

The `jvlisting` package

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

2011/11/17 v0.7

Abstract

This package provides the \LaTeX environment `listing`, an alternative to the built-in `verbatim` environment. The `listing` environment is specially tailored 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 document source and that `listing` environments can easily be customised and extended.

Contents

1 Basic Usage	1
2 Extending the <code>listing</code> Environment	2
3 Extending the <code>filelisting</code> Command	3
4 Implementation	4
4.1 Processing the Lines of Input	4
4.2 Defining Listing Environments	6
4.3 Reading Listings from File	6
4.4 Finishing Touches	7

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 previous example, despite the fact that the code in the two `listing` environments has 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 `listing` 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 `listing` 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 font size 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 white space).
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]{\ifvmode\penalty\listingpenalty\noindent\fi
\hskip\listingindent\strut#1\par}
\newcommand{\prelistingskip}{\endgraf\ifdim\lastskip>\listingskipamount\else
\remove\lastskip\penalty\prelistingpenalty\vskip\listingskipamount\fi}
\newcommand{\postlistingskip}{\endgraf\penalty\postlistingpenalty
\vskip-\parskip\nobreak\vskip\listingskipamount\noindent}
\NewListingEnvironment{listing}{0}{\prelistingskip}%
{\listingfont\let\ListingStartHook\listingstarthook}{\ListingTypesetLine}%
{\postlistingskip\ignorespacesafterend}
```

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. Here we need to be careful in case the quote character ‘ can be present in the listing: by default, `\listingfont` makes ‘ an active character in order to prevent problems with ligatures, but this fix causes damage to the output of the `\write` command. To work around this problem, we can use `\DisableLigatureFix` (defined by the `jvlisting` package for just this purpose) to temporarily disable the ligature fix.

```
\newwrite\outfile
\newcommand{\copytypeset}[1]{\ListingTypesetLine{#1}%
{\DisableLigatureFix\immediate\write\outfile{#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 white space).
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 `jv1@`. The following is the preamble for the package file.

```

\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{jvlisting}[2011/11/17 v0.7 Formatted Program Listings]

```

4.1 Processing the Lines of Input

We start by providing a macro `\jv1@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 argument `#1` to the `\jv1@iterlines` macro. Each line is processed by prepending `\jv1@dropempty` (for the first and last line) or `\jv1@tryhook` (for all other lines).

```

\def\jv1@iterlines#1{\expandafter\jv1@iterla\expandafter{#1}{\jv1@dropempty}}
{\obeylines\gdef\jv1@iterla#1#2#3
{\def\jv1@testmarker##1#1{}}%
\expandafter\def\expandafter\w\expandafter{\jv1@testmarker#3#1}%
\ifx\w\empty%
\def\next{#2{\jv1@iterla{#1}{\jv1@tryhook}}#3}

```

```

    }%
  \else%
    % special treatment for the last line
    \def\y##1##2#1{\jvl@dropempty{\jvl@end##2}##1
    }%
    \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 before the closing `end` statement), if these lines consist only of white space.

```

\def\jvl@dropempty#1{\jvl@dropa{#1}{}}
{\obeylines\gdef\jvl@dropa#1#2#3{\ifx
#3\def\next{#1}%
\else%
\if#3 \def\next{\jvl@dropa{#1}{#2#3}}\else%
\def\next{\jvl@tryhook{#1}#2#3}\fi%
\fi\next}}

```

Next we try to apply the `\ListingStartHook`.

```

\def\jvl@tryhook#1{\ifx\ListingStartHook\undefined
\def\next{\jvl@countspaces{#1}}%
\else
\def\jvl@trya{\let\ListingStartHook\undefined\jvl@countspaces{#1}}%
\def\next{\ListingStartHook{\jvl@trya}}%
\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#1{\@tempcnta=0\jvl@counta{#1}}
{\catcode'\^^I=12
\gdef\jvl@counta#1#2{\if^^I#2%
\advance\@tempcnta by8\divide\@tempcnta by8\multiply\@tempcnta by8
\def\next{\jvl@counta{#1}}%
\else
\expandafter\if\noexpand#2 %
\advance\@tempcnta by1
\def\next{\jvl@counta{#1}}%
\else
\def\next{\jvl@fixspaces{#1}#2}%
\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=\m@ne
{\obeylines\gdef\jvl@fixspaces#1#2{\ifx
#2\else%
\ifnum\jvl@idt<0%
\jvl@idt=\@tempcnta%
\else%
\ifnum\@tempcnta<\jvl@idt\jvl@idt=\@tempcnta\fi%
\fi\fi\jvl@fixa{#1}#2}}
\def\jvl@fixa#1{\ifnum\@tempcnta>\jvl@idt
\advance\@tempcnta by\m@ne
\def\next{\jvl@fixa{#1}\space}%
\else
\def\next{\jvl@output{#1}}%
\fi\next}

```

Finally, we apply the output function `\jvl@typeset` to the processed line and starting processing of the next line.

```
{\obeylines\gdef\jvl@output#1#2
  {\jvl@typeset{#2}#1}}
```

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

4.2 Defining Listing Environments

In order to allow for verbatim printing, the macro `jvl@fixparsing` switches off the special meaning of most characters. Some care is needed to avoid problems with spaces and hyphenation.

```
{\catcode'\ =\active%
\gdef\jvl@obeyspaces{\frenchspacing\catcode'\ =\active\let \space}}
\def\jvl@fixparsing{\let\do\@makeother\dospecials\catcode'\^I=12
  \jvl@obeyspaces\hyphenchar\font\m@ne}
```

Since the `\hyphenchar` setting is global, we save the value of `hyphenchar` before entering the listing environment and restore the original setting at the end.

```
\newcount\jvl@tmphyphenchar
\def\jvl@begingroup{\jvl@tmphyphenchar=\hyphenchar\font
  \begingroup \parskip0pt \advance\leftskip by\@totalleftmargin}
\def\jvl@endgroup{\endgroup \hyphenchar\font=\jvl@tmphyphenchar}
```

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` macro, because the text we need to match uses category code 12 (“other”) for the characters `\`, `{` and `}`.

```
\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\NewListingEnvironment#1#2#3#4#5#6{\jvl@makemarker{#1}%
  \expandafter\newcommand\csname #1\endcsname[#2]{#3\jvl@begingroup
  #4\def\jvl@end{\end{#1}}\let\jvl@typeset=#5%
  \jvl@fixparsing\obeylines
  \expandafter\def\expandafter\jvl@start\expandafter{%
    \expandafter\jvl@iterlines\csname jvl@#1marker\endcsname}%
  \let\next\jvl@start
  \next}%
  \expandafter\gdef\csname end#1\endcsname{\jvl@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 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@iterfb{#1}}}
```

```
{\obeylines\gdef\jvl@iterfb#1{\jvl@tryhook{\relax}#1
}}
```

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@begingroup
    #4\let\jvl@typeset=#5%
    \jvl@fixparsing\catcode'\^M=12
    \openin\jvl@fileinput=#1\jvl@iterfile\closein\jvl@fileinput
    \jvl@endgroup#6%
    \ignorespaces
  }%
}
```

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 define the default font for use in listings. Some care is needed to avoid problems with ligatures like ‘. We break such ligatures by making ‘ and active character. For use in file listings we also provide `\DisableLigatureFix` to (temporarily) disable this fix.

```
{\catcode'\‘=\active\gdef{\relax\lq}%
\gdef\DisableLigatureFix{\def{\lq}}
\def\normallistingfont{\normalfont\ttfamily\catcode'\‘=\active}
```