The jvlisting package

Jochen Voss
voss@seehuhn.de
http://seehuhn.de/

2011/06/27 v0.1

Abstract

This package provides the LaTeX environment `listing`, an alternative to the built-in `verbatim` environment. The `listing` environment is specially taylored for including listings of computer program source code into documents. The main advantages over the original `verbatim` environment are that `listing` environments automatically fix leading white-space so that the environment and program listing can be indented with the rest of the LaTeX source code and that `listing` environments can easily be customised and extended.

Contents

1 Basic Usage

The `listing` environment allows to include source code of computer programs in LaTeX documents by including the code between `\begin{listing}` and `\end{listing}`.

Example. In order to typeset the listing of a simple Python function (computing the cumulative sum of a list) you can use the following code in your LaTeX document.

```
\begin{listing}
  def cumsum(iterable):
      return reduce(lambda res, x: res+[res[-1]+x], iterable, [0])[1:]
\end{listing}
```

These commands result in the following output:

```
def cumsum(iterable):
    return reduce(lambda res, x: res+[res[-1]+x], iterable, [0])[1:]
```

Differently from the LaTeX `verbatim` environment, the `listing` environment can be indented to match the surrounding document source. The following will work as expected:

```
\begin{figure}[h]
  \begin{listing}
    def cumsum(iterable):
        return reduce(lambda res, x: res+[res[-1]+x], iterable, [0])[1:]
  \end{listing}
\end{figure}
```

The Python code will be typeset using the same indentation as in the example above, despite the fact that they have different indentation in the LaTeX source code.

To make it easier to copy real program source code directly into your LaTeX file, the code in listings can be indented by TAB characters instead of spaces. The TAB spacing is assumed to be 8 character columns.

In addition to the LaTeX environment described above, there is also a command `\filelisting` to typeset source code contained in a file. The command takes one argument, the name of the file to include, and behaves very similar to the the `verbatim` environment.

Example. In order to typeset the contents of a file `cumsum.py`, we could use the following command.

```
\filelisting{cumsum.py}
```

There are various parameters to customise how listings are typeset. These parameters apply both to the `verbatim` environment and to the `\filelisting` command.

- The amount of space inserted before and after a listing is given by `\listingskipamount`. The default value is `1ex`. Commands like the following can be used to adjust the amount of space.

  ```
  \setlength{\listingskipamount}{1\baselineskip}
  ```

- The indentation of the left margin of the typeset code is determined by `\listingindent`. The default value is `2em`. Commands like the following can be used to adjust indentation.

  ```
  \setlength{\listingindent}{2cm}
  ```

- The font used in the listing is controlled by the macro `\listingfont`. The default value is `\normallistingfont` which sets up a typewriter-like font. Example: The following command can be used to obtain more compact listings by slightly reducing the fontsize and the line spacing.

  ```
  \renewcommand{\listingfont}%
    {\normallistingfont\small\renewcommand{\baselinestretch}{0.95}}
  ```

- The penalty for page breaks inside a listing is given by `\listingpenalty`. The higher this value, the less attractive page breaks inside the listing are to LaTeX's page breaking algorithm. The default value is 500. The following command can be used to completely disable page breaks inside listings.

  ```
  \listingpenalty=10000
  ```

- The penalty for page breaks just before or after a listing is given by `\prelistingpenalty` (default value 100) and `\postlistingpenalty` (default value -50), respectively.

2 Extending the listing Environment

New `listing`-like environments can be defined using the `\NewListingEnvironment` macro. This macro takes six arguments and uses them to define a new LaTeX environment. The arguments, in order, are

1. The name of the new environment.
2. The number of extra arguments for the new environment (normally 0). These extra arguments of the environment, if any, are substituted into the initialisation commands (*i.e.* into the fourth argument of `\NewListingEnvironment`); see the `copylisting` environment, below, for an example.
3. Commands to execute before the environment is entered (*e.g.* to add vertical whitespace).
4. Initialisation commands, executed inside the environment (*e.g.* font/margin setup).
5. The name of a macro (receiving one argument) which will be used to typeset each line of input.
6. Commands to execute after the environment is completed.

Example. The `listing` environment provided by the `jvlisting` package is defined using the following commands:

```
\let\listingfont=\normallistingfont
\newcommand{\ListingTypesetLine}[1]{\noindent\hskip\listingindent\strut
  #1\par\penalty\listingpenalty}
```

```
\newcommand{\prelistingskip}{\endgraf\ifdim\lastskip<\listingskipamount
  \removelastskip\penalty\prelistingpenalty\vskip\listingskipamount\fi}
\newcommand{\postlistingskip}{\endgraf\penalty\postlistingpenalty
  \vskip\listingskipamount}
\NewListingEnvironment{listing}{0}{\prelistingskip}%
  {\listingfont}{\ListingTypesetLine}{\postlistingskip}
```

Example. An `nlisting` environment which generates listings with additional line numbers can be defined as follows:

```
\newcounter{lineno}
\newcommand{\typesetnline}[1]{\addtocounter{lineno}{1}%
  \noindent\hskip\listingindent\llap{{\it\scriptsize\arabic{lineno}: }}%
  \strut #1\par\penalty\listingpenalty}
\NewListingEnvironment{nlisting}{0}{\prelistingskip}%
  {\setcounter{lineno}{0}\listingfont}{\typesetnline}{\postlistingskip}
```

Example. The following code defines a new `copylisting` environment which does not only typeset the source code, but also saves a copy in an external file.

```
\newwrite\outfile
\newcommand{\copytypeset}[1]{\immediate\write\outfile{#1}%
  \ListingTypesetLine{#1}}
\NewListingEnvironment{copylisting}{1}{\prelistingskip}%
  {\immediate\openout\outfile=#1\listingfont}{\copytypeset}%
  {\immediate\closeout\outfile\postlistingskip}
```

Here we used the second argument to `\NewListingEnvironment` to indicate that the `copylisting` environment should take an additional parameter (the output file name). The new environment is used as follows:

```
\begin{copylisting}{listing1.c}
  #include <stdio.h>

  int
  main()
  {
    puts("hello, world!");
    return 0;
  }
\end{copylisting}
```

3 Extending the **filelisting** Command

New `filelisting`-like commands can be defined using the `\NewFileListingCommand` macro. This macro takes six arguments and uses them to define a new macro. The arguments, in order, are

1. The name of the new command including the leading backslash.
2. The number of arguments for the new command (normally 1). These arguments, including the first one which gives the file name, are substituted into the third, fourth and sixth argument of `\NewFileListingCommand`); see the `\prefixfilelisting` command, below, for an example.
3. Commands to execute before the listing is started (*e.g.* to add vertical whitespace).
4. Initialisation commands, executed inside the scope of the listing (*e.g.* font/margin setup).
5. The name of a macro (receiving one argument) which will be used to typeset each line of input.
6. Commands to execute after the listing is completed.

The first argument of the newly defined macro always denotes the name of the file to include.

Example. Using the auxiliary functions for the `listing` environment, the built-in `filelisting` command can be defined as follows:

```
\NewFileListingCommand{\filelisting}{1}{\prelistingskip}%
  {\listingfont}{\ListingTypesetLine}{\postlistingskip}
```

Example. The following command defines a macro to read a file and to prefix every line of the resulting listing with a given string.

```
\newcommand{\pfxtypeset}[1]{\noindent\hskip\listingindent\strut
  \pfx#1\par\penalty\listingpenalty}
\NewFileListingCommand{\prefixfilelisting}{2}{\prelistingskip}%
  {\listingfont\def\pfx{#2}}{\pfxtypeset}{\postlistingskip}
```

4 Implementation

This section describes the internal implementation of the `jvlisting` package. In order to avoid name clashes with other packages, the names of all internal macros defined in this package start with the prefix `jvl@`. The following is the preamble for the package file.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{jvlisting}[2011/06/27 v0.1 Formatted Program Listings]
```

4.1 Processing the Lines of Input

We start by provinding a macro `\jvl@iterlines` which can be used to iterate over all lines of input until a line containing some marker is found. The marker is given as the argument `#1` to the `\jvl@iterlines` macro. Each line is processed by appending the token `\jvl@eol` (used later to recognise the end of line) and by prepending `\jvl@dropempty` (for the first and last line) or `\jvl@countspaces` (for all other lines).

```
\def\jvl@iterlines#1{\obeylines
  \expandafter\jvl@iterla\expandafter{#1}{\jvl@dropempty}}
{\obeylines
\gdef\jvl@iterla#1#2#3
  {\def\jvl@testmarker##1#1{}%
  \expandafter\def\expandafter\w\expandafter{\jvl@testmarker#3#1}%
  \ifx\w\empty%
    \def\next{#2#3\jvl@eol\jvl@iterla{#1}{\jvl@countspaces}}%
  \else%
    \def\y##1#1##2#1{\jvl@dropempty ##1\jvl@eol\jvl@end##2}%
    \def\next{\y#3#1}%
  \fi\next}}
```

The next step in processing is to drop the first line (whatever comes directly after the opening `begin` statement) and the last line (whatever comes directly after the closing `end` statement), if they consist only of white space.

```
\def\jvl@dropempty{\jvl@dropa{}}
\def\jvl@dropa#1#2{\ifx\jvl@eol#2%
    \let\next=\relax
  \else
    \if#2 \def\next{\jvl@dropa{#1#2}}\else
      \def\next{\jvl@countspaces #1#2}\fi
  \fi\next}
```

Next we determine the indentation level of the current line by expanding TAB characters and then counting spaces. The result is stored in the scratch counter `@tempcnta`.

```
\def\jvl@countspaces{\@tempcnta=0\jvl@counta}
{\catcode`\^^I=12
```

```
\gdef\jvl@counta#1{\expandafter\if\noexpand#1^^I%
    \advance\@tempcnta by8\divide\@tempcnta by8\multiply\@tempcnta by8
    \let\next=\jvl@counta
  \else
    \expandafter\if\noexpand#1 %
      \advance\@tempcnta by1
      \let\next=\jvl@counta
    \else
      \def\next{\jvl@fixspaces #1}
    \fi
  \fi\next}}
```

Using the value in `@tempcnta`, we fix the indentation by first subtracting the common indentation level (stored in `\jvl@idt`) and then inserting the required number of spaces (using `\space`).

```
\newcount\jvl@idt \jvl@idt=-1
\def\jvl@fixspaces#1{\ifx\jvl@eol#1\else
  \ifnum\jvl@idt<0
    \jvl@idt=\@tempcnta
  \else
    \ifnum\@tempcnta<\jvl@idt\jvl@idt=\@tempcnta\fi
  \fi\fi\jvl@fixa#1}
\def\jvl@fixa{\ifnum\@tempcnta>\jvl@idt
    \advance\@tempcnta by\m@ne
    \def\next{\jvl@fixa\space}%
  \else
    \let\next\jvl@output
  \fi\next}
```

Finally, by matching the `jvl@eol` inserted by the `\jvl@iterlines` function above, we can apply the output function `\jvl@typeset` to the processed line.

```
\def\jvl@output#1\jvl@eol{\jvl@typeset{#1}}
```

The symbol `\jvl@typeset` will be defined inside the `\begin{listing}` command; since neested listings are not possible, using a global name for the output function is no problem.

4.2 Defining Listing Environments

Given the code above, we can now define the `\NewListingEnvironment` macro. Some care is needed when constructing the marker for use in the `\jvl@iterlines` method, because the text we need to match uses category code 12 ("other") for all characters.

```
\begingroup
  \catcode`|=0 \catcode`[=1 \catcode`]=2
  \catcode`\{=12 \catcode`\}=12 \catcode`\\=12
  |gdef|jvl@makemarker#1[%
    |expandafter|gdef|csname jvl@@#1marker|endcsname[\end{#1}]]
|endgroup
\def\jvl@setup{\begingroup
  \parskip0pt \advance\leftskip by\@totalleftmargin
  \let\do\@makeother\dospecials \catcode`\^^I=12}
\def\NewListingEnvironment#1#2#3#4#5#6{\jvl@makemarker{#1}%
  \expandafter\newcommand\csname #1\endcsname[#2]{#3\jvl@setup
    #4\def\jvl@end{\end{#1}}\let\jvl@typeset=#5%
    \expandafter\jvl@iterlines\csname jvl@@#1marker\endcsname}%
  \expandafter\gdef\csname end#1\endcsname{\endgroup #6}}
```

4.3 Reading Listings from File

When reading listings from a file, we get the lines terminated by `^^M` characters. The following function is a replacement for `\jvl@iterlines`, used to read lines from a file and to strip of the

trailing ^^M characters:

```
\newread\jvl@fileinput
\def\jvl@iterfile{\read\jvl@fileinput to\l
  \ifeof\jvl@fileinput
    \let\next\relax
  \else
    \expandafter\jvl@iterfa\l
    \let\next\jvl@iterfile
  \fi\next}
{\catcode`\^^M=12
\gdef\jvl@iterfa#1^^M{\jvl@countspaces#1\jvl@eol}}
```

In analogy to \NewListingEnvironment, the following macro is used to define new functions for listing file contents.

```
\def\NewFileListingCommand#1#2#3#4#5#6{%
  \ifnum#2<1
    \PackageError{jvlisting}{%
      Invalid number of arguments: `#2'
    }{%
      The second argument to \protect\NewFileListingCommand\space must be at
      least 1.
    }%
  \fi
  \newcommand{#1}[#2]{#3\jvl@setup\catcode`\^^M=12
    #4\let\jvl@typeset=#5%
    \openin\jvl@fileinput=##1\jvl@iterfile\closein\jvl@fileinput
    \endgroup #6
  }
}
```

4.4 Finishing Touches

The following definitions provide default values for the customisable parameters of the listing environment.

```
\newskip\listingskipamount \listingskipamount=1ex
\newdimen\listingindent \listingindent=2em
\newcount\prelistingpenalty \prelistingpenalty=100
\newcount\listingpenalty \listingpenalty=500
\newcount\postlistingpenalty \postlistingpenalty=-50
```

Finally, we need to define the font for use in listing environments. Some care is needed to avoid problems with spaces and ligatures.

```
{\catcode`\ =\active%
\gdef\jvl@obeyspaces{\frenchspacing\catcode`\ =\active\let \space}}
{\catcode`\`=\active\gdef`{\relax\lq}}
\gdef\jvl@noligs{\catcode`\`=\active}
\def\normallistingfont{\normalfont\ttfamily
  \jvl@obeyspaces\jvl@noligs\hyphenchar\font-1}
```