The minted package: Highlighted source code in LaTeX*

Konrad Rudolph
konrad_rudolph@madrat.net

2010/01/27

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.

Contents

1 Introduction

minted is a package that allows formatting source code in LaTeX. For example:

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

will highlight a piece of code in a chosen language. The display can be customized by a number of arguments and colour schemes.

*This document corresponds to minted v1.6, last changed 2010/01/27.

Unlike some other packages, most notably listings, minted requires the installation of an additional software, Pygments. This may seem like a disadvantage but there are advantages, as well:

Pygments provides far 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 token kind 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:

```ruby
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 to Pygments.

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

2 Installation

Pygments is written in Python so make sure that at least Python 2.6 is installed on you system:

```
$ python --version
Python 2.6.2
```

If that's not the case, you can download it from the website or use your operating system's package manager.

Next, install setuptools which facilitates the distribution of Python applications.

You can then install Pygments using the following simple command:

```
$ sudo easy_install Pygments
```

(If you've already got Pygments installed, be advised that minted requires at least version 1.2.)

2.1 Windows

Windows support is sketchy at the moment. There are two complications: installation and usage.

Installation The above setting assumes that `easy_install` is in a path that Windows automatically find. to do this, you must usually set your `PATH` environment variable accordingly (e.g. to `C:\Python26\Scripts`).

Usage Pygments currently does not ship with a Windows compatible application. In order to still run it, you need to create a small command script and put it someplace where Windows will find it (e.g. the aforementioned `Scripts` directory, which you will have registered in the `PATH` variable anyway). The script needs to be called `pygmentize.cmd` and it needs to contain the following content:

```
@echo off
set PYTHONPATH=C:\Python26
%PYTHONPATH%\python.exe %PYTHONPATH%Scripts\pygmentize %*
```

3 Basic usage

3.1 Preliminary

Since minted makes calls to the outside world (i.e. Pygments), you need to tell the LaTeX 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`.

3.2 Formatting source code

`minted` Using minted is straightforward. For example, to highlight a 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 one-line source codes, you can alternatively use a shorthand notation similar to \verb:

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

The complete syntax is \mint[⟨*options*⟩]{⟨*language*⟩}/⟨*code*⟩/ Where the code delimither /, like with \verb, can be almost any punctuation character. Again, this command supports a number of options described below.

\inputminted Finally, there's the comment \inputminted command to read and format whole files. Its syntax is \inputminted[⟨*options*⟩]{⟨*language*⟩}{⟨*filename*⟩}.

3.3 Using different styles

\usemintedstyle Instead of using the default style you may choose an another stylesheet provided by Pygments by its name. For example, this document uses the "trac" style. To do this, put the following into the prelude of your document:

```
\usemintedstyle{name}
```

To get a list of all available stylesheets, execute the following command on the command line:

```
$ pygmentize -L styles
```

Creating own styles is also very easy. Just follow the instructions provided on the website.

3.4 Supported languages

Pygments at the moment supports over 150 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
```

4 Floated listings

listing minted provides the listing environment to wrap around a source code block. That way, the source code will be put 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):

```
\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 into the document:

```
\listoflistings
```

List of listings

1 Example of a listing. 5

`\listingscaption` The string "Listing" in a listing's caption can be changed. To do this, simply redefine the macro `\listingscaption`, e.g.:

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

`\listoflistingscaption` Likewise, the caption of the listings list, "List of listings" can be changed by redefining `\listoflistingscaption` like so:

```
\renewcommand\listoflistingscaption{List of program codes}
```

5 Options

5.1 Usage

All minted highlight 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 ∑_{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:

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

versus

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

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

versus

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

5.2 Available options

Following is a full list of available options. For more detailed option descriptions please refer to the fancyvrb documentation, except where noted otherwise.

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

bgcolor (string): Background color of the listing (default: *none*). Notice that the

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

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

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

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

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

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

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

fontsize (font size): The size of the font to use (default: `auto` – the same as the current font).

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

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

frame (`none`|`leftline`|`topline`|`bottomline`|`lines`|`single`): : The type of frame to put around the source code listing (default: `none`).

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

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

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

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

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

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

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

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

mathescape (boolean): Enable LaTeX math mode inside comments (default: `false`). Do *not* use spaces inside math mode – they will be rendered like other full-width verbatim spaces. Usage as in package listings.

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

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

obeytabs (boolean): Treat tabs as tabs instead of converting them to spaces (default: `false`).

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

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

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

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

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

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

tabsize (integer): The number of spaces a tab is equivalent to if `obeytabs` is not active (default: `8`).

texcl (boolean): Enables LaTeX code inside comments (default: `false`). Usage as in package listings.

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

xrightmargin (dimension): Indentation to add after the listing (default: `0`).

6 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.

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}
                                        my␣$foo␣=␣$bar;
\perl/my $foo = $bar;/
```

7 To do list

- Allow multiple stylesheets in one file.
- Allow quotes in fancyvrb arguments.

8 Known issues

- Extended characters do not work inside the `minted` environment, even in conjunction with package inputenc. **Solution**: Use `xelatex` instead of plain LaTeX.

9 Implementation

9.1 System-specific settings

Since we communicate with the "outside world", some operations must be defined system-dependently.

\DeleteFile Delete a file; we're careful in case someone has already defined this macro elsewhere.

```
\ifwindows
  \providecommand\DeleteFile[1]{\immediate\write18{del #1}}
\else
  \providecommand\DeleteFile[1]{\immediate\write18{rm #1}}
\fi
```

\TestAppExists Check whether a given application exists on the system. Usage is a bit roundabout (should be fixed?) – to test whether an application exists, use the following code:

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

```
\newboolean{AppExists}
\providecommand\TestAppExists[1]{
  \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.

```
    \DeleteFile{\jobname.aex}
    \immediate\write18{for \string^\@percentchar i in (#1.exe #1.bat #1.cmd)
      do set >\jobname.aex <nul: /p x=\string^\@percentchar \string~$PATH:i>>\jobn
    \newread\@appexistsfile
    \immediate\openin\@appexistsfile\jobname.aex
    \expandafter\def\expandafter\@tmp@cr\expandafter{\the\endlinechar}
    \endlinechar=-1\relax
    \readline\@appexistsfile to \@apppathifexists
    \endlinechar=\@tmp@cr
    \ifthenelse{\equal{\@apppathifexists}{}}
     {\AppExistsfalse}
     {\AppExiststrue}
    \immediate\closein\@appexistsfile
    \DeleteFile{\jobname.aex}
\immediate\typeout{file deleted}
  \else
```

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

```
    \immediate\write18{which -s #1 && touch \jobname.aex}
    \IfFileExists{\jobname.aex}
     {\AppExiststrue
      \DeleteFile{\jobname.aex}}
     {\AppExistsfalse}
  \fi}
```

9.2 Option processing

`\minted@resetoptions` Reset options.

```
\newcommand\minted@resetoptions{}
```

`\minted@defopt` Define an option internally and register it with in the `\minted@resetoptions` command.

```
\newcommand\minted@defopt[1]{
  \expandafter\def\expandafter\minted@resetoptions\expandafter{%
    \minted@resetoptions
    \@namedef{minted@opt@#1}{}}}
```

`\minted@opt` Actually use (i.e. read) an option value. Options are passed to `\detokenize` so that `\immediate\write18` will work properly.

```
\newcommand\minted@opt[1]{
  \expandafter\detokenize%
    \expandafter\expandafter\expandafter{\csname minted@opt@#1\endcsname}}
```

`\minted@define@opt` Define a generic option with an optional default argument. If a `key` option is specified without `=value`, the default is assumed.

```
\newcommand\minted@define@opt[3][]{
  \minted@defopt{#2}
  \ifthenelse{\equal{#1}{}}{
    \define@key{minted@opt}{#2}{\@namedef{minted@opt@#2}{#3}}}
   {\define@key{minted@opt}{#2}[#1]{\@namedef{minted@opt@#2}{#3}}}}
```

`\minted@define@switch` Define an option switch (values are either `true` or `false`, and `true` may be omitted, e.g. `foobar` is the same as `foobar=true`).

```
\newcommand\minted@define@switch[2]{
  \minted@defopt{#1}
  \define@booleankey{minted@opt}{#1}{
    \@namedef{minted@opt@#1}{#2}}
   {\@namedef{minted@opt@#1}{}}}
```

`\minted@define@extra` Extra options are passed on to fancyvrb.

```
\minted@defopt{extra}
\newcommand\minted@define@extra[1]{
  \define@key{minted@opt}{#1}{
    \expandafter\def\expandafter\minted@opt@extra\expandafter{%
      \minted@opt@extra,#1=##1}}}
```

`inted@define@extra@switch` Extra switch options are also passed on to fancyvrb.

```
\newcommand\minted@define@extra@switch[1]{
  \define@booleankey{minted@opt}{#1}
```

```
    {\expandafter\def\expandafter\minted@opt@extra\expandafter{%
       \minted@opt@extra,#1}}
    {\expandafter\def\expandafter\minted@opt@extra\expandafter{%
       \minted@opt@extra,#1=false}}}
```

Actual option definitions.

```
\minted@define@switch{texcl}{-P texcomments}
\minted@define@switch{mathescape}{-P mathescape}
\minted@define@switch{linenos}{-P linenos}
\minted@define@opt{gobble}{-F gobble:n=#1}
\minted@define@opt{bgcolor}{#1}
\minted@define@extra{frame}
\minted@define@extra{framesep}
\minted@define@extra{framerule}
\minted@define@extra{rulecolor}
\minted@define@extra{numbersep}
\minted@define@extra{firstnumber}
\minted@define@extra{stepnumber}
\minted@define@extra{firstline}
\minted@define@extra{lastline}
\minted@define@extra{baselinestretch}
\minted@define@extra{xleftmargin}
\minted@define@extra{xrightmargin}
\minted@define@extra{fillcolor}
\minted@define@extra{tabsize}
\minted@define@extra{fontfamily}
\minted@define@extra{fontsize}
\minted@define@extra{fontshape}
\minted@define@extra{fontseries}
\minted@define@extra{formatcom}
\minted@define@extra@switch{numberblanklines}
\minted@define@extra@switch{showspaces}
\minted@define@extra@switch{resetmargins}
\minted@define@extra@switch{samepage}
\minted@define@extra@switch{showtabs}
\minted@define@extra@switch{obeytabs}
```

9.3 Internal helpers

`\minted@bgbox` Here, we define an environment that may be wrapped around a minted code to assign a background color.

First, we need to define a new save box.

```
\newsavebox{\minted@bgbox}
```

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

```
\newenvironment{minted@colorbg}[1]{
%\setlength{\fboxsep}{-\fboxrule}
  \def\minted@bgcol{#1}
```

```
  \noindent
  \begin{lrbox}{\minted@bgbox}
  \begin{minipage}{\linewidth-2\fboxsep}}
 {\end{minipage}
  \end{lrbox}%
  \colorbox{\minted@bgcol}{\usebox{\minted@bgbox}}}
```

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

```
\newwrite\minted@code
\newcommand\minted@savecode[1]{
  \immediate\openout\minted@code\jobname.pyg
  \immediate\write\minted@code{#1}
  \immediate\closeout\minted@code}
```

`\minted@pygmentize` Pygmentize the file given as first argument (default: `\jobname.pyg`) using the options provided.

```
\newcommand\minted@pygmentize[2][\jobname.pyg]{
  \def\minted@cmd{pygmentize -l #2 -f latex -F tokenmerge
    \minted@opt{gobble} \minted@opt{texcl} \minted@opt{mathescape}
    \minted@opt{linenos} -P "verboptions=\minted@opt{extra}"
    -o \jobname.out.pyg #1}
  \immediate\write18{\minted@cmd}
  \ifthenelse{\equal{\minted@opt@bgcolor}{}}
   {}
   {\begin{minted@colorbg}{\minted@opt@bgcolor}}
  \input{\jobname.out.pyg}
  \ifthenelse{\equal{\minted@opt@bgcolor}{}}
   {}
   {\end{minted@colorbg}}
  \DeleteFile{\jobname.out.pyg}}
```

`\minted@usedefaultstyle` Include the default stylesheet.

```
\newcommand\minted@usedefaultstyle{\usemintedstyle{default}}
```

9.4 Public API

`\usemintedstyle` Include stylesheet.

```
\newcommand\usemintedstyle[1]{
  \renewcommand\minted@usedefaultstyle{}
  \immediate\write18{pygmentize -S #1 -f latex > \jobname.pyg}
  \input{\jobname.pyg}}
```

`\mint` Highlight a small piece of verbatim code.

```
\newcommand\mint[3][]{
  \DefineShortVerb{#3}
  \minted@resetoptions
```

```
  \setkeys{minted@opt}{#1}
  \SaveVerb[aftersave={
    \UndefineShortVerb{#3}
    \minted@savecode{\FV@SV@minted@verb}
    \minted@pygmentize{#2}
    \DeleteFile{\jobname.pyg}}]{minted@verb}#3}
```

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

```
\newcommand\minted@proglang[1]{}
\newenvironment{minted}[2][]
 {\VerbatimEnvironment
  \renewcommand{\minted@proglang}[1]{#2}
  \minted@resetoptions
  \setkeys{minted@opt}{#1}
  \begin{VerbatimOut}{\jobname.pyg}}%
 {\end{VerbatimOut}
  \minted@pygmentize{\minted@proglang{}}
  \DeleteFile{\jobname.pyg}}
```

`\inputminted` Highlight an external source file.

```
\newcommand\inputminted[3][]{
  \minted@resetoptions
  \setkeys{minted@opt}{#1}
  \minted@pygmentize[#3]{#2}}
```

9.5 Command shortcuts

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

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

```
\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`").

```
  \ifthenelse{\equal{#1}{}}
   {\def\minted@envname{#2code}}
   {\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.

```
  \newenvironment{\minted@envname}
   {\VerbatimEnvironment\begin{minted}[#3]{#2}}
   {\end{minted}}
  \newenvironment{\minted@envname *}[1]
   {\VerbatimEnvironment\begin{minted}[#3,##1]{#2}}
   {\end{minted}}}
```

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

```
\newcommand\newmint[3][]{
```

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

```
  \ifthenelse{\equal{#1}{}}
   {\def\minted@shortname{#2}}
   {\def\minted@shortname{#1}}
```

And define the macro.

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

\newmintedfile Finally, define a new language-specific alias for \inputminted.

```
\newcommand\newmintedfile[3][]{
```

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

```
  \ifthenelse{\equal{#1}{}}
   {\def\minted@shortname{#2file}}
   {\def\minted@shortname{#1}}
```

. . . and define the macro.

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

9.6 Float support

listing Defines a new floating environment to use for floated listings.

```
\newfloat{listing}{h}{lol}
```

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

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

The following definition should not be changed by the user.

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

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

```
\newcommand\listoflistingscaption{List of listings}
```

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

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

9.7 Epilogue

Load default stylesheet – but only if user has not yet loaded a custom stylesheet in the preamble.

```
\AtBeginDocument{
  \minted@usedefaultstyle}
```

Check whether LaTeX was invoked with `-shell-escape` option.

```
\AtEndOfPackage{
  \ifeof18
    \PackageError{minted}
     {You must invoke LaTeX with the
      -shell-escape flag}
     {Pass the -shell-escape flag to LaTeX. Refer to the minted.sty
      documentation for more information.}\fi}
```

Check whether pygmentize is installed.

```
  \TestAppExists{pygmentize}
  \ifAppExists\else
    \PackageError{minted}
     {You must have 'pygmentize' installed
      to use this package}
     {Refer to the installation instructions in the minted
      documentation for more information.}
  \fi
```

Change History