Reading and executing source code

Herbert Voß

Abstract

A frequent question that arises in the various forums is whether specific regions of source code can be displayed both verbatim and with the output of its execution. The packages `fancyvrb` and `listings` support writing to external files and partial reading of source code of arbitrary type. Further packages, such as `showexpl`, allow executing parts of a LaTeX source. This article shows how to apply this to arbitrary types of code.

1 Introduction

When creating a manuscript for an article or book, the text is, depending on the subject, augmented by examples that often refer to the output created by a particular programme. It can be beneficial to control the source code for these programmes from within the document to make sure that any changes are reflected in both the source code and the output in the final document. This can avoid mistakes, especially in longer documents.

2 Simple LaTeX sequences

2.1 Areas of source code

For LaTeX examples, only the source code between `\begin{document}` and `\end{document}` is relevant. The packages `fancyvrb` and `listings` both support specifying an area by line numbers. Such numbers need to be changed, however, when lines are added to or removed from the source code. It therefore makes more sense to specify a string of characters for start and end of the area. The package `listings` provides the option `linerange`; the specification of the interval is in principle the same as specifying line numbers. Only special characters have to be escaped by prefixing them with a backslash: `\\begin\{document\}`. The option `includerangemarker=false` omits the output of the string marking the area; otherwise, the `\begin{document}` and `\end{document}` would appear in the output.

```
\lstinputlisting[
  linerange=\\begin\{document\}-\\end\{document\},
  includerangemarker=false]{demo.tex}
```

The command above yields the following source code of a LaTeX document, which will be used as an example throughout this article.

```
\begin{tabular}{@{}
  m{0.5\linewidth}@{}
  >{\lstinputlisting[
      includerangemarker=false,
      rangeprefix=\%,
      linerange=START-STOP]{\jobname.tmp}}
  m{0.5\linewidth} @{}}
\begin{Example}
\pspicture(3,2)
%START
\psframe*[linecolor=blue!30](3,2)
%STOP
\endpspicture
\end{Example}
& \tabularnewline

\begin{Example}
\pspicture(3,2)
%START
\psframe*[linecolor=red!30](3,2)
\endpspicture
%STOP
\end{Example}
& \tabularnewline
\end{tabular}
```

The same can be achieved with the package `fancyvrb`. The area can be specified through the options `firstline` and `lastline`. The following example outputs its own text body.

```
\documentclass{article}
\usepackage{fancyvrb}
\begin{document}
\VerbatimInput[frame=single,
  fontsize=\footnotesize,
  firstline=\string\begin{document},
  lastline=\string\end{document},
]{\jobname.tex}
\end{document}
```

The `firstline` and `lastline` options define macros `\FancyVerbStartString` and `\FancyVerbStopString`. In special cases, these can be manipulated directly. The macro definition must contain leading whitespace if it is present in the source code. The macros do not exist and therefore need to be defined through `\newcommand` or `\edef` if TeX-specific special characters are used, as in this case. The following example outputs the preamble of our sample document. The source document contains two spaces in front of `\begin{document}`, which have to be taken care of through `\space`.

```
\edef\FancyVerbStartString{%
  \string\documentclass{article}}
\edef\FancyVerbStopString{%
  \space\space\string\begin{document}}% 2 spaces
\VerbatimInput[frame=single,fontsize=\footnotesize]
  {demo.tex}
```

```
\makeatletter
\let\pc\@percentchar
\makeatother
\usepackage{pstricks,fancyvrb,array,listings}
\lstset{basicstyle=\ttfamily\small}

\def\endExample{\end{VerbatimOut}
```

```
  \def\START{}\def\STOP{}\input{\jobname.tmp}}
\newcommand\Example{%
\VerbatimEnvironment
\begin{VerbatimOut}{\jobname.tmp}}
```

2.2 Source code and output

Another frequent use case is the display of source code and the result of its compilation with LaTeX, as in the following example.

foo

bar

```
foo\newline\mbox{%
 \put(7,2){%
  \circle*{\strip@pt\normalbaselineskip}}}%
\newline
bar
```

The entire source code is not always of interest, as in the example above, which actually contains two additional lines.

```
\makeatletter
%START
foo\newline\mbox{%
 \put(7,2){%
  \circle*{\strip@pt\normalbaselineskip}}}%
\newline
bar
%STOP
\makeatother
```

To restrict the output to the result of compiling the actual lines, the so-called markers `%START` and `%STOP` were added to define the relevant area. They do not affect the result of the compilation as they are prefixed with the LaTeX comment character `%`. Of course the comment character should be changed according to the language being used.

Here, to typeset the source code and the output side by side a table was used. The right-hand column is explicitly left blank. The respective command was added to the column definition and only the column separator & must be specified, even if no other material appears in the table.

```
\begin{tabular}{@{}
  m{0.2\linewidth}@{}
  >{\lstinputlisting[includerangemarker=false,
      rangeprefix=\%,
      linerange=START-STOP]{\jobname.tmp}}
  m{0.8\linewidth} @{}}
\begin{Example}
\makeatletter
%START
foo \put(12,0){\circle*{\strip@pt\normalbaselineskip}}
\hspace{2\normalbaselineskip}bar
%STOP
\makeatother
\end{Example}
& \tabularnewline
\end{tabular}
```

The environment `Example` uses `fancyvrb` to write everything to a temporary file which is read immediately afterwards through `\input` and thus executed.

```
\newcommand\Example{%
\VerbatimEnvironment
\begin{VerbatimOut}{\jobname.tmp}}
\def\endExample{%
  \end{VerbatimOut}
  \input{\jobname.tmp}}
```

Instead of a table, a `minipage` could have been used to achieve the arrangement as well. In neither case, however, can page breaks occur within examples. If the output should appear below the source, a different definition must be used. In the following example, a table with normal table header and partial source code is output.

A table without using `tabularx` which is as wide as the line. This is created with this source below, which contains several line breaks.

Table 1: Example for calculated table width

foo	bar	baz
and now a somewhat longer text to show line breaks	and now a somewhat longer text to show line breaks	and now a somewhat longer text to show line breaks

```
\begin{tabular}{@{}
  >{\RaggedRight}p{1.5cm}|
  >{\Centering}p{2cm}  |
  >{\RaggedLeft}p{\linewidth-3.5cm-4\tabcolsep-0.8pt}
  @{}}\hline
foo & bar & baz\\\hline
and now a somewhat longer text to show line breaks &
and now a somewhat longer text to show line breaks &
and now a somewhat longer text to show line breaks\\
\hline
\end{tabular}
```

The source code can now be arbitrarily long as page breaks are possible. The package `fancyvrb` does not support UTF-8 characters; they remain active and would be output in their expanded form. The `inputenx` package provides a workaround, but by default non-ASCII characters have to be specified in their TeX-notation, for example `\"u`. The corresponding example environment `ExampleB` for the above example looks like the following:

```
\newcommand\ExampleB{%
\VerbatimEnvironment
\begin{VerbatimOut}{\jobname.tmp}}
\def\endExampleB{%
```

```
  \end{VerbatimOut}
  {\centering \input{\jobname.tmp}}
  \lstinputlisting[
      includerangemarker=false,
      rangeprefix=\%,
      linerange=START-STOP]{\jobname.tmp}}

\begin{ExampleB}
\begin{table}[!htb]
\hrulefill\par
A table without using \texttt{tabularx} which is as wide
as the line. This is shown by this text, which contains
several line breaks.
\caption{Example for calculated table width}
%START
\begin{tabular}{@{}
  >{\RaggedRight}p{1.5cm}|
  >{\Centering}p{2cm}  |
  >{\RaggedLeft}p{\linewidth-3.5cm-4\tabcolsep-.8pt}
  @{}}\hline
foo & bar & baz\\\hline
and now a somewhat longer text to show line breaks &
and now a somewhat longer text to show line breaks &
and now a somewhat longer text to show line breaks\\
\hline
\end{tabular}
%STOP
\end{table}
\end{ExampleB}
```

The reverse order of the output can be achieved by swapping the `\input` and `\lstinputlisting`. For outputting source code, the corresponding command `\VerbatimInput` from the package `fancyvrb` can also be used. Not using it here was an arbitrary decision.

2.3 Entire documents

To show the source code and result of entire LaTeX document or non-LaTeX code, a different approach must be taken — a simple `\input` does not work any more. A general solution would be to include the result of the execution of the source code as a figure through `\includegraphics`. If the same font is used as in the document, there will be no difference compared to using `\input`, even for pure text. To identify the externally created figures, a custom counter is defined: `\newcounter{FigureCounter}`. The files are created as `\jobname-\theFigureCounter.tex` and can easily be assigned to source code.

A `Makefile` can be used to simplify the entire procedure of creating the figures independently. After a first pdfLaTeX run, which can use the option `-draftmode` for improved speed, all files with names `\jobname-*` can be run with the respective programme through the `Makefile`. In the example below, PSTricks code is processed with XeLaTeX to be able to get PDF output. To remove any white margin from the figure, it is processed with `pdfcrop` after the XeLaTeX run. The extension of the written files can be used to identify the programme to process them with, for example, `.cpp` for a C++ example. After all the external files have been created, pdfLaTeX is run again to read the created PDF figures.

The PDF files do not exist at the time of the first pdfLaTeX run. To avoid error messages because of this, their presence is checked through `\IfFileExists`. We now have the following:

```
\newcounter{FigureCounter}
\newcommand\ExampleC{%
\refstepcounter{FigureCounter}%
\VerbatimEnvironment
\begin{VerbatimOut}{\jobname-\theFigureCounter.tex}}
\def\endExampleC{%
  \end{VerbatimOut}
  \IfFileExists{%
    \jobname-\theFigureCounter.pdf}% PDF exists?
    {\includegraphics{\jobname-\theFigureCounter.pdf}}%
    {\fbox{PDF missing!}}% no, output message
  \lstinputlisting[
    linerange=\\begin\{document\}-\\end\{document\},
    includerangemarker=false]%
       {\jobname-\theFigureCounter.tex}}
```

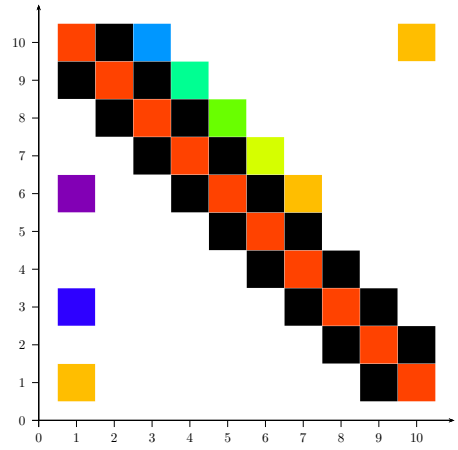
This is only suitable for LaTeX or XeLaTeX documents. The preamble and postamble typical for LaTeX are defined as the macros `\FVB@VerbatimOut` and `\FVE@VerbatimOut` from the package `fancyvrb` to avoid the user having to specify them every time.

```
\renewcommand\FVB@VerbatimOut[1]{%
  \@bsphack%
  \begingroup
  \FV@UseKeyValues%
  \FV@DefineWhiteSpace%
  \def\FV@Space{\space}%
  \FV@DefineTabOut%
  \def\FV@ProcessLine##1{%
    \toks@{##1}\immediate\write\FV@OutFile{\the\toks@}}%
  \immediate\openout\FV@OutFile #1\relax%
  \WritePSTricksPreamble%<<=== write preamble
  \let\FV@FontScanPrep\relax
  \let\@noligs\relax%
  \FV@Scan}
\renewcommand\FVE@VerbatimOut{%<<=== write postamble
   \WriteLine{\string\end{document}}% <<
   \immediate\closeout\FV@OutFile\endgroup\@esphack}
```

The macro `\WriteLine` allows us to use a specific preamble every time; in the following example, for PSTricks code. For a C++ example, a different preamble would be defined.

```
\newcommand\WriteLine[1]{%
  \begingroup%
  \let\protect\@unexpandable@protect%
  \edef\reserved@a{\immediate\write\FV@OutFile{#1}}%
  \reserved@a%
  \endgroup}
\newcommand\WritePSTricksPreamble{%
  \WriteLine{\string\documentclass{article}}%
  \WriteLine{\string\usepackage{pstricks-add}}%
  \WriteLine{\string\pagestyle{empty}}%
  \WriteLine{\string\begin{document}}%
}
```

These definitions provide the preliminaries for using the new environment `ExampleC`. The example shown here is a so-called surface plot.



```
\psscalebox{0.5}{%
\begin{pspicture}(-0.5,-0.75)(11,11)
\psaxes[ticksize=-5pt 0]{->}(11,11)
\psMatrixPlot[colorType=5,dotsize=1.1cm,
              xStep=1,yStep=1,
              dotstyle=square*]{10}{10}{matrix1.data}
\end{pspicture}}
```

To make the preamble more flexible and be able to output parts of it as code, two additional macros can be used to specify the invisible and the visible part of the preamble. Here, the invisible part is output into the external file first, but this is arbitrary. The macro `\FVB@VerbatimOut`, which was modified above already, is changed again to omit a preamble for a special case (`\WritePSTricksPreamble`), instead now including a template preamble (`\WritePreamble`). The template preamble does not load additional packages; they have to be loaded by the user. This allows for more flexibility. The package `listings` allows not only specifying a region, but also a sorted, comma-separated list in curly braces.

```
\lstinputlisting[
  linerange={\%PSTART-\%PSTOP,%
  \\begin\{document\}-\\end\{document\}},
  includerangemarker=false]{%
    \jobname-\theFigureCounter.tex}%
```

In this case, everything in the source document between `%START` and `%STOP` and also between `\begin{document}` and `\end{document}` is output. To distinguish between preamble and text body, they are output with different background colours and a small additional space between them. The part to insert the code is now

```
  \IfFileExists{\jobname-\theFigureCounter.pdf}%
    {\begin{center}\expandafter\includegraphics%
     \expandafter[\GraphicxOptions]%
       {\jobname-\theFigureCounter.pdf}
     \end{center}}%
    {\fbox{PDF missing!}}%
  \def\GraphicxOptions{}%
  \lstinputlisting[backgroundcolor=\color{black!10},
                   linerange=\%PSTART-\%PSTOP,
                   includerangemarker=false,]%
                  {\jobname-\theFigureCounter.tex}
  \lstinputlisting[backgroundcolor={},
     linerange=\\begin\{document\}-\\end\{document\},
     includerangemarker=false]%
         {\jobname-\theFigureCounter.tex}%
  \gdef\Invisible@Part{}%
  \gdef\Visible@Part{}%
```

The macro `\GraphicxOptions` saves the optional parameter of the `ExampleD` environment, which may contain key/value pairs for `\includegraphics`. The invisible part of the preamble is passed as an argument to the macro `\PreambleInvisible` and the visible part to `\PreambleVisible`. The external LaTeX document now has the following preamble.

```
\newcommand\WritePreamble{%
  \WriteLine{\string\documentclass{article}}%
  \WriteLine{\string\pagestyle{empty}}%
  \WriteLine{\Invisible@Part}
  \WriteLine{\@percentchar PSTART}
  \WriteLine{\Visible@Part}%
  \WriteLine{\@percentchar PSTOP}
  \WriteLine{\string\begin{document}}%
}
```

A page break is now possible after the figure and within the code output, as shown by the following example.

The binding energy in the liquid drop model is composed of the following parts.

- the surface part,
- the volume part,

$$E = a_v A + -a_f A^{2/3} + -a_c \frac{Z(Z-1)}{A^{1/3}} + -a_s \frac{(A-2Z)^2}{A} + E_p \qquad (1)$$

- the Coulomb part,
- the asymmetry part,
- and a pairing part.

```
\usepackage{tgpagella}
\usepackage{pst-node}
```

```
\psset{nodesep=3pt}
The binding energy in the liquid drop model is composed
of the following parts.
\begin{itemize}
\item the \rnode{b}{surface part},
\item the \rnode{a}{volume part},\\[1cm]
\def\xstrut{\vphantom{\frac{(A)^1}{(B)^1}}}
\begin{equation}
E =
\rnode[t]{ae}{\psframebox*[fillcolor=black!8,
```

```
  linestyle=none]{\xstrut a_vA}} +
\rnode[t]{be}{\psframebox*[fillcolor=black!16,
  linestyle=none]{\xstrut -a_fA^{2/3}}} +
\rnode[t]{ce}{\psframebox*[fillcolor=black!24,
  linestyle=none]{\xstrut -a_c\frac{Z(Z-1)}{A^{1/3}}}} +
\rnode[t]{de}{\psframebox*[fillcolor=black!32,
  linestyle=none]{\xstrut -a_s\frac{(A-2Z)^2}{A}}} +
\rnode[t]{ee}{\psframebox*[fillcolor=black!40,
  linestyle=none]{\xstrut E_p}}
\end{equation}\\[0.25cm]
\item the \rnode{c}{Coulomb part},
\item the \rnode{d}{asymmetry part},
\item and a \rnode{e}{pairing part}.
\end{itemize}
\nccurve[angleA=-90,angleB=90]{->}{a}{ae}
\nccurve[angleB=45]{->}{b}{be}
\nccurve[angleB=-90]{->}{c}{ce}
\nccurve[angleB=-90]{->}{d}{de}
\nccurve[angleB=-90]{->}{e}{ee}
```

The macro `\Preamble`, which saves the invisible and the visible part of the preamble, is somewhat more complex because the special characters like \, $, &, # ^, _, % and ~ and line endings must be handled separately. If an arbitrary optional argument is specified, it is assumed that it is the invisible part of the preamble.

```
\def\MakeVerbatimNewLine{^^J}
\begingroup
\catcode`\^^M=\active %
  \gdef\obeylines@Preamble{\catcode`\^^M\active
     \let^^M\MakeVerbatimNewLine}%
\endgroup

\newcommand\Preamble{%
 \par
 \begingroup
 \makeatother
 \let\do\@makeother
 \do\ \do\\\do\$\do\&\do\#\do\^\do\_\do\~\do\%
 \obeylines@Preamble
 \@ifnextchar[\PreambleInvisible@{\PreambleVisible@[]}}
\long\def\PreambleInvisible@[#1]#2{%
  \long\xdef\@gtempa{#2}%
   \endgroup\let\Invisible@Part\@gtempa}
\long\def\PreambleVisible@[#1]#2{\long\xdef\@gtempa{#2}%
   \endgroup\let\Visible@Part\@gtempa}
```

This can be used to control the output of the preamble in the example code. Only the parts which are of interest to the reader can be output while other things can be defined as well and written into the exported TEX file, but do not appear as source code in the final document. For the example above:

```
\Preamble[Invisible]{\usepackage[T1]{fontenc}
\usepackage{mathpazo}
\usepackage{pstricks}
}

\Preamble{\usepackage{tgpagella}
\usepackage{pst-node}}
```

3 Arbitrary source code type

It has already been mentioned that in principle any language can be used in the exported file. The `Makefile` can do the appropriate processing based on the extension of the file. Only the example environment must know the type of file to be exported. Our final example shows an external Perl programme, which is written from this document and executed. The output of the programme is saved with the same base name and the extension `.out`. Finally, the output is inserted back into this document as pure text.

A standardised Perl code could have the following header (preamble).

```
\newcommand\SchreibePerlPraeambel{%
  \WriteLine{\numbersign !/usr/bin/perl}%
  \WriteLine{\numbersign }%
  \WriteLine{\numbersign  Herbert Voss 20110201}%
  \WriteLine{use strict;}%
  \WriteLine{\Invisible@Part}
  \WriteLine{\numbersign PSTART}
  \WriteLine{\Visible@Part}%
  \WriteLine{\numbersign PSTOP}
  \WriteLine{\numbersign }%
  \WriteLine{\numbersign bodystart!!}%
}
```

The definition of the example environment is in principle the same as the LATEX version shown above. Instead of including a generated PDF file, the text output created by the external Perl programme is input with `\lstinputlisting`. The optional argument of the environment `ExampleE` can be used to specify the formatting; it is passed through to `\lstinputlisting`.

```
\newcommand\ExampleE[1][]{%
  \def\lstOptions{#1}%
  \refstepcounter{FigureCounter}%
  \VerbatimEnvironment
\begin{VerbatimOut}{\jobname-\theFigureCounter.pl}}
\def\endExampleE{%
 \end{VerbatimOut}
 \IfFileExists{\jobname-\theFigureCounter.out}%
  {\expandafter\lstinputlisting\expandafter[\lstOptions]%
   {\jobname-\theFigureCounter.out}}%
      {\fbox{Output missing!}}%
  \medskip
  \def\lstOptions{}%
  \lstinputlisting[backgroundcolor=\color{black!10},
                   linerange=\#PSTART-\#PSTOP,
                   includerangemarker=false,]%
                  {\jobname-\theFigureCounter.pl}
  \lstinputlisting[
    backgroundcolor={},
    linerange=\#bodystart!!-\#bodyend!!,
    includerangemarker=false]%
                  {\jobname-\theFigureCounter.pl}%
  \gdef\Invisible@Part{}%
  \gdef\Visible@Part{}%
}
```

The following example determines so-called Kaprekar constants (see `http://en.wikipedia.org/wiki/Kaprekar_constant`). These are natural numbers with the following properties: If the digits are sorted ascending and descending, the result is a largest and a smallest number whose difference is the same as the original number. The algorithm here uses a brute force approach to keep the code simple; all numbers are generated and tested.

```
Determining Kaprekar constants
1 digits:
2 digits:
3 digits: 495,
4 digits: 6174,
5 digits:
```

```
### Determining Kaprekar constants ###
my $number = 1;
my $start = 1;
my $end = 10;

print("Determining␣Kaprekar␣constants\n");
while ($number < 6) {
  print "$number␣digits:␣";
  foreach ($start...$end) {     # for each line $_
    my @chars = split(//,$_);
    my $Min = join("",sort(@chars));
    my $Max = reverse($Min);
    my $Dif=$Max-$Min;
    if($_ eq $Dif) { print $_,","; }
  }
  $number = $number+1;
  $start = $start*10;
  $end = $end*10;
  print "\n"; }
```

The output of the entire Perl programme is the same as for a LaTeX example as well.

```
\Preamble[Invisible]{# Example for a Kaprekar constant}
\Preamble{### Determining Kaprekar constants ###
my $number = 1;
my $start = 1;
my $end = 10;}

\begin{ExampleE}[basicstyle=\ttfamily\footnotesize,
                 frame=LR]
print("Determining␣Kaprekar␣constants\n");
while ($number < 6) {
  print "$number␣digits:␣";
  foreach ($start...$end) {     # for each line $_
    my @chars = split(//,$_);
    my $Min = join("",sort(@chars));
    my $Max = reverse($Min);
    my $Dif=$Max-$Min;
    if($_ eq $Dif) { print $_,","; }
  }
  $number = $number+1;
  $start = $start*10;
  $end = $end*10;
  print "\n"; }
\end{ExampleE}
```

4 Creating this document

This document creates several external example files which are then run by XeLaTeX and Perl. The created PDFs from XeLaTeX are cropped to eliminate the whitespace and then inserted as PDF graphics, while the output from the Perl program is inserted as a text file. All this is done for Linux with the following simple shell script:

```
#!/bin/sh
pdflatex voss2011         # main doc
xelatex voss2011-1.tex    # 1st created external file
pdfcrop voss2011-1        # cut whitespace
mv voss2011-1-crop.pdf voss2011-1.pdf # rename
xelatex voss2011-2.tex    # 2nd created external file
pdfcrop voss2011-2        # cut whitespace
mv voss2011-2-crop.pdf voss2011-2.pdf # rename
perl voss2011-3.pl > voss2011-3.out   # 3rd external file
pdflatex voss2011         # main doc
```

5 Summary

This article has shown how to create external source files of arbitrary types and execute them through a `Makefile` after a first LaTeX run. The file extension of the created file should designate the type of programme used for its execution. The output of the programmes can be included in subsequent LaTeX runs as figures or text. The author retains the full control over example programmes. If there is a large number of examples, the created file can be written into a temporary directory and compared with an existing file through the Unix `diff` command to avoid executing the programme again if the source code has not changed.

There are more optional parameters possible for inserting the output into the document to, for example, specify left/right alignment. Figures could be processed with `pdfcrop` to remove white margins.

References

[1] Carsten Heinz: *The `listings` package*, Version 1.4, Feb. 2007; `mirror.ctan.org/macros/latex/contrib/listings/`

[2] Rolf Niepraschk: *The `showexpl` package*, Version 0.3h, Feb. 2007; `mirror.ctan.org/macros/latex/contrib/showexpl/`

[3] Timothy Van Zandt: *The `fancyvrb` package — Fancy verbatims in LaTeX*, Version 2.8, May 2010; `mirror.ctan.org/macros/latex/contrib/fancyvrb/`

◇ Herbert Voß
Wasgenstraße 21
14129 Berlin, Germany
herbert (at) dante dot de
http://tug.org/PSTricks