Typesetting external program code and its output: hvextern

Herbert Voß

Abstract

When writing a book with a mathematical, scientific or technical background, the output of programs is often inserted as text or an illustration; in many cases, also with complete or partial indication of the respective source code. As an author, you have the problem of keeping such external sample programs in sync with the current manuscript. If you keep the source code in the book manuscript itself, and create the external examples at the same time as typesetting the main document, you can be sure the code and output stay consistent.

1 Introduction

If you use LaTeX to write a book about LaTeX, you can easily insert the output of the examples directly in the main document. [2] This does not necessarily mean that the examples will also work as small individual documents. All examples in a larger book use the main document's preamble, which is not available to a reader of the book.

It usually makes more sense to create examples as separate documents or programs that are independent of the main document. To do this, the complete source code is written from the main document into an external file, which is then processed using a specified program and the result is integrated back into the main document as a PDF, PNG, text, or whatever form is appropriate. From the entire source code of the example, you can use markers to place the essential lines of code in the main document before or next to the example output.

> The output of the following examples was generated "on-the-fly" when compiling this *TUGboat* article. Any change in the example code therefore automatically led to updated output during the next compilation process.

To begin, first we'll show a short example: *TUGboat* is normally compiled with pdfLaTeX, so there are problems with an example that absolutely requires the use of XeLaTeX. The example must be created externally and the output integrated as a PDF. It makes more sense to do this from the main document and include the output directly, as shown here: 美好的一天.

This is made possible by the `hvextern` package, which defines only one environment and one command. [5]

The corresponding code for the above inline example is:

Inline example

```
[...] as shown here:
\begin{externalDocument}[
  compiler=xelatex, inline, runs=2, force,
  grfOptions={height=8pt}, crop, cropmargin=0,
  cleanup, docType=latex]{voss}
\documentclass{ctexart}% needs xelatex
\pagestyle{empty}
\begin{document}
美好的一天.
\end{document}
\end{externalDocument}
This is made [...]
```

Any change in this example will automatically be kept in sync — during the next translation process, the output of the main document will be updated, and with it the new code of the example will be run, and thus also the new output will be inserted.

In the example above, only the output was included without showing the source code. Depending on the application, it may be desirable to display portions or all of the source code; this is described on the following pages.

Currently the `hvextern` package supports external documents for METAPOST, TeX, ConTeXt, LaTeX, LuaTeX, LuaLaTeX, XeTeX, XeLaTeX, Lua, Perl, Java, Python, and shell scripts.

2 Syntax

The package, which has no special options, is loaded as usual: `\usepackage{hvextern}`. The package defines only one environment, `{externalDocument}`, and one command, `\runExtCmd`:

Syntax

```
\begin{externalDocument}[⟨options⟩]
        {⟨output filename without extension⟩}
... source code ...
\end{externalDocument}

\runExtCmd[⟨options⟩]
    {⟨command⟩}
    {⟨output filename without extension⟩}
```

The main document must be compiled with the `-shell-escape` option (or with two dashes, as usual), otherwise no external commands will be run and thus the correct output will not be shown.

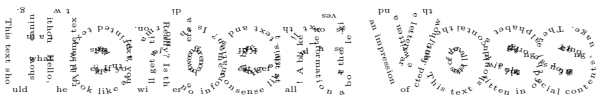
lualatex invocation

```
lualatex --shell-escape ⟨latexfile⟩
```

First published in *Die TEXnische Komödie* 2/2022, pp. 30–60. Translation by the author.

doi.org/10.47397/tb/43-3/tb135voss-extern

Let's show another example: The following code for character manipulation must be compiled using the program sequence `latex→dvips→ps2pdf`, because it does not work with other TEX engines. With the environment `externalDocument`, however, you can write the complete code in an external file, specify the necessary process, and embed the result as a PDF. The only important thing is that when creating the graphic, the standard output of the page number is suppressed and any white space is cut off using the crop option. Of course, this does not apply if a complete page is to be included (see page 284).



The code of the above example looks like:

dvips example

```
\begin{externalDocument}[compiler=latex,crop,
  force=false, cleanup={log,aux,ps,dvi},
  grfOptions={width=\linewidth}]{voss}
\documentclass{article}
\usepackage[american]{babel}
\pagestyle{empty}
\usepackage{pst-text,blindtext}
\begin{document}
\DeclareFixedFont{\SF}{T1}{phv}{b}{n}{2cm}
\pstextpath(0,-1ex){\pscharpath*[
  linestyle=none]{\SF Herbert Voss}}{%
\tiny \blindtext}
\end{document}
\end{externalDocument}
```

The meaning of each option:

compiler=latex Use LaTeX to compile. The rest of the invocation, including other programs, is determined by the internal definition of `\hv@extern@runLATEX`.

crop Crop the whitespace with `pdfcrop`.

force Recreate the output even if it already exists.

cleanup={log,aux,ps,dvi} Delete the specified auxiliary files at the end.

grfOptions={width=\linewidth} Scale output to the current linewidth.

voss Filename that is extended internally by a consecutive number.

The external filename, extended by a consecutive number, can be printed into the margin by setting the keyword `showFilename`. In general it is printed in the outer margin, or in twocolumn mode in the outer column. If the example is set in twocolumn mode but inside a starred floating environment over both columns, then use the keyword `outerFN` (see Figure 1). Then `hvextern` doesn't test for twocolumn mode.

A vertical shift of the filename is possible by specifying a length for `shiftFN`, e.g., `shiftFN=5ex`.

Essentially, it doesn't matter which programming language is used, as long as minimum communication between the main document and the external program is guaranteed: this consists only of the requirement that the external document must provide its output with the same file name with which it was called. However, even this can be a problem in some programming languages, as shown below with some examples.

By default, source code and output are displayed one above the other, so that a page break in the source code is not a problem. The following example creates and runs a Python program, and then includes the output as a PNG format file. The header of the `externalDocument` environment is:

Options for Python

```
\begin{externalDocument}[compiler=python3,
  code, ext=py, docType=py, usefancyvrb,
  grfOptions={width=\linewidth}]{voss}
... Python code ...
\end{externalDocument}
```

It is only in rare cases that you will want to output the complete source code. Therefore, areas can be defined using so-called markers, which then delimit the output. The markers are written to the external file as normal comments, the only reason why they are programming language dependent; the comment character is not uniform. For Python the markers are:

Python marker lines

```
\hv@extern@ExampleType{py}
  {\NumChar StartVisibleMain}
  {\NumChar StopVisibleMain}
  {\NumChar StartVisiblePreamble}
  {\NumChar StopVisiblePreamble}
```

and for plain TEX:

TEX marker lines

```
\hv@extern@ExampleType{tex}
  {\perCent StartBody}
  {\string\bye}
  {\perCent StartVisiblePreamble}
  {\perCent StopVisiblePreamble}
```

`\perCent` and `\NumChar` are the TEX and Python comment characters % and #, which must be escaped for LaTeX. Internally, the category of the character is changed so that it is available as a normal character using the `\perCent` or `\NumChar` command.

After calling the Python program, it must be ensured that the file name is determined in order to provide the output with the same main file name and a different file extension. In Python this is possible with the following code:

Python: get filename

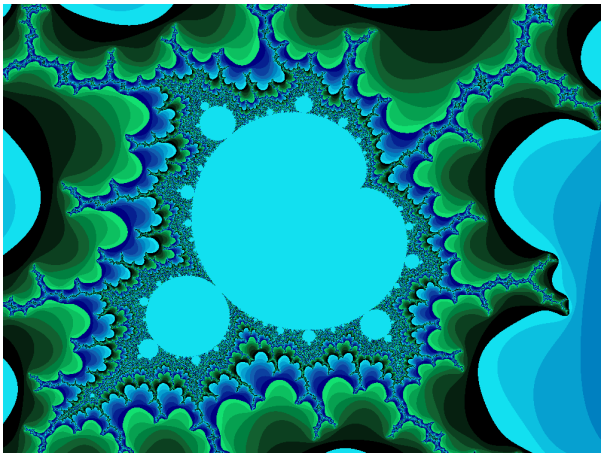
```
fileName = os.path.basename(
   os.path.splitext(__file__)[0])
```

Depending on the output format, this file name is extended by `.pdf`, `.png`, or `.txt`, so that the output can be easily inserted into the LaTeX main document. In addition, the markers are now used so that the output of parts of the Python source code can be done, requested with the `code` keyword. (Output grayscaled for printed *TUGboat*.)

voss-3.py

```
from PIL import Image
import subprocess
# drawing area (xa < xb and ya < yb)
xa = -0.1716
xb = -0.1433
ya = 1.022
yb = 1.044
maxIt = 1024  # iterations
imgx = 1000   # image size
imgy = 750
image = Image.new("RGB", (imgx, imgy))
```

```
for y in range(imgy):
    cy = y * (yb - ya) / (imgy - 1)  + ya
    for x in range(imgx):
        cx = x * (xb - xa) / (imgx - 1) + xa
        c = complex(cx, cy)
        z = 0
        for i in range(maxIt):
            if abs(z) > 2.0: break
            z = z * z + c
        r = i % 4 * 6
        g = i % 8 * 32
        b = i % 16 * 16
        image.putpixel((x, y), b*65536 + g*256 + r)
```



In a purely formal way, the output of the source code can be defined by analogy to LaTeX as a preamble (general definitions) and program body (application), whereby two slightly different background colors are used for differentiation. The markers can be used anywhere in the document. The above example was created with the following LaTeX code:

Complete example code

```
\begin{externalDocument}[compiler=python3, force=false,
  %showFilename,
  runs=1, code, ext=py, docType=py,
  usefancyvrb, grfOptions={width=\linewidth}]{python}
import os
#StartVisiblePreamble
from PIL import Image
import subprocess
# drawing area (xa < xb and ya < yb)
xa = -0.1716; xb = -0.1433
ya = 1.022;   yb = 1.044
maxIt = 1024  # iterations
imgx = 1000   # image size
imgy = 750
image = Image.new("RGB", (imgx, imgy))
#StopVisiblePreamble

#StartVisibleMain
for y in range(imgy):
    cy = y * (yb - ya) / (imgy - 1)  + ya
    for x in range(imgx):
        cx = x * (xb - xa) / (imgx - 1) + xa
        c = complex(cx, cy)
        z = 0
        for i in range(maxIt):
            if abs(z) > 2.0: break
            z = z * z + c
        r = i % 4 * 6
        g = i % 8 * 32
        b = i % 16 * 16
        image.putpixel((x, y), b*65536 + g*256 + r)
#StopVisibleMain
# now get the filename created by the latex
imageName = os.path.basename(
  os.path.splitext(__file__)[0])
image.save(imageName+".png", "PNG")
\end{externalDocument}
```

By specifying a width for the output of the source code, code and result can be arranged side by side, as shown in Figure 1.

3 Using markers in the source code

The markers identify the areas of the source code that are to be output in the (LaTeX) main document. For an external document with TeX or LaTeX code, the use of the markers are shown in the following examples:

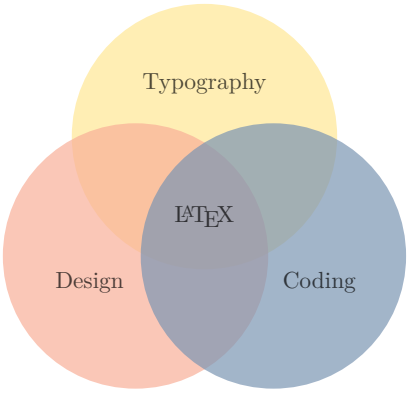
LaTeX marker lines

```
[...]
%StartVisiblePreamble
[... listed preamble code ...]
%StopVisiblePreamble
[...]
\begin{document}
[... listed body code ...]
\end{document}
```

Everything between the `%StartVisiblePreamble` and `%StopVisiblePreamble` lines is printed with the background color `BGpreamble` (default `black!12`). All of the lines between `\begin{document}` and `\end{document}`, on the other hand, are considered as the text body and printed with the background color `BGbody` (default `black!8`).

```
\usepackage{tikz}
\usepackage[hks,pantone,xcolor]{xespotcolor}
```

```
\SetPageColorSpace{HKS}
\definecolor{HYellow}{spotcolor}{HKS05N,0.5}
\definecolor{HRed}{spotcolor}{HKS14N,0.5}
\definecolor{HBlue}{spotcolor}{HKS38N,0.5}
\begin{tikzpicture}[fill opacity=0.7]
 \fill[HYellow]( 90:1.2) circle (2);
 \fill[HRed]   (210:1.2) circle (2);
 \fill[HBlue]  (330:1.2) circle (2);
 \node at ( 90:2)  {Typography};
 \node at ( 210:2) {Design};
 \node at ( 330:2) {Coding};
 \node {\LaTeX};
\end{tikzpicture}
```



voss-4.tex

Figure 1: Example for side-by-side code and output inside a `figure*` environment in twocolumn mode.

For TEX we use:

TEX marker lines

```
[...]
%StartVisiblePreamble
[... listed preamble code ...]
%StopVisiblePreamble
[...]
%StartBody
[...]
\bye
```

Now everything between `%StartBody` and `\bye` is the printed text body.

The markers are defined by the internal macro `\hv@extern@ExampleType`. This macro expects five parameters, for example for Java:

Java marker lines

```
\hv@extern@ExampleType{java}
  {//StartVisibleMain}
  {//StopVisibleMain}
  {//StartVisiblePreamble}
  {//StopVisiblePreamble}
```

The comment starter for Java is `//`, for Lua `--`, and for Perl `#`. The latter must be escaped by using `\NumChar`, as already shown in an example above. In general, the option `docType` defines the type of the source code (the comment starter), and it must have one of these values:

```
context java latex lua mp pl py tex sh
```

As you can see, only `tex` is allowed for `docType`, not `latex`, `pdflatex`, etc. This is because the comment starter is uniformly `%` for all TEX variants.

The `compiler` option defines the base program to be run, and the entire invocation, which may involve additional programs. The following `compiler` values are currently supported:

```
context java latex lua lualatex luatex
mpost pdflatex perl python3 sh tex texlua
xelatex xetex
```

The configurations for Lua, Perl, Java, shell, and Python all have the same structure; they only differ in the comment character to be used. For Lua, we have

Lua marker lines

```
\hv@extern@ExampleType{lua}
  {--StartVisibleMain}
  {--StopVisibleMain}
  {--StartVisiblePreamble}
  {--StopVisiblePreamble}
```

Sometimes, both `docType` and `compiler` are the same, for example when using Lua: `docType=lua` and `compiler=lua`. Indeed, for Lua files that also have the `.lua` extension, the value `lua` must be assigned three times:

Options for Lua code

```
ext=lua, compiler=lua, docType=lua,
```

The following Lua example writes plain text to standard output, so we pass the `redirect` option to the `externalDocument` environment; the output is then redirected into a file of the same main name but with the extension `.txt`. This is read verbatim from within the main LaTeX document and can therefore contain any characters.

```
io.write("1. "..type("Hello world").."  ")
print("2. "..type(10.4*3))
io.write("3. "..type(print).."  ")
io.write("4. "..type(type).."  ")
print("5. "..type(true))
io.write("6. "..type(nil).."  ")
print("7. "..type(type(X)))
```

voss-5.lua

```
io.write("8. "..type(a).."  ")

a = 10
io.write("9. "..type(a).."  ")
a = "a string!!"
io.write("10. "..type(a).."  ")
a = print
print("11. "..type(a))
```

```
1. string
2. number
3. function
4. function
5. boolean
6. nil
7. string
8. nil
9. number
10. string
11. function
```

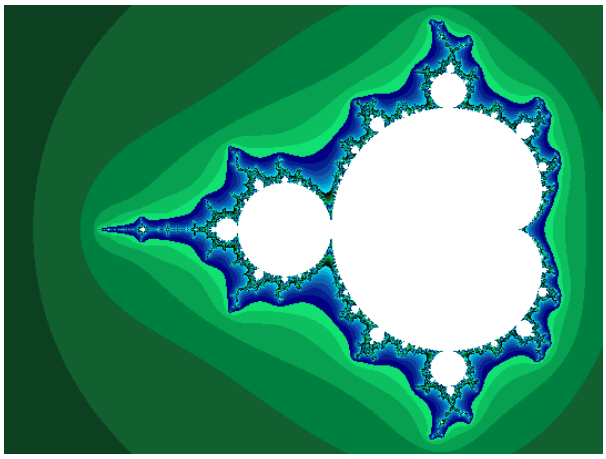
The same applies to the following example with Java code:

Options for Java code

```
ext=java, compiler=java, docType=java,
```

```
  public static int iterZahl(
    final double cx,
    final double cy,
    int maxIt,
    final double radius){
  // count the number of iterations
    int zaehler = 0;
    double zx = 0.0, zy = 0.0, tmp;
    do {
      tmp = zx*zx - zy*zy + cx;
      zy = 2*zx*zy + cy;
      zx = tmp;
      zaehler++;
  // run as long as the kength of the vector
  // is smaller than the radius
    } while (zx*zx+zy*zy<=radius && zaehler<maxIt);
    return zaehler;
  }
```

```
    double xa = -2.5, xe = 0.7, ya = -1.2, ye = 1.2;
    double dx = (xe-xa)/(imageWidth-1),
           dy = (ye-ya)/(imageHeight-1);
    double cx, cy; int R, G, B;
    double radius = 10.0;  int maxIt = 1024;
    cx = xa;
    for (int sp = 0; sp < imageWidth; sp++) {
      // from top to bottom:
      cy = ye;
      for (int ze = 0; ze < imageHeight; ze++) {
        int zIter = iterZahl(cx,cy,maxIt,radius);
        if (zIter == maxIt) {
          g.setColor(Color.WHITE);
          g.drawLine(sp, ze, sp, ze);
        } else {
          R = zIter % 4 * 6 ;
          G = zIter % 8 * 32;
          B = zIter % 16 * 16;
          g.setColor(new Color(R,G,B));
          g.drawLine(sp, ze, sp, ze);
        }
        cy = cy - dy;
      } // for ze
      cx = cx + dx;
    } // for sp
```



voss-6.java

4 Options

4.1 Program(s) and number of runs

In general, any selected `compiler` program should be found in your search path, with `pdflatex` being the default. However, in rare cases it may be necessary to specify a path for the program, which is done by assigning to `progpath`. A `/` must appear at the end, for example '`progpath=./bin/`'.

Here is the code defining the options `progpath`, `compiler`, `runs`, and `runsequence`. The full list of `compiler` values was given on previous page. (The definitions are omitted.)

Compiler options

```
\define@key{hv}{progpath}[]{...}
\define@choicekey*+{hv}{compiler}[\val\nr]{%
    context,...,xetex}
    [pdflatex]{...}
\define@key{hv}{runs}[1]{...}
\define@key{hv}{runsequence}[]{...}
```

Instead of using `compiler`, `biber` and `xindex`, an explicit run sequence can also be specified via the `runsequence` parameter. A comma-separated list is expected. The input filename is added to each program being run. For example, this sequence generates the bibliography and (with additional options) index, besides the main document:

Invocation (runsequence) example

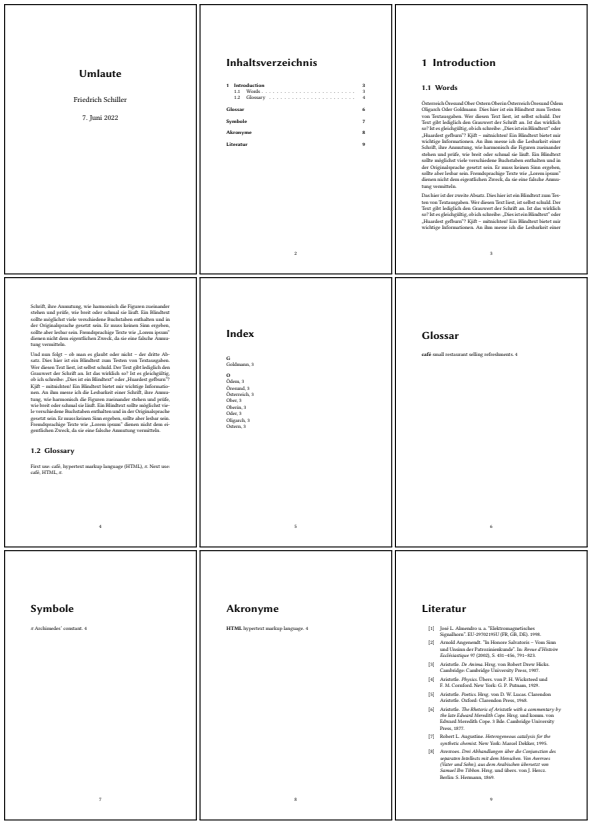
```
runsequence={lualatex,biber,xindex -l de -c DIN2,
  makeglossaries,lualatex,lualatex},
cleanup={log, aux, toc, bbl, blg,
  run.xml, bcf, idx, ilg},
pages={1,2,3,4,5,6,7,8,9},
```

The example also prints pages 1–9 of the created external document, which also has a glossary and a list of symbols and acronyms.

voss-7.tex

```
\documentclass[paper=a5,parskip=half-,DIV=12,
     bibliography=totoc,
     listof=totoc,fontsize=12pt]{scrreprt}
\usepackage[ngerman]{babel}
\usepackage{libertinus-otf,hvindex}
\usepackage{biblatex,makeidx}\makeindex
\addbibresource{biblatex-examples.bib}
\usepackage[abbreviations,symbols,postdot,
    stylemods,style=index]{glossaries-extra}
\makeglossaries
\title{Umlaute} \author{Friedrich Schiller}
```

```
\maketitle \tableofcontents
\chapter{Introduction}   \section{Words}
\Index{Österreich} \Index{Öresund}
\Index{Ober} \Index{Ostern} \Index{Oberin}
\Index{Österreich} \Index{Öresund}
\Index{Ödem} \Index{Oligarch} \Index{Oder}
\Index{Goldmann}  \Blindtext[3]
\section{Glossary}
First use: \gls{cafe}, \gls{html}, \gls{pi}.
Next use: \gls{cafe}, \gls{html}, \gls{pi}.
\printindex    \printglossaries
\nocite{*}\raggedright\printbibliography
```



4.2 Graphics options

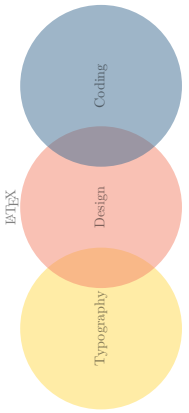
Graphics options

```
\define@key{hv}{grfOptions}[]{...}
```

The value for `grfOptions` is passed to the well-known `\includegraphics` macro, for example `{angle=90, width=\linewidth}`, as shown in the following example.

```
\usepackage{tikz}
% needs xelatex:
\usepackage[hks,pantone,
  xcolor]{xespotcolor}
```

```
\SetPageColorSpace{HKS}
\definecolor{HYellow}{spotcolor}
  {HKS05N,0.8}
\definecolor{HRed}{spotcolor}
  {HKS14N,0.8}
\definecolor{HBlue}{spotcolor}
  {HKS38N,0.8}
\begin{tikzpicture}
  [fill opacity=0.5]
\fill[HYellow](0,0)circle(2);
\fill[HRed]   (3,0)circle(2);
\fill[HBlue]  (6,0)circle(2);
 \node at (0,0){Typography};
 \node at (3,0){Design};
 \node at (6,0){Coding};
 \node at (3,2.2){\LaTeX};
\end{tikzpicture}
```



voss-8.tex

Since source code and output are arranged next to each other here, the specification `\linewidth` refers to the current width of the minipage. Thus the output has the maximum possible size.

4.3 Listing options

Listings options

```
\define@key{hv}{lstOptions}[]{...}
```

The value assigned is passed to either the macro `\lstinputlisting` or, if `usefancyvrb` is specified, to the macro `\VerbatimInput` from the `fancyvrb` package. It should be noted that the options for the respective packages have different meanings and names, so it is not so easy to switch between `listings` and `fancyvrb`.

The following example uses the `listings` package, which is the default and therefore does not require any parameter setting. A slightly exotic list of options is given, and we omit the graphical output (which is seen in the next example), purely for the demonstration:

Listings options example

```
lstOptions={basicstyle=\sffamily\slshape\scriptsize,
            columns=fullflexible,showoutput=false},
```

voss-9.tex

```
\documentclass[landscape]{article}
\usepackage[margin=1cm]{geometry}
\usepackage{pst-calendar}
```

```
\psscalebox{0.5}{%
  \psCalDodecaeder[
    Year=2022,style=june]%
}
\hspace{4cm}
\psscalebox{0.5}{%
  \psCalDodecaeder[
    Year=2022,style=july]%
}
```

4.4 Background color for the code

Different colors for the background and the frame can be selected. They can be modified via the following parameters, which show the defaults in brackets. (The actual definitions are omitted.) `BG` is the abbreviation for "background" and `BO` for "border":

Color options

```
\define@key{hv}{BGpreamble}[black12]{...}
\define@key{hv}{BGbody}[black8]{...}
\define@key{hv}{BOpreamble}[black12]{...}
\define@key{hv}{BObody}[black8]{...}
```

The parameter values are passed to a `tcolorbox` environment (of the package with the same name), and evaluated there. [3] Because the background and frame have the same color, the frame remains "invisible" by default. This changes with different values, for example:

Differing frame and background colors

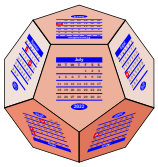
```
  BGpreamble=red!10, BOpreamble=red,
  BGbody=blue!8, BObody=blue,
```

Typically, you should use subtle colors so that the output does not fade into the background compared to the code.

voss-10.tex

```
\usepackage{pst-calendar}
```

```
\psscalebox{0.3}{%
  \psCalDodecaeder[
    Year=2022,style=july]%
}
```



We'll return to the default gray colors now.

4.5 Type of source code

The current version of `hvextern` supports source code in METAPOST, plain TEX, LATEX, ConTEXt, Python, Lua, shell, and Perl. Each language's definition contains the source code markers already mentioned, and the program invocation sequence if special treatment is necessary. For example, source code in LATEX requires special treatment if the program used is `latex`; the corresponding definition contains the following:

Marker and run setting for dvips

```
\hv@extern@ExampleType{latex}
  % for _all_ LaTeX engines
  {\string\begin\string{document\string}}
  {\string\end\string{document\string}}
  {\perCent StartVisiblePreamble}
  {\perCent StopVisiblePreamble}

% only for the sequence latex->dvips->ps2pdf
\def\hv@extern@runLATEX#1#2#3#4{%
  %path/compiler/file/extension
  \ShellEscape{#1#2\space #3#4}%
  \ShellEscape{#1dvips\space #3.dvi}%
  \ShellEscape{#1ps2pdf\space
     -dAutoRotatePages=/None\space
     -dALLOWPSTRANSPARENCY\space#3.ps}%
}
```

The macro must have the following structure:

Macro implementing the run sequence

```
\def\hv@extern@run<NAME>#1#2#3#4{%
  %path/compiler/file/extension
  ...}
```

The definition for TEX is similar. The type of source code and the program used can be different for TEX, LATEX and ConTEXt, for example type `latex` but program `lualatex`.

4.6 Output of one or more full pages

In the event that only a subset of pages are to be output, this can be controlled via the `pages` parameter, as we saw in a previous example. It expects a comma-separated list of the pages to be printed. The individual pages can be framed with the `frame` parameter in order to achieve a clearer presentation.

Output page selection

```
\define@key{hv}{pages}[1]{...}
\define@key{hv}{pagesep}[1em]{%
     \hv@extern@pagesep=#1}
\define@boolkey{hv}[hv@extern@]{frame}[true]{}
```

It is up to the user to use the `grfOptions` parameter to ensure that the pages for the output are scaled as needed. This example outputs the first three pages of a document:

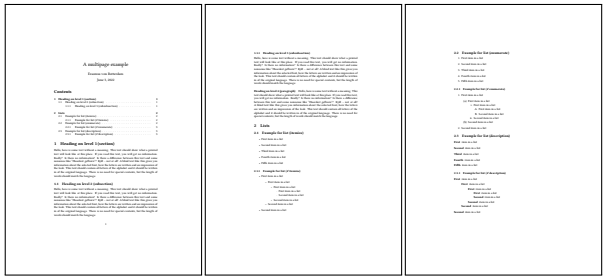
Page selection example

```
pages={1,2,3}, grfOptions={width=0.3\linewidth},
pagesep=1pt,
frame, compiler=lualatex, runs=2, % for the TOC
```

voss-11.tex

```
\usepackage[american]{babel}
\usepackage{libertinus}
\usepackage{blindtext}
```

```
\title{A multipage example}
\author{Erasmus von Rotterdam}
\maketitle
\tableofcontents
\blinddocument
```



4.7 Output as a float

As a rule, the output is in the running text, which can be undesirable if the text width is relatively small. Larger free spaces can then arise on one side, which is always unfavorable. In such cases you should use the `float` option, in which case, as usual, a caption can be specified using the `caption` parameter and a cross-reference label can be specified using `label`. The floating type is by default `figure` and the placement can be set by the optional argument `floatsetting`. It is preset to `!htb`.

voss-12.tex

```
\usepackage{pst-coxeterp}
```

```
\begin{pspicture}(-1,-1)(1,1)
  \Simplex[dimension=2]\end{pspicture}
\begin{pspicture}(-1,-1)(1,1)
  \Simplex[dimension=3]\end{pspicture}
\begin{pspicture}(-1,-1)(1,1)
  \Simplex[dimension=5]\end{pspicture}
\begin{pspicture}(-1,-1)(1,1)
  \Simplex[dimension=7]\end{pspicture}
```

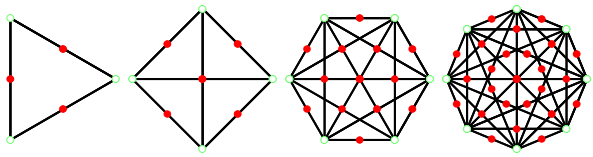


Figure 2: An example for Coxeter images.

Float options

```
\define@boolkey{hv}[hv@extern@]{float}[true]{}
\define@key{hv}{caption}[]{...}
\define@key{hv}{label}[]{...}
```

Figure 2 shows an example as a floating object. It was created with the following parameters:

Float example

```
[...]
 float,
 caption={An example for Coxeter images.},
 label=img:cox,
[...]
```

The specification `float` refers only to the output; otherwise, a previous listing could not have a page break and the typesetting of the text would be more difficult. On the other hand, it may well be that other text appears between the code and the output of an example. Then manual intervention is necessary, for example by using the command `\captionof` from the package `caption`, which allows a caption without floating space.

5 Cropping white space

When displaying the output of examples, usually only the area that contains a graphic or text is of interest, and we want to remove any surrounding white space. For documents that consist of only one page, the document class `standalone` can generally be used, which automatically removes all white space. If you have more than one page or want to use another special document class, `hvextern` provides the `crop` option:

Crop options

```
\define@boolkey{hv}[hv@extern@]{crop}[true]{}
\define@key{hv}{cropmargin}[2]{...}% in pt
```

`crop` can also be applied to documents with multiple pages. In this case, however, you should make sure that the pages have headers and footers so that the white space that is cut off always has the same size. Otherwise the pages end up with different heights, as shown in the example below, which is usually undesirable. Among other things, the following parameters were set:

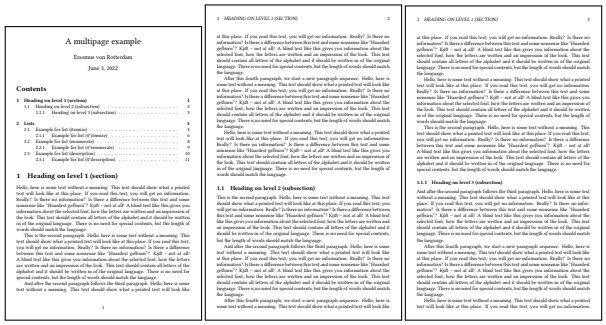
Crop example

```
pages={1,2,3}, grfOptions={width=0.3\linewidth},
frames, pagesep=1pt, crop, cropmargin=5,%is 5pt
compiler=lualatex, runs=2, % for the TOC
```

voss-13.tex

```
\usepackage[american]{babel}
\usepackage{libertinus}
\usepackage{blindtext}
\pagestyle{headings}
```

```
\title{A multipage example}
\author{Erasmus von Rotterdam}
\maketitle
\tableofcontents
\Blinddocument
```



5.1 Source code and output side by side

Normally the source code is printed first and then the output. This order cannot be changed with the current version of `hvextern`. For side-by-side output, the `mpwidth` parameter determines the width of a left minipage and is always evaluated if it is greater than 0 pt. A second minipage is then reserved for output for the remainder of the line, except for the value of `mpsep`. Both minipages are aligned to the value of `mpvalign`, the top edge by default.

Side-by-side options
```
\define@key{hv}{mpwidth}[0pt]{...}
\define@key{hv}{mpsep}[1em]{...}
```

The default distance between the two minipages is 1 em, as shown.

5.2 Horizontal alignment of the output

The code is always left-aligned, whereas the output can use various known alignments via the `align` option. The use of the `ragged2e` package does not have any advantages here.

Horizontal alignment
```
\define@key{hv}{align}[\centering]{...}
```

The default with `align=\centering`:

```
\rule{0.5\linewidth}{3mm}
```

Left-justified with `align=\raggedright`:

```
\rule{0.5\linewidth}{3mm}
```

Right-justified with `align=\raggedleft`:

```
\rule{0.5\linewidth}{3mm}
```

Side by side, default with `align=\centering`:

```
\rule{0.2\linewidth}{3mm}
```

Side by side, with `align=\raggedright`:

```
\rule{0.2\linewidth}{3mm}
```

Side by side, with `align=\raggedleft`:

```
\rule{0.2\linewidth}{3mm}
```

5.3 Inline output, rather than displayed

The output can be printed within a line in the so-called inline mode. This can make sense if you don't have certain characters available in your document's font, but they can be generated externally and then input as PDF. Here, the corresponding source code should not be shown, so `code=false` is automatically set with `inline`.

Inline option
```
\define@boolkey{hv}[hv@extern@]{inline}[true]{...}
```

An example has already been shown on page 280.

5.4 Handling plain text output

For LaTeX documents, the output is generally PDF, but when using a programming language such as Perl, the output would more typically be plain text. This must be redirected or written to a file so that it can be inserted verbatim into the main document. This can be controlled with the parameters `includegraphic` and `redirect`. The output is typeset with `listings` or `fancyvrb`, and options for the typesetting environment set with `textoptions`.

With `includegraphic=false` it is up to the user to ensure that every output within the external document is written to a text file; `hvextern` looks for a file with the right name. This is done automatically with `redirect`, but then all program output ends up in the external text file.

Plain text output options
```
\define@boolkey{hv}[hv@extern@]{redirect}[true]{}
\define@boolkey{hv}{includegraphic}[true]{}
\define@key{hv}{textOptions}[]{...}
```

The text file must have the same main file name as the external file, but with the extension `.txt`. As we've seen, each program can determine by itself what name it was called with, so it is easily possible to determine the correct name for the text output file. For a Perl program, this could be achieved with the following code, for example:

Perl: get filename
```
my $filename = $0; # the current filename
$filename =~ s/\.pl//; # without extension .pl
$filename = "${filename}.txt"; # for the output
open(my $fh, '>', $filename);
```

However, in the next example, the optional keyword `redirect` is given, so determining the filename in the code is not needed. The example is set with:

Example output redirect
```
compiler=perl, includegraphic=false, docType=pl,
ext=pl, usefancyvrb, runs=1, code, redirect,
tcbox=false, force, lstOptions={fontsize=\small,
  fontfamily=tt, frame=lines}
```

```
my $number = 1;
my $start = 1;
my $end    = 9;
my $found = 0;
```

```
print "Searching for Kaprekar constants\n";
while ($number < 8) {
 print "${number}-digits: ";
 foreach ($start...$end){
  @chars = split(//,$_);
  $Min = join("",sort(@chars));
  $Max = reverse($Min);
  $Dif=$Max-$Min;
  if($_ eq $Dif) {
```

voss-20.pl

```
   $found = 1;
   print $_,",  ";
  }
 }
 if (!$found) { print "---\n"; }
 else         { print "\n"; }
 $found = 0;
 $number++;
 $start = $start*10;
 $end   = $end*10;
}
```

```
Searching for Kaprekar constants
1-digits: ---
2-digits: ---
3-digits: 495,
4-digits: 6174,
5-digits: ---
6-digits: 549945,  631764,
7-digits: ---
```

(Just for the sake of completeness: A Kaprekar constant is a number A with $\max(A) - \min(A) = A$, where max and min are the sorted/reverse-sorted digits of the number, e.g., $A = 495 = 954 - 459$.)

Our next example is in Lua, and also produces text output; but instead of using `redirect`, the code outputs to the appropriate file. This filename can be determined as follows:

Lua: get filename

```
-- get full filename
local filename = arg[0]
-- delete extension
local shortFN = str:match("(.+)%..+")
-- open external file
outFile = io.open(shortFN..".txt","w+")
```

```
function nextrow(t)
  -- fill table
  local ret = {}
  t[0], t[#t+1] = 0, 0
  for i = 1, #t do
    ret[i]=t[i-1]+t[i]
  end
  return ret
end

function triangle(n)
 t = {1}
 for i = 1, n do
  m = (n - i)
  for j = 1,m do
    outFile:write("  ")
  end
  for k = 1,i do
   outFile:write(
    string.format("%4s",t[k]))
  end
  outFile:write("\n")
  t = nextrow(t)
 end
end
```

voss-21.lua

```
triangle(10)
```

```
                  1
                1   1
              1   2   1
            1   3   3   1
          1   4   6   4   1
        1   5  10  10   5   1
      1   6  15  20  15   6   1
    1   7  21  35  35  21   7   1
  1   8  28  56  70  56  28   8   1
1   9  36  84 126 126  84  36   9   1
```

5.5 Generating the bibliography and index

The current version of `hvextern` has predefined support for constructing the bibliography with Biber and an index with `xindex`, via the following parameters:

Index and bibliography options

```
\define@boolkey{hv}[hv@extern@]{biber}[true]{}
\define@boolkey{hv}{xindex}[true]{}
\define@key{hv}{xindexOptions}[]{...}
```

In principle, the external run of Biber does not require any further parameters, whereas the `xindex` program requires information about the language, the style file, etc., for example. The next example is generated with the following parameters:

xindex example

```
\begin{externalDocument}[
compiler=lualatex,runs=2,pages=2,crop,
xindex,xindexOptions={-l DE --config DIN2},
docType=latex,cleanup={log,aux,ilg,idx},...]
```

To use other bibliography or index programs, you can use the `runsequence` option; see the example on p. 284.

```
\usepackage{makeidx}
\makeindex
\usepackage{hvindex}
```

voss-22.tex

```
\Index{Österreich} \Index{Öresund}
\Index{Ödipus} \Index{Öchsle}
\Index{Ostern} \Index{Ober} \Index{Oberin}
\Index{Österreich} \Index{Öresund}
\Index{Ödem} \Index{Oligarch} \Index{Oder}
\Index{Ostern} \Index{Ober} \Index{Oberin}
\Index{Obstler} \Index{Öl} \Index{ölen}
\Index{Oder|seealso{Fluss}} \Index{Göbel}
\Index{oder} \index{Fluss!Oder}
\Index{Goethe} \Index{Göthe} \Index{Götz}
\Index{Goldmann}
\printindex
```

Index

6 Verbatim modes: `listings` and `fancyvrb`

6.1 Using `listings`

By default the command `\lstinputlisting` from the package `listings` is used for printing the source code. We saw an example of setting `hvextern`'s `lstOptions` option for it earlier.

6.2 Package `fancyvrb`

There are no fundamental objections to the `listings` package, but sometimes it makes more sense to use `\VerbatimInput` from the `fancyvrb` package, especially when displaying non-ASCII Unicode characters. Most of the examples in this article use `fancyvrb`, by passing the `usefancyvrb` option.

6.3 Vertical space

Vertical space can be controlled by four keywords for the stretchable vertical space:

`aboveskip` The vertical space before the environment `externalDocument` or the command `\runExtCmd` (default `\medskipamount`).

`belowpreambleskip` The vertical space between the preamble and body (default `\smallskipamount`). If the preamble is missing, then there will be only `aboveskip`.

`belowbodyskip` The vertical space between body and output (default `\smallskipamount`).

`belowskip` The vertical space after the environment `externalDocument` or the command `\runExtCmd` (default `\medskipamount`).

7 Supported METAPOST and TEX engines

Here we show a few examples using the common TEX-world programs. (LATEX is omitted here since most of the examples throughout this article use LATEX.)

7.1 METAPOST example

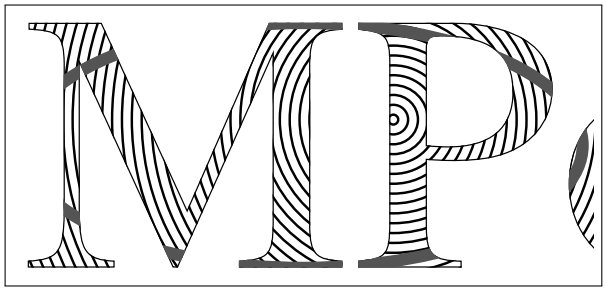
MetaPost example

```
\begin{externalDocument}[
  compiler=mpost,docType=mp,...]
```

```
defaultfont:="ptmr8r";
warningcheck:=0;
```

```
draw fullcircle shifted (0.5,0.6) xscaled 8cm
  yscaled 3.5cm withpen pencircle scaled 5bp
  withcolor 0.33; % gray bands
special("/Times-Roman findfont 150 "
& " scalefont setfont "
& " 0 10 moveto (MPost) false charpath 2 "
& " clip stroke gsave 150 70 translate "
& " 2 4 600 {dup 0 moveto 0 exch 0 exch"
& " 0 360 arc stroke} for grestore ");
```

voss-23.mp



Here is the definition of the command sequence for running METAPOST, just in case you want to modify something:

MetaPost run sequence

```
\def\hv@extern@runMP#1#2#3#4{%
  % path / compiler / file / extension
  \ShellEscape{#1#2\space -tex=tex\space #3#4}%
  \ShellEscape{#1tex\space "\string\input\space
    epsf\string\relax\string\nopagenumbers
    \string\epsfbox{#3.1}\string\bye"}%
  \ShellEscape{#1dvips\space -j\space -E\space
    -o\space #3.eps\space epsf.dvi}%
  \ShellEscape{#1epstopdf\space #3.eps}%
}
```

7.2 Plain TEX example

Plain TEX example

```
\begin{externalDocument}[
  compiler=tex,docType=tex,...]
```

voss-24.tex

```
\footline={\footsc the electronic journal
 of combinatorics {\footbf 16} (2009),
 \#R00\hfil\footrm\folio}
```

```
\font\bigrm=cmr12 at 14pt
\centerline{\bigrm An elementary proof
  of the reconstruction conjecture}

\bigskip\bigskip
\centerline{D. Remifa\footnote*{Thanks to the
  editors of this journal!}}
\smallskip
\centerline{Department of Inconsequential Studies}
\centerline{Solatido College, North Kentucky, USA}
\centerline{\tt remifa@dis.solatido.edu}
\bigskip
\centerline{\footrm
Submitted: Jan 1, 2009; Accepted: Jan 2, 2009;
  Published: Jan 3, 2009}
\centerline{\footrm Mathematics Subject
  Classifications: 05C88, 05C89}
\bigskip\bigskip
\centerline{\bf Abstract}
\smallskip
{\narrower\noindent
The reconstruction conjecture states that the
multiset of unlabeled vertex-deleted subgraphs
of a graph determines the graph, provided it
has at least 3 vertices.  A version of the problem
was first stated by Stanis\l aw Ulam. In this
paper, we show that the conjecture can be proved
by elementary methods.  It is only necessary to
integrate the Lenkle potential of the Broglington
manifold over the quantum supervacillatory measure
in order to reduce the set of possible
counterexamples to a small number (less than a
trillion).  A simple computer program that
implements Pipletti's classification theorem for
torsion-free Aramaic groups with symplectic socles
can then finish the remaining cases.}

\bigskip
\beginsection 1. Introduction.

This is the start of the introduction.
```


An elementary proof of the reconstruction conjecture

D. Remifa*
Department of Inconsequential Studies
Solatido College, North Kentucky, USA
remifa@dis.solatido.edu

Submitted: Jan 1, 2009; Accepted: Jan 2, 2009; Published: Jan 3, 2009
Mathematics Subject Classifications: 05C88, 05C89

Abstract

The reconstruction conjecture states that the multiset of unlabeled vertex-deleted subgraphs of a graph determines the graph, provided it has at least 3 vertices. A version of the problem was first stated by Stanisław Ulam. In this paper, we show that the conjecture can be proved by elementary methods. It is only necessary to integrate the Lenkle potential of the Broglington manifold over the quantum supervacillatory measure in order to reduce the set of possible counterexamples to a small number (less than a trillion). A simple computer program that implements Pipletti's classification theorem for torsion-free Aramaic groups with symplectic socles can then finish the remaining cases.

1. Introduction.
This is the start of the introduction.

* Thanks to the editors of this journal!

THE ELECTRONIC JOURNAL OF COMBINATORICS 16 (2009), p.R00 1

7.3 ConTEXt example (mkIV)

This example is run with ConTEXt from TEX Live 2022, but it should also work with the new LMTX.

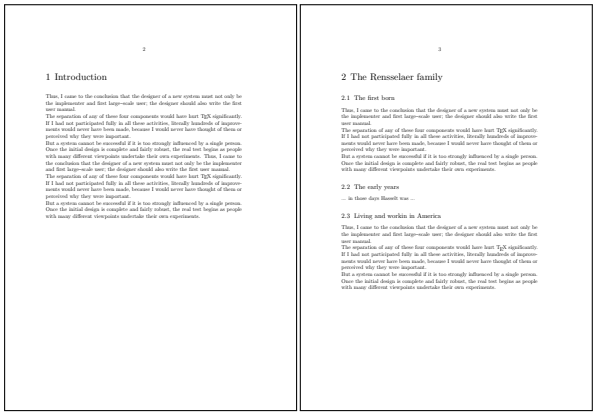
ConTEXt example

```
\begin{externalDocument}[pages={3,4},
compiler=context,docType=context,runs=2,...]
```

voss-25.tex

```
\definehead
  [myhead]
  [section]
\setuphead
  [myhead]
  [numberstyle=bold,
   textstyle=bold,
   before=\hairline\blank,
   after=\nowhitespace\hairline]
```

```
\startstandardmakeup
\midaligned{From Hasselt to America}
\midaligned{by}
\midaligned{J. Jonker and C. van Marle}
\stopstandardmakeup
\placecombinedlist[content]
\chapter{Introduction}
\input knuth \input knuth
\chapter[rensselaer]{The Rensselaer family}
\section{The first born}
\input knuth
\section{The early years}
... in those days Hasselt was ...
\section{Living and workin in America}
\input knuth
\chapter[lansing]{The Lansing family}
\input knuth
... the Lansing family was also ...
\chapter[cuyler]{The Cuyler family}
\input knuth
... much later Tydeman Cuyler ...
\myhead[headlines]{And the end}
foo
```


2

1 Introduction

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large-scale user; the designer should also write the first user manual.
The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.
But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments. Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large-scale user; the designer should also write the first user manual.
The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.
But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

3

2 The Rensselaer family

2.1 The first born

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large-scale user; the designer should also write the first user manual.
The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.
But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

2.2 The early years

... in those days Hasselt was ...

2.3 Living and workin in America

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large-scale user; the designer should also write the first user manual.
The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.
But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

8 Running arbitrary external commands

To typeset listing of the current directory of this document we can use the macro `\runExtCmd` with the optional argument `redirect`. We filter the output with additional commands.

```
\runExtCmd[redirect]
  {ls -laB | awk '{print $6,$7,$8,$9}' }
  {voss}
```

to produce the directory listing:

```
Nov 3 18:49 .
Nov 1 23:39 ..
Jun 3 17:36 .dict.pws
Nov 3 18:49 Exa-extern
Jun 3 18:04 Makefile
Nov 3 18:49 firstpage.tex
Nov 3 18:49 lastpage.tex
Nov 3 18:49 tb135voss-extern.aux
Nov 3 18:49 tb135voss-extern.bbl
Jun 3 17:36 tb135voss-extern.bib
Nov 3 18:49 tb135voss-extern.blg
Nov 3 18:49 tb135voss-extern.log
Oct 31 16:12 tb135voss-extern.ltx
Nov 3 18:49 tb135voss-extern.out
Nov 3 18:49 tb135voss-extern.pdf
```

The general behaviour is similar to the environment, `externalDocument`: the output is saved in an intermediate file, in this case `voss-`⟨*num*⟩`.txt` and then read by `\VerbatimInput`. The options `code` and `showFilename` are off by default.

9 Other options

Most of the options which `hvextern` provides for `externalDocument` and `\runExtCommand` have been discussed. Here is a brief summary of some that have not been seen, or mentioned only in passing.

`force` With `force=false` an existing output file is used, thus reducing compilation time. This option should only be used in exceptional cases, because with it, a change in the main document in the source code of the example does not lead to updated example output.

`moveToExampleDir` Move all generated example files, both source and output, to the directory specified by `ExamplesDir`. This can ease document development and maintenance when there are many examples. The directory itself must be created by the user.

`ExampleDir` The directory to which example files are moved if requested.

`cleanup` Auxiliary files from an (LA)TEX or other run can be deleted to improve the overview in a directory. By default, these are the `.aux` and `.log` files: `cleanup={aux,log}`.

`framesep` Value for `\fbox` if keyword `frame` is used.

`mpsep` Distance between code and output (default 1 em).

`pagesep` Distance between pages for multipage output (default 1 em).

`shiftFN` Length to shift marginal filename; positive values shift up.

`inline` False by default; if true, include the generated output in the paragraph, not as a display.

`showoutput` True by default; if false, omit the generated output.

`code` True by default (unless `inline=true`); if false, omit the source code listing.

`tcbox` If `false`, do not use any box commands from the `tcolorbox` package (for debugging and in case of bugs).

`eps` Convert the generated PDF file to EPS (historical reasons).

10 Caveats

Due to issues with `tcolorbox`, you can expect problems if a page break appears in the code part immediately before the first code line is printed. In such a case put a `\newpage` or `\pagebreak` just *before* the `externalDocument` environment. If you do not need `tcolorbox` features, you can disable its use with the option `tcbox=false`.

References

[1] C. Heinz, J. Hoffmann, B. Moses. The listings package, version 1.8d, 2020-03-24. ctan.org/pkg/listings

[2] R. Niepraschk. The showexpl package, version 0.3s. ctan.org/pkg/showexpl

[3] T.F. Sturm. The tcolorbox package, version 5.0.2, 2022-01-07. ctan.org/pkg/tcolorbox

[4] T. Van Zandt, H. Voß, et al. The fancyvrb package, version 4.2. ctan.org/pkg/fancyvrb

[5] H. Voß. The hvextern package, version 0.28. ctan.org/pkg/hvextern

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