\VerbatimEnvironment
172  \renewcommand{\minted@proglang}[1]{#2}
173  \minted@resetoptions
174  \setkeys{minted@opt}{#1}
175  \begin{VerbatimOut}[codes=\catcode\^^I=12]{\jobname.pyg}}%
176 {\end{VerbatimOut}
177  \minted@pygmentize{\minted@proglang{}}
178  \DeleteFile{\jobname.pyg}}
179 \newcommand\inputminted[3][]{
180  \minted@resetoptions
181  \setkeys{minted@opt}{#1}
182  \minted@pygmentize[#3]{#2}}
183 \newcommand\newminted[3][]{
184  \ifthenelse{\equal{#1}{} }
185    {\def\minted@envname{#2code}}
186    {\def\minted@envname{#1}}
187  \newenvironment{\minted@envname
188   {\VerbatimEnvironment\begin{minted}[#3]{#2}}
189   {\end{minted}}}
190  \newenvironment{\minted@envname *}[1]
191   {\VerbatimEnvironment\begin{minted}[#3,#\#1]{#2}}
192   {\end{minted}}}}
193 \newcommand\newmint[3][]{
194  \ifthenelse{\equal{#1}{} }
195    {\def\minted@shortname{#2}}
196    {\def\minted@shortname{#1}}
197  \expandafter\newcommand\csname\minted@shortname\endcsname[2][]{
198   \mint[#3,#\#1]{#2}##2}}
199 \newcommand\newmintedfile[3][]{
200  \ifthenelse{\equal{#1}{} }
201    {\def\minted@shortname{#2file}}
202    {\def\minted@shortname{#1}}
203  \expandafter\newcommand\csname\minted@shortname\endcsname[2][]{
204   \inputminted[#3,#\#1]{#2}##2}}
205 \@ifundefined{minted@float@within}
206  {\newfloat{listing}{h}{lol}}
207  {\newfloat{listing}{h}{lol}[\minted@float@within]}
208 \newcommand\listingscaption{Listing}
209 \floatname{listing}{\listingscaption}
210 \newcommand\listoflistingscaption{List of listings}
211 \providecommand\listoflistings{\listof{listing}{\listoflistingscaption}}
212 \AtBeginDocument{
213  \minted@usedefaultstyle}
214 \AtEndOfPackage{
215  \ifnum\pdf@shellescape=1\relax\else

```

```
216 \PackageError{minted}  
217   {You must invoke LaTeX with the  
218   -shell-escape flag}  
219   {Pass the -shell-escape flag to LaTeX. Refer to the minted.sty  
220   documentation for more information.}\fi  
221 \TestAppExists{pygmentize}  
222 \ifAppExists\else  
223   \PackageError{minted}  
224     {You must have 'pygmentize' installed  
225     to use this package}  
226     {Refer to the installation instructions in the minted  
227     documentation for more information.}  
228   \fi}  
229 %%%% Begin minted1 modification  
230 \fi  
231 %%%% End minted1 modification
```