listings-ext — A collection of LaTeX commands and some helper files to support the automatic integration of parts of source files into a documentation*

Jobst Hoffmann
Fachhochschule Aachen, Standort Jülich
Ginsterweg 1
52428 Jülich
Bundesrepublik Deutschland
email: j.hoffmann@fh-aachen.de

printed on August 31, 2009

Abstract

This article describes the use and the implementation of the LaTeX package listings-ext.sty, a package to simplify the insertion of parts of source code files into a document(ation).

Contents

1 Preface

This package is intended as an implementation of the macros which are described in [Lin07]. If a software developer wants to document a piece of software, in most cases she/he doesn't need to print out whole source code files, but only parts of the files. This can be achieved by the listings-package [HM07] and especially by the command

`\lstinputlisting[linerange={`⟨*first1*⟩`-`⟨*last1*⟩`}, ...]{`⟨*filename*⟩`}`

*This file has version number v48, was last revised on 2009/08/31, the documentation dates from 09/08/27.

In [Lin07] there are described three macros, which can be created automatically, so that in the case of changes in the source code the developer mustn't change the contents of the lineranges, but she/he only has to rerun a program, which regenerates the meaning of the macros. In the following the three macros and a Bash-script to deal with these three macros are provided.

2 Hints For Installation And Maintenance

The package listings-ext.sty is belonging to consists of altogether four files:

`README`,
`listings-ext.ins`,
`listings-ext.dtx`,
`listings-ext.pdf`.

In order to generate on the one hand the documentation of the package and on the other hand the corresponding .sty-file and the supporting Bash-script one has to proceed as follows:

First the file listings-ext.ins must be formatted e.g. by

```
latex listings-ext.ins
```

This formatting run generates several other files. These are first of all the previous mentioned .sty-file `listings-ext.sty`, the style file which can be used by `\usepackage{listings-ext}` and the Bash-script `listings-ext.sh`. Then there are the files `listings-ext.bib` and `hyperref.cfg`, which are needed to produce this documentation. The creation of the documentation is is simplified by the files `listings-ext.makemake` and `listings-ext.mk`, which can be used to make a `Makefile` (see section **??**). Another helper file is `listings-ext.el`, which can be used with the (X)Emacs editor (see section **??**). Some simple tests of this package can be done by using the files `listings-ext*exmpl*` and `listings-ext*test*`, which are also created; these files use the configuration file `listings.cfg`, which can also be used as a base or supplement for own configuration files of the listings-package. This file can be put into the current directory for local use or into the `TEXMFHOME` directory for use with every document. You have to take care about the fact, that the local file in the current directory prevent any `listings.cfg` from being loaded. Finally there is a file `getversion.tex`, which is used for the creation of a "versioned" distribution.

The common procedure to produce the documentation is to execute the commands

```
latex listings-ext.dtx
bibtex listings-ext
latex listings-ext.dtx
makeindex -s gind.ist listings-ext.idx
makeindex -s gglo.ist -o listings-ext.gls -t listings-ext.glg \
        listings-ext.glo
latex listings-ext.dtx
```

The result of this formatting run is the documentation in form of a .dvi-file, that can be manipulated the normal way. It ain't possible to use `pdflatex` because of the integrated PostScript based figures.

One can find further informations about working with the integrated documentation in [Mit06] and [MDB+05].[1]

This documentation itself is generated by the following internal driver code:

```
\documentclass[a4paper, ngerman, english]{ltxdoc}

\usepackage[T1]{fontenc}
\usepackage{lmodern}
\usepackage{babel,babelbib}
\usepackage[svgnames]{pstricks}

\usepackage{listings-ext}
\GetFileInfo{listings-ext.sty}

\newif\ifcolor \IfFileExists{color.sty}{\colortrue}{}
\ifcolor \RequirePackage{color}\fi

\usepackage[numbered]{hypdoc}
\usepackage{url}


%\EnableCrossrefs
%\DisableCrossrefs    % say \DisableCrossrefs if index is ready
%\RecordChanges       % gather update information
%\CodelineIndex       % index entry code by line number
\OnlyDescription    % comment out for implementation details
\MakeShortVerb{\|}   % |\foo| acts like \verb+\foo+

\begin{document}

    \DocInput{listings-ext.dtx}%
\end{document}
```

3 The User's Interface

3.1 Preparing the LaTeX code

The user's interface is as simple as possible: just load the required package by saying

`\usepackage[`style=⟨*style-name*⟩`]{listings-ext}`

⟨*style-name*⟩ is the name of a listings style, defined by the command [HM07, sec. 4.5]. You can find examples for such styles in the exemplary configuration file `listings.cfg`.

`\lstdefinestyle{`⟨*style name*⟩`}{`⟨*key=value list*⟩`}`

After loading the package provides three commands:

1. `\lstdef{`⟨*identifier*⟩`}{`⟨*file-name*⟩`}{`⟨*range*⟩`}`

   defines the ⟨*identifier*⟩, by which the line range ⟨*range*⟩ in the file defined by ⟨*file-name*⟩ can be referenced. If you identify several ⟨*filename*⟩s or ⟨*range*⟩s

[1] Generating the documentation is much easier with the `make` utility, see section **??**.

by the same ⟨*identifier*⟩, the last definition is valid. If you don't like that behaviour, put the corresponding `\lstdef`- and `\lstuse`-commands (see below) into a group of its own.

2. `\lstuse[`⟨*options*⟩`]{`⟨*identifier*⟩`}`

   includes the source code which is referenced by ⟨*identifier*⟩ by (internally) calling `\lstinputlisting` of the package listings [HM07], the way of formatting can be influenced by ⟨*options*⟩.

3. `\lstcheck{`⟨*identifier*⟩`}{`⟨*file-name*⟩`}`

   can be used, if one prepares a file ⟨*file-name*⟩ consisting of a lot of `\lstdef` commands. If ⟨*identifier*⟩ isn't yet defined, the file defined by ⟨*file-name*⟩ is `\input`. This command is especially helpful, if you prepare a presentation and you want to format only single slides for testing their look.

3.2 Preparing the source code

If you just want to include a small part of a source code file, you can do that without touching the source: just write the corresponding commands `\lstdef` and `\lstuse` into your LaTeX-code. But if the source code changes, you have to adapt the changes in the `\lstdef` command. That may be very tedious, if are you changing your sources often.

It' better to automate that procedure, and one way of implementig that is done at the Bash-script `listings-ext.sh`. For working with that script you have to tag the parts of the source, which you want to document, by comments.

At the moment there are three tags, which can be described by the following regular expressions:

1. `^\ +`⟨*endline-comment-character(s)*⟩`\ be:\ `⟨*string*⟩`$`

   This expression defines the beginning of the environment, which should be `\lstinput` into the document.

2. `^\ +`⟨*endline-comment-character(s)*⟩`\ ee:\ `⟨*string*⟩`$`

   This expression defines the end of that environment.

3. `^\ +`⟨*endline-comment-character(s)*⟩`\ ce:\ `⟨*list of keywords*⟩

   This expression defines, how the the environments defined by the above introduced should be processed.

The meaning of the regular expressions is: start a line with at least one blank space "␣", add endline comment characters (C++ and Java: `//`, Fortran: `!`) followed by another blank space. Then you have to enter "`be:␣`" for the beginning of a code environment, and "`ee:␣`" for the end. In both cases the line must be ended with a string which should denote the meaning of the environment , the strings for the beginning and the end must be identical.[2]

If you have prepared a source code file with these tags, you can process it by the Bash-script provided by the package. The simplest way to do it is the call

`listings-ext.sh -c -o` ⟨*file list*⟩

[2] You can also use the standard C comments `/*` ... `*/`, but in that case the trailing "`*/`" is seen as the end of the ⟨*string*⟩.

⟨*file list*⟩ is a list of one or more file names. This call creates the file ⟨*directory*⟩`.lst`, where ⟨*directory*⟩ is the name of the current directory. The file consists of a header and a list of `\lstdef` commands, of the form

`\lstdef{`⟨*identifier*⟩`}{`⟨*filename*⟩`}{`⟨*line range(s)*⟩`}`

You can `\input` the file ⟨*directory*⟩`.lst` into your documentation; its header looks like (for example)

```
%% -- file listings-ext.lst generated on Mi 26. Aug 22:05:20 CEST 2009
        by listings-ext.sh
\csname listingsextLstLoaded\endcsname
\let\listingsextLstLoaded\endinput
```

The first line is wrapped by hand, the second and third line prohibit a second load of that file. One of the `\lstdef` could look like

```
\lstdef{listingsExtExmplAA}{/home/xxx%
      /listings-ext%
      /listings-ext_exmpl_a.java}{3-5}
```

You can input this file in two ways into your document:

1. by saying

   ```
   \input{listings-ext.lst}
   ```

   at the beginning of a file or

2. by saying

   ```
   \lstcheck{listingsExtExmplAA}{listings-ext.lst}
   ```

   in an environment to keep the definition local.

After that you can use the command `\lstuse` to integrate the source code parts into your documentation. The usage is

`\lstuse[`⟨*options*⟩`]{`⟨*identifier*⟩`}`

at the place, where you want the part of your source code.

The ⟨*identifier*⟩ is generated automatically, it is derived from the file name, you have to transfer the identifiers from the .lst-file to the `\lstuse`-command by hand, but that happens typically only one time. So in this case the `\lstuse` command could look like — as said you can add options —

```
\lstuse{listingsExtExmplAA}
```

For more information about the use of the Bash-script `listings-ext.sh` enter the command

```
listings-ext.sh -h
```

There is one optional initial tag `ce:` (*control environment*): It needs one argument ⟨*mode*⟩. The argument describes the further processing. ⟨*mode*⟩ may be one of the following values:

combine: `be: ...ee:` groups with the same description are combined into one piece of code in the output

join: all `be: ...ee:` groups (independent of the description) are combined into one piece of code in the output

The behaviour of the three modes of operation is shown in Figure 1. `ce:` has to be put before all other tags in the source code.

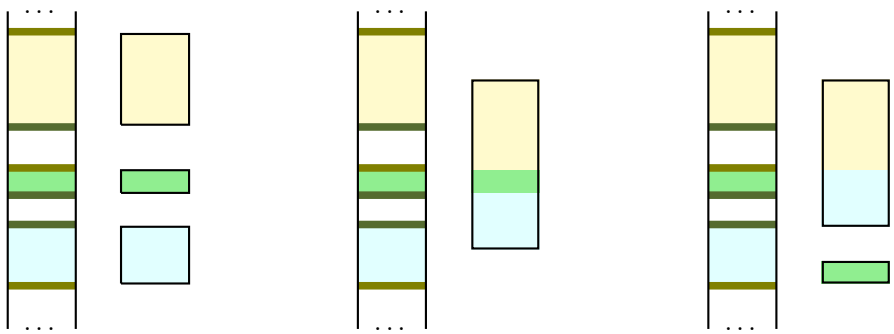

Figure 1: Modes of operation: a) no special control of the tagged parts, every part can be processed by itself, b) control by `ce: join`, all tagged parts are joint into one piece, which can be further processed, c) control by `ce: combine`, tagged parts with the same describing string are joint into one piece, which can be further processed, all other parts can be processed by their own

4 References

[HM07] Heinz, Carsten and Brooks Moses: *The listings package*, February 2007. Version 1.4. 1, 3, 4

[Lin07] Lingnau, Anselm: *LaTeX-Hacks*. O'Reilly, Beijing; Cambridge; Farnham; Köln; Paris; Sebastopol; Taipei; Tokyo, 1. Auflage, 2007. Tipps & Techniken für den professionellen Textsatz. 1, 2

[MDB+05] Mittelbach, Frank, Denys Duchier, Johannes Braams, Marcin Woliński, and Mark Wooding: *The* DocStrip *program*, July 2005. version number 2.5d. 3

[Mit06] Mittelbach, Frank: *The* doc *and* shortvrb *Packages*, February 2006. version number 2.1d. 3