The texments package*

Marek Kubica
marek@xivilization.net

December 27, 2008

1 Introduction

texments is a package that is used to connect the not-yet award winning source code colorizer library Pygments (`http://pygments.org/`) with LaTeX documents. That means that it is neccessary to have Pygments installed and that the `pygmentize` command is available.

2 Usage

This package does not provide many fancy features, it is just meant as an easy bridge between the highlighter and LaTeX. It calls internally the `pygmentize` command, so you need to start it with `-shell-escape` activated. `-shell-escape` allows the LaTeX processor to call arbitary shell commands which can be a security problem if compiling untrusted source, but unfortunately this is unavoidable because `pygmentize` is an external command.

Using texments is simple: you only need to add `\usepackage{texments}` to your document and can already use the texments commands described below.

A typical compilation of a document that uses texments looks like this:

```
$ pdflatex -shell-escape yourdocument.tex
```

Pygments supports many languages, to see which languages your version of Pygments supports, tell `pygmentize` to list all supported languages use the command:

```
$ pygmentize -L lexer
```

it will return a rather long list with the names of the supported languages. Pygments is in constant development, so expact this list to grow over time.

Note that if you use the texments commands to highlight text inside of beamer presentations, you need to mark the frames with the code "fragile", like this:

`\begin{frame}[fragile] ... \end{frame}`

So in case you get strange errors, try setting the containers "fragile".

`\usestyle{`⟨*stylename*⟩`}`

*This document corresponds to texments 0.2.0, dated 2008/12/27.

This macro needs to be called at least once to set the style that pygments should use to highlight all code. Internally it calls `pygmentize` to get all the color-codes that the style uses. Every language can be highlighted using every style so the style choice is purely for aesthetical reasons.

One quite useful style is the "bw" style which is, as the name somehow implies, black and white and therefore useful for printing documents that can only be black and white like books that should be published (black and white is usually cheaper) or printing on laser printers so that the printer does not have to guess which shade of grey to use.

To change the style, `usestyle` can be called more than once, so that the following source codes get highlighted with a different style.

For this document, we'll use the `default` style only.

`\pygment{`⟨*language*⟩`}{`⟨*code*⟩`}`

For simple one-line snippets the `pygment` command can be used. It will be rendered using the current selected style. Note that it will be put inside of a `Verbatim` environment.

An example for this is the following Python code which displays the typical beginner program, with fancy coloring.

```
print "Hello World"
```

This command won't be used very often, because most of the time it is more useful to use the provided environment to directly include bigger chunks of source code.

`\includecode[`⟨*language*⟩`]{`⟨*filename*⟩`}`

Includes an external file that will be highlighted according to the syntax rules of the language. It has an optional argument, the language which can be set or not. If the argument is not set, Pygments tries to guess the language by looking at the file extension. The language specified can be, as always, any language that Pygments supports.

Let's try this with a new file, called "texments.py" that we'll create. It will be a one-liner, and should print "Hello TeXmented world".

First we specify the language explicitly, using `\includecode[python]texments.py`:

```
print "Hello TeXmented world"
```

As you see this worked rather fine. Now, as the file has the proper ".py" extension, we can also leave out the language and write `\includecodetexments.py`:

```
print "Hello TeXmented world"
```

which again, renders the code correctly.

`pygmented{`⟨*language*⟩`}`

For longer snippets of code it is usually better to use the `pygmented` environment. The parameter is the language that the snippet is in, so it can be highlighted properly. Everything inside the environment will be highlighted according to the syntax of the selected language.

The first and only parameter describes the language that the snippet is in. It highlights the code to the specified rules.

```
class FancyColoredStuff(object):
    def __init__(self, language):
        self.language = language

    def highlight(self):
        print "This is colorful code in %s" % self.language

colorthing = FancyColoredStuff("Python")
colorthing.highlight()
```

Of course Python is not the only supported language, how about something more exotic? Let's try some Scheme code:

```
(define frequencies
  (foldl (lambda (word hash)
           (let ((current-value (hash-ref hash word 0)))
             (hash-set hash word (add1 current-value))))
         (make-immutable-hash '())
         ; feed in the separated words
         (string-tokenize (port->string (current-input-port)))))
```

3 Implementation

`\usestyle` This macro calls `pygmentize` with the appropriate style name and gets the color definitions that will be used in all subsequent code listings.

```
\newcommand{\usestyle}[1]{
  \immediate\write18{pygmentize -S #1 -f latex > \jobname.pyg}
  \input{\jobname.pyg}
}
```

`\pygment` This macro takes the code in the second argument and feeds it to `pygmentize`, so that it outputs code highlighted to the rules of the first argument.

```

\newwrite\code
\newcommand{\pygment}[2]{
  \immediate\openout\code\jobname.pyg
  \immediate\write\code{#2}
  \immediate\closeout\code

  \immediate\write18{pygmentize -l #1 -f latex -o \jobname.out.pyg \jobname.pyg}
  \input{\jobname.out.pyg}
}
```

`\includecode` This macro has two parameters: language and file name. The language can be omitted in which case it will be set to `auto` and causes `pygmentize` to guess the language. The highlighted source is being written in an external file and imported automatically into the main document.

```
\newcommand{\lexercommand}[1]{}

\newcommand{\includecode}[2][auto]{
  \ifthenelse{\equal{#1}{auto}}
    {\renewcommand{\lexercommand}[1]{}}
    {\renewcommand{\lexercommand}[1]{-l #1}}
  \immediate\write18{pygmentize \lexercommand{} -f latex -o #2.out.pyg #2}
  \input{#2.out.pyg}
}
```

`pygmented` An environment that highlights the code that is contained inside using the currently selected style. It uses `VerbatimOut` from the fancyvrb package to write the contents into a file, calls `pygmentize` on it and imports the highlighted source.

```
\newcommand{\proglang}[1]{}

\newenvironment{pygmented}[1]%
  {\VerbatimEnvironment
   \renewcommand{\proglang}[1]{#1}
   \begin{VerbatimOut}{\jobname.pyg}}%
  {\end{VerbatimOut}
   \immediate\write18{pygmentize -l \proglang{} -f latex -o \jobname.out.pyg \jobname.pyg}
   \input{\jobname.out.pyg}}
```