The **svg** Package

Philip Ilten
`philten@cern.ch`

v1.0 (2012/09/05)

Contents

1 Introduction **1**

2 Usage **2**

3 Example **4**

4 ROOT **5**

5 Implementation **7**

6 Thanks **11**

1 Introduction

The open source program Inkscape has provided an excellent resource for the simple and easy creation of images and diagrams using a graphical user interface. The work by Johan B. C. Engelen has further enhanced the ability of Inkscape to split an svg into a text component that can be imported into LaTeX, and an image component that can be imported as a pdf.[1] Consequently it is now possible to include an svg into a LaTeX document where the text within the svg has been rendered natively by LaTeX.

The purpose of this package is twofold. First, the syntax of the command `\includegraphics` from the `graphicx` package has been extended to an `\includesvg` command, which allows the specification of the svg width and height using keys in an optional first argument. Second, this package allows for the extraction of the svg, as rendered within the LaTeX document, to an independent image file. This is particularly useful when attempting to provide images to journals or collaborators, and one wishes the image to appear exactly as it does within the original LaTeX document.

[1]For further information see the `svg-inkscape` documentation on CTAN.

There is actually a third purpose to this package, which will almost certainly be relevant to experimental particle physicists only, who frequently use the analysis package ROOT. Further details on how to obtain beautiful ROOT plots using this package are given in Section 4.

This documentation is broken into five parts: an explanation of the usage is given in Section 2, an example is given in Section 3, further details with use in ROOT is given in Section 4, and finally, the full implementation is given in Section 5 which hopefully should not need to be read.

There is one further point which is important to mention. This package relies heavily upon executing commands from the shell using the `\write18` command, and so it is necessary for the flag `-shell-escape` to be included when compiling documents using this package. Additionally, this package requires a working installation of INKSCAPE and PDFLATEX in order for an SVG to be included or extracted to a PDF. In order to extract to EPS and PNG formats, the programs PDFTOPS and CONVERT (part of ImageMagick) must be installed respectively. Finally, this package will not work on Windows, but should run on any *nix platform as long as the paths to the appropriate programs are correctly defined.

2 Usage

`\includesvg` The command to include an SVG is similar to the `\includegraphics` command provided by the `graphicx` package. However, now the command

```
\includesvg[<options>]{<svg filename>}
```

is used where `<svg filename>` is the filename of the SVG without the path or the `.svg` postfix.

`\setsvg` The `<options>`, described in detail below, can be specified globally for the package

```
\usepackage[<options>]{svg}
```

and reset locally when supplied to the `\includesvg` macro. The options can also be reset globally using the macro `\setsvg`

```
\setsvg{<options>}
```

where `<options>` is a comma separated list of options.

options `width` `height` `svgpath` The width of the SVG can be specified via the `width` option and the height by the `height` option. If both the width and height are specified, the width will be used and the height will be rescaled to match the aspect ratio of the SVG. The path to the SVG can be specified using the `svgpath` option, where the path must terminate in a `/`. The default `svgpath` is set to the current directory, `./`.

options `pdf` `eps` `png` The included SVG can be extracted from the document into a PDF, EPS, or PNG independent of the document. The `pdf` flag enables PDF extraction, while the `eps` and `png` flags enable EPS and PNG extraction respectively. For example,

```
\includesvg[pdf,eps,<additional options>]{<svg filename>}
```

will extract the SVG to both PDF and EPS formats By default, all of these flags are set to false and no extraction of the SVG is performed. The extraction will render the SVG to the specified output(s) of choice using the same size as specified within the `\includesvg` command. Consequently, the scale between the image and text in the extracted output(s) will remain identical to the scale within the document from which the SVG was extracted.

options `name` `path`

The root name of the extracted output can be specified with the `name` option. For example,

```
\includesvg[name=foo,eps,png,<additional options>]{<svg filename>}
```

will extract the SVG to the files `foo.eps` and `foo.png` in the current directory. By default, `name` is set to `Fig.\arabic{svgfigure}\alph{subfigure}` and so any SVG included within a `figure` or `subfigure` environment will automatically be labeled; i.e. if an SVG is included in the first figure and second subfigure of the document, and PDF extraction was requested, the SVG will be extracted to the file `Fig.1b.pdf`. A path for the extracted files can also be specified with the `path` option, which must terminate with a `/`. The default `path` is set to the current directory, `./`.

options `clean` `exclude`

Because a large number of files is generated for each SVG extraction, it is oftentimes desirable to automatically remove the temporary files. Using the option `clean` will remove any generated files created other than the extracted output(s) requested. The `clean` option is by default set as `false` to enable debugging. Additionally, sometimes it may be necessary to export an SVG without including it in the current document. If the flag `exclude` is specified, the SVG will not be rendered in the current document, but will be extracted to the requested output(s).

options `pretex` `postex`

Commands prior and post to the inclusion of the SVG may be desired, such as font or color commands. For example, to change the text size of the include SVG text one could use

```
{\tiny \includsvg[<options>]{<svg filename>}}
```

where now the text will be rendered in the font size specified by `\tiny`. In this example, however, the `\tiny` command would not be included in the extracted output and so the options `pretex` and `postex` are provided where the LaTeX provided to `pretex` is included before the SVG, and `postex` after the SVG. Consequently, the example above can be rewritten as

```
\includsvg[pretex=\tiny,<additional options>]{<svg filename>}
```

where now the changed font size will be propagated to the extracted output.

options `preamble` `end`

Specialized LaTeX macros can be used in the SVG which can then be defined in the preamble of the LaTeX document in which the SVG is to be included. Additionally, specialized packages such as `\relsize` may be needed by the LaTeX code extracted from the SVG. Consequently, the preamble of the current LaTeX document is used for the extraction of the SVG by default. It is possible, however, to specify a different preamble with the option `preamble` where the file to use as the preamble (including path and postfix) is given as the argument. The default definition of `preamble` is `\jobname.tex`, and should suffice for most cases. The preamble up to the line defined by the option `end` will be used, which is set to a default of `\begin{document}`. Notice that an exact match must be made, and so if any comments or text are on the same line after the `\begin{document}`, the preamble will not be correctly extracted.

options `inkscape` `pdflatex` `pdftops` `convert`

A variety of commands are executed directly to the system, via `\write18` using this package and consequently, it may be necessary to change the binary paths and options for each individual command. For the inclusion of an SVG, INKSCAPE is used to separate the text and image from the SVG and can be set using the `inkscape` option. By default the `inkscape` option is set to `inkscape -z -C` which performs a non-gui export of SVG page (notice that the `-C` option indicates page and not drawing). For the extraction of a PDF, the LaTeX program is used which is set by the `pdflatex` option and set to `pdflatex` by default. The extraction of an EPS is performed by converting a PDF to an EPS using PDFTOPS. This command is set with the `pdftops` option and is set by default to `pdftops -eps`. Finally, conversion to PNG is accomplished via the CONVERT program which is set with the `convert` option and by default set to `convert -density 300` where `-density` controls the resolution of the extracted PNG in dots per inch.

3 Example

As an example[2] take the following lines of code

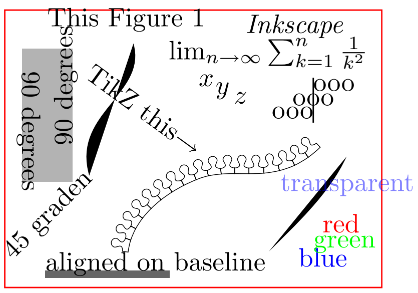
```
\begin{figure}
  \subfloat[This text is too large!]{\includesvg[clean,
  preamble=preamble.tex,pdf,width=5cm]{example}}
  \subfloat[This text fits better.]{\includesvg[clean,
  preamble=preamble.tex,eps,pretex=\relscale{0.5},width=5cm]{example}}
  \caption{An example figure.\label{fig:example}}
\end{figure}
```

where the SVG `example.svg` within this directory has been included twice using the `\svginclude` command. The output is shown in Figure 1.

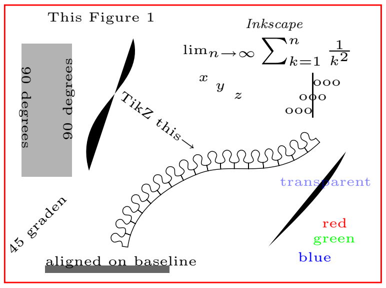
The first subfigure is created with the export option `pdf` with the default `name`, and so the file `Fig.1a.pdf` is extracted. However, the text is overrunning the margins of the image, and so the second subfigure decreases the relative size of the text within the image using the `pretex` option. Now, an EPS is requested for extraction, and so the file `Fig.1b.eps` is created.

Notice that for both subfigures, the `preamble` was set to `preamble.tex`, also included in this current directory, rather than the default current document. Additionally, the flag `clean` has been used which forces the cleanup of any extraneous generated files.

[2]The image used here is a slightly modified version of the image used in the initial documentation on how to include an SVG in LaTeX by J. Engelen available on CTAN.



(a) This text is too large!



(b) This text fits better.

Figure 1: An example figure.

4 ROOT

ROOT has the ability to export directly to an SVG, which means that it is possible to completely by-pass all of ROOT's internal text rendering machinery, and let LaTeX handle the text natively. This means that all of the ugly fonts that are rendered by ROOT can now be completely avoided, with the additional bonus of being able to add references within plots. So how does one go about using this package with ROOT?

1. Create the plot with ROOT as normal, but turn off all LaTeX interpretation of text strings. This is a bit tricky, but can be accomplished by setting the font in ROOT to a precision of zero as described in the documentation for `TAttFill`. Remember that the font is set using the function `(TAttFill*)->SetTextFont(i)` where `i` is the (font type) $\times$ 10 + (font precision). In the following lines of code, a `TStyle` is defined which sets the font to type "Courier New" with a precision of zero.

   ```
   TStyle *style = new TStyle("style","style"); int FONT = 80;
   style->SetTextFont(FONT);
   style->SetLabelFont(FONT,"XYZ");
   style->SetTitleFont(FONT,"XYZ");
   style->SetTitleFont(FONT,"");
   gROOT->SetStyle("style");
   gROOT->ForceStyle();
   ```

   Now, just use the standard LaTeX syntax for creating labels, etc. Note however, that the backslash must be escaped due to interpretation of special characters by C++.

2. Print the plot as an SVG.

   ```
   gPad->Print("foo.svg");
   ```

3. Include the SVG within the document using this package.

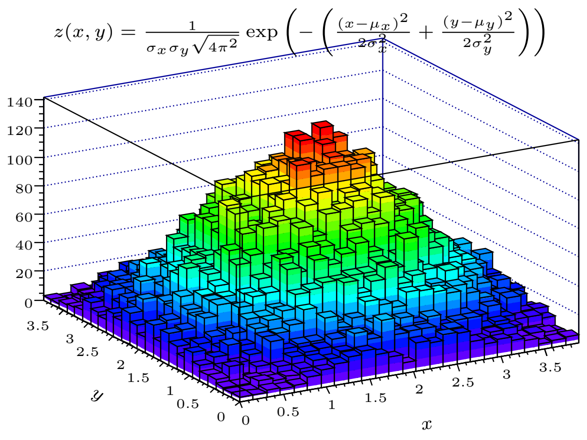


Figure 2: Rendering of a ROOT plot (no more "Comic CERNs").

```
\usepackage[clean,pdf]{svg}
...
\includesvg[width=\columnwidth]{foo}
```

Consider the example image produced by ROOT in Figure 2. This figure was generated by the ROOT macro `root.C`, provided within this directory, which produces the SVG `root.svg` when run. The code used to produce this SVG from within ROOT is

```
void root() {

  // Set the style.
  gStyle->SetTextFont(80);     gStyle->SetLabelFont(80,"XYZ");
  gStyle->SetTitleFont(80,""); gStyle->SetTitleFont(80,"XYZ");
  gStyle->SetPalette(1);       gStyle->SetOptStat(0);

  // Draw the plot.
  TH2D *h = new TH2D("", "", 25, 0, 3.9, 25, 0, 3.9); TRandom r;
  for (int i = 0; i < 30000; i++) h->Fill(r.Gaus(2.,1), r.Gaus(2.,1));
  h->GetXaxis()->CenterTitle(); h->GetXaxis()->SetTitleOffset(2.5);
  h->GetYaxis()->CenterTitle(); h->GetYaxis()->SetTitleOffset(2.5);
  h->GetXaxis()->SetTitle("\\larger[2]$x$");
  h->GetYaxis()->SetTitle("\\larger[2]$y$");
  h->Draw("LEGO2");

  // Draw additional text.
  TText *t = new TText(); t->SetTextAlign(31);
  t->DrawText(0.7, 0.9, "\\larger[2]$z(x,y) = \\frac{1}{\\sigma_x\\sigma_y"
            "\\sqrt{4\\pi^2}}\\exp\\left(- \\left(\\frac{(x-\\mu_x)^2}"
            "{2\\sigma_x^2} + \\frac{(y-\\mu_y)^2}{2\\sigma_y^2} \\right)"
            "\\right)$");

  // Print the plot.
  gPad->Print("root.svg");
}
```

where the text produced within the Root plot is set to a precision of zero. The plot was then included within this document using the LaTeX code

```
  \begin{figure}
    \begin{center}
      \includesvg[clean,preamble=preamble.tex,pdf,png,height=6cm,pretex=\tiny]
      {root}
    \end{center}
    \caption{Rendering of a \croot plot (no more ‘‘Comic
    CERNs’’).\label{fig:root}}
  \end{figure}
```

which produces the extracted images `Fig.2.pdf` and `Fig.2.png`. Enjoy plots from Root with natively rendered LaTeX!

5 Implementation

`initialization` The package, which requires the packages `xkeyval` for the options, `subfig` for automatic labeling within the subfigure command, the `import` package for correct handling of paths, `graphicx` for the PDF inclusion commands, `transparent` for transparency, and `xcolor` for color, is initialized.

```
\ProvidesPackage{svg}[2012/09/05 v1.0 SVG inclusion and extraction]%
\@ifpackageloaded{xkeyval}{}{\RequirePackage{xkeyval}}%
\@ifpackageloaded{subfig}{}{\RequirePackage{subfig}}%
\@ifpackageloaded{import}{}{\RequirePackage{import}}%
\@ifpackageloaded{graphicx}{}{\RequirePackage{graphicx}}%
\@ifpackageloaded{transparent}{}{\RequirePackage{transparent}}%
\@ifpackageloaded{xcolor}{}{\RequirePackage{xcolor}}%
```

`input definition` All commands used for input (i.e. for the SVG and preamble) are defined within the `\SVG@in` prefix, and set by the key definition of the line following their definition. The `exclude` boolean, used to stop the inclusion of the SVG within the document is also defined.

```
\def\SVG@in@preamble{\jobname.tex}%
\define@key[SVG]{svg.sty}{preamble}{\def\SVG@in@preamble{#1}}%
\def\SVG@in@path{./}%
\define@key[SVG]{svg.sty}{svgpath}{\def\SVG@in@path{#1}}%
\def\SVG@in@end{\begin{document}}%
\define@key[SVG]{svg.sty}{end}{\def\SVG@in@end{#1}}%
\define@boolkey[SVG]{svg.sty}[SVG@in@]{exclude}[true]{}%
```

`length definition` All commands used for output are defined within the `\SVG@out` prefix, beginning with the dimensions of the extracted image. If no dimensions are supplied both `useheight` and `usewidth` are `false`, and so the natural dimensions of the SVG are used. If both `usewidth` and `useheight` are `true`, the width is used.

```
\newlength\SVG@out@width%
\newif\ifSVG@out@usewidth%
\define@key[SVG]{svg.sty}{width}%
{\setlength{\SVG@out@width}{#1}\SVG@out@usewidthtrue}%
\newlength\SVG@out@height%
\newif\ifSVG@out@useheight%
\define@key[SVG]{svg.sty}{height}%
{\setlength{\SVG@out@height}{#1}\SVG@out@useheighttrue}%
```

`extract booleans` The booleans for the extraction formats are defined. Additionally, the global export variable is defined, which is set to `true` whenever any extraction is requested.

```
\define@boolkey[SVG]{svg.sty}[SVG@out@]{pdf}[true]{}%
\define@boolkey[SVG]{svg.sty}[SVG@out@]{eps}[true]{}%
\define@boolkey[SVG]{svg.sty}[SVG@out@]{png}[true]{}%
\newif\ifSVG@out@extract
```

`output definitions` The extraction path, extraction root name, clean boolean, pre-LaTeX commands, and post-LaTeX commands are defined.

```
\def\SVG@out@path{./}%
\define@key[SVG]{svg.sty}{path}{\def\SVG@out@path{#1}}%
\def\SVG@out@name{Fig.\arabic{svgfigure}\alph{subfigure}}%
\define@key[SVG]{svg.sty}{name}{\def\SVG@out@name{#1}}%
\define@boolkey[SVG]{svg.sty}[SVG@out@]{clean}[true]{}%
\def\SVG@out@pretex{}%
\define@key[SVG]{svg.sty}{pretex}{\def\SVG@out@pretex{#1}}%
\def\SVG@out@postex{}%
\define@key[SVG]{svg.sty}{postex}{\def\SVG@out@postex{#1}}%
```

`command definitions` The command options are defined within the prefix `\SVG@cmd` and are set by the key definition following each command definition.

```
\def\SVG@cmd@inkscape{inkscape -z -C}%
\define@key[SVG]{svg.sty}{inkscape}{\def\SVG@cmd@inkscape{#1}}%
\def\SVG@cmd@pdflatex{pdflatex}%
\define@key[SVG]{svg.sty}{pdflatex}{\def\SVG@cmd@pdflatex{#1}}%
\def\SVG@cmd@pdftops{pdftops -eps}%
\define@key[SVG]{svg.sty}{pdftops}{\def\SVG@cmd@pdftops{#1}}%
\def\SVG@cmd@convert{convert -density 300}%
\define@key[SVG]{svg.sty}{convert}{\def\SVG@cmd@convert{#1}}%
```

`process options` All the options for the package are processed, and the SVG counter is defined. The SVG counter is used to correctly handle the `subfigure` counting.

```
\ProcessOptionsX[SVG]%
\newcounter{svgfigure}[figure]%
```

`\setsvg` Define the macro to globally set keys.

```
\def\setsvg#1{\setkeys[SVG]{svg.sty}{#1}}%
```

`\includesvg` Define the macro used to include an SVG. Set the keys and determine if extraction should occur.

```
\def\includesvg{\@ifnextchar[\@includesvg{\@includesvg[]}}%
\def\@includesvg[#1]#2{%
  \setkeys[SVG]{svg.sty}{#1}%
  \SVG@out@extractfalse%
  \ifSVG@out@pdf \SVG@out@extracttrue \fi%
  \ifSVG@out@eps \SVG@out@extracttrue \fi%
  \ifSVG@out@png \SVG@out@extracttrue \fi%
```

Run Inkscape to separate the svg into text and image. Only run Inkscape if the svg is newer than the generated text and image.

```
\ifnum\pdfstrcmp%
{\pdffilemoddate{\SVG@in@path#2.svg}}%
{\pdffilemoddate{\SVG@in@path#2.pdf}}>0%
\immediate\write18{\SVG@cmd@inkscape \space -f\SVG@in@path#2.svg%
  \space-A\SVG@in@path#2.pdf --export-latex}%
\fi%
```

Determine the image width and height using `\includegraphics`.

```
\ifSVG@out@usewidth%
\settoheight\SVG@out@height%
{\includegraphics[width=\SVG@out@width]{\SVG@in@path#2}}%
\else\ifSVG@out@useheight%
\settowidth\SVG@out@width%
{\includegraphics[height=\SVG@out@height]{\SVG@in@path#2}}%
\else%
\settoheight\SVG@out@height{\includegraphics{\SVG@in@path#2}}%
\settowidth\SVG@out@width{\includegraphics{\SVG@in@path#2}}%
\fi%
\fi%
```

Open the output file for extraction.

```
% Open the output file.
\ifSVG@out@extract%
\newwrite\SVG@out@file%
\setcounter{svgfigure}{\value{figure}}%
\stepcounter{svgfigure}%
\def\SVG@out@filename{\SVG@out@name}%
\immediate\openout\SVG@out@file=\SVG@out@path\SVG@out@filename.tex%
\fi%
```

Open and write the preamble. Notice that the catcodes for `#` need to be changed to prevent double expansion when reading the line.

```
\ifSVG@out@extract%
\def\SVG@in@line{}%
\newread\SVG@in@file%
\immediate\openin\SVG@in@file=\SVG@in@preamble%
\fi%
\newif\ifSVG@in@read%
\ifSVG@out@extract \SVG@in@readtrue \fi%
\@whilesw\ifSVG@in@read\fi{%
  \catcode`\#=12\relax\endlinechar=-1%
  \immediate\read\SVG@in@file to \SVG@in@line%
  \ifx\SVG@in@end\SVG@in@line%
  \SVG@in@readfalse%
  \else%
  \immediate\write\SVG@out@file{\unexpanded\expandafter{\SVG@in@line}}%
  \fi%
  \ifeof\SVG@in@file\SVG@in@readfalse\fi%
  \endlinechar=13\catcode`\#=6\relax}%
\ifSVG@out@extract \immediate\closein\SVG@in@file \fi%
```

Now write everything needed after the preamble. This includes requiring the `import` package and defining all the dimensions need to match the document size with the image size.

```
\ifSVG@out@extract%
\def\SVG@out@defpack{\makeatletter%
  \@ifpackageloaded{import}{}{\RequirePackage{import}}%
  \@ifpackageloaded{graphicx}{}{\RequirePackage{graphicx}}%
  \@ifpackageloaded{transparent}{}{\RequirePackage{transparent}}%
  \@ifpackageloaded{xcolor}{}{\RequirePackage{xcolor}}\makeatother}%
\def\SVG@out@defwidth{\def\svgwidth{0.99\textwidth}}%
\def\SVG@out@definput{\import{\SVG@in@path}{#2.pdf_tex}}%
\immediate\write\SVG@out@file{\unexpanded\expandafter{\SVG@out@defpack}}%
\immediate\write\SVG@out@file%
{\noexpand\setlength{\pdfpagewidth}{\the\SVG@out@width}}%
\immediate\write\SVG@out@file%
{\noexpand\setlength{\pdfpageheight}{\the\SVG@out@height}}%
\immediate\write\SVG@out@file%
{\noexpand\setlength{\paperheight}{\pdfpageheight}}%
\immediate\write\SVG@out@file%
{\noexpand\setlength{\paperwidth}{\pdfpagewidth}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\textheight}{\paperheight}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\textwidth}{\paperwidth}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\textheight}{\paperheight}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\oddsidemargin}{-1in}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\evensidemargin}{-1in}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\topmargin}{-1in}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\headheight}{0in}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\headsep}{0in}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\topskip}{0in}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\footskip}{0in}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\parindent}{0in}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\parsep}{0in}}%
\immediate\write\SVG@out@file{\noexpand\setlength{\parskip}{0in}}%
\immediate\write\SVG@out@file{\noexpand\begin{document}}%
  \immediate\write\SVG@out@file{%
    \noexpand\pagestyle{empty}%
    \noexpand\begin{center}%
      \unexpanded\expandafter{\SVG@out@defwidth}%
      \unexpanded\expandafter{\SVG@out@pretex}%
      \expandafter\noexpand\SVG@out@definput%
      \unexpanded\expandafter{\SVG@out@postex}%
      \noexpand\end{center}}%
  \immediate\write\SVG@out@file{\noexpand\end{document}}%
\immediate\closeout\SVG@out@file%
\fi%
```

Run LaTeX on the extracted file and create the PDF.

```
\ifSVG@out@extract%
\immediate\write18{\SVG@cmd@pdflatex\space\SVG@out@path\SVG@out@filename.tex}%
\fi%
```

Convert the PDF to EPS if requested.

```
\ifSVG@out@eps%
\immediate\write18{\SVG@cmd@pdftops\space\SVG@out@filename.pdf}%
\immediate\write18{mv \SVG@out@filename.eps%
  \space\SVG@out@path\SVG@out@filename.eps}%
\fi%
```

Convert the PDF to PNG if requested.

```
\ifSVG@out@png%
\immediate\write18{\SVG@cmd@convert\space\SVG@out@filename.pdf%
  \space\SVG@out@filename.png}%
\immediate\write18{mv \SVG@out@filename.png%
  \space\SVG@out@path\SVG@out@filename.png}%
\fi%
```

Clean up if requested.

```
\ifSVG@out@extract%
\ifSVG@out@pdf%
\immediate\write18{mv \SVG@out@filename.pdf%
  \space\SVG@out@path\SVG@out@filename.pdf}%
\else \ifSVG@out@clean \immediate\write18{rm \SVG@out@filename.pdf} \fi%
\fi%
\ifSVG@out@clean%
\immediate\write18{rm \SVG@out@path\SVG@out@filename.tex%
  \space\SVG@out@filename.aux \SVG@out@filename.log \SVG@out@filename.out}%
\fi\fi%
```

Finally, include the SVG in the current document and end the package.

```
\ifSVG@in@exclude \else {\def\svgwidth{\the\SVG@out@width}%
\SVG@out@pretex\import{\SVG@in@path}{#2.pdf_tex}\SVG@out@postex} \fi%
}%
```

6 Thanks

Thanks to my lovely wife Éadaoin for being a very patient beta tester and important collaborator. Thanks is also due to J. Engelen for creating this functionality within INKSCAPE, and of course to all the developers of INKSCAPE.