PSfragX: one graphic in one file *

Pascal Kockaert

2012/05/02

Abstract

\usepackage[*options*]{pfragx} inputs the packages psfrag and graphicx, and adds essentially one LaTeX command, which is \includegraphicx (with an 'x´ at the end).

This command differs from \includegraphics in the fact that it inputs \psfrag replacements contained into the included EPS file itself.

If the EPS files contains those replacements at the right place, \usepackage[sub]{psfragx} will substitute \includegraphicx to \includegraphics automatically. At the same time, it is possible to include overpic commands into the EPS file, and they will be automatically processed.

The EPS file can be written by a matlab script, so that the user needs only to call the script in order to print the matlab figure. No additional work will be necessary.

Contents

---

* This file has version v1.1. It was processed on 2012/05/02.

1   Warning

Some options of this package allow to overwrite some files ending in .pfx and in .ovp. Be sure to understand how these options work before using them.

The text below assumes that you are used to \includegraphics from the graphicx package, and also to \psfrag from the psfrag package. Reading the documentation of overpic could also help to understand what follows.

2   Motivation

Using graphics drawn by mathematical softwares is very convenient but does not offer all the flexibility of TeX and LaTeX when it comes to write labels.

Some solutions exist, like the matlab laprint.m function (http://www.uni-kassel.de/fb16/rat/matlab/laprint/) to print from matlab(TM) into EPS files suited to be easily handled by psfrag. All the labels (including numbers on the axis) are converted into strings like 'x01´ that should be replaced by their values, like '3.141'.

Though the result is pleasant, it is MANDATORY to keep track of the substitutions. This is why the laprint.m function takes care to write a TeX file that contains all the \psfrag commands necessary to obtain the original

labels. This TeX file can be edited to modify the \psfrag commands. This scheme works well, but has some limitations. You must obviously take care to move the .tex and the .eps file together. But in addition, you must input the graphic using an \input command. If you intend to modify its size or change some \psfrag replacements, you need to open and modify the original .tex file. If you want to use packages like overpic, you must modify the .tex file output by laprint or copy all the psfrag replacements that it contains to your main TeX file. In case you would choose to copy the psfrag replacements into your main TeX file, you will end up with a lot of lines like

```
\psfrag{x01}[B][B][1][0]{3.141}
\psfrag{x02}[B][B][1][0]{6.283}
\psfrag{x03}[B][B][1][0]{9.425}
\psfrag{x04}[B][B][1][0]{12.566}
```

into your LaTeX document.

In fact, you DO NOT NEED to see all these lines, and should never see them, except, for example, if you want to replace them with

```
\psfrag{x01}[B][B][1][0]{$\pi$}
\psfrag{x02}[B][B][1][0]{$\2\,\pi$}
```

and so on.

What is said here about \psfrag commands can be transposed to the overpic environment that allows to put picture objects over a graphic.

The authors of psfrag have designed a mechanism that allows to embed \psfrag commands into the EPS file itself[1]. Though this mechanism can be convenient, it presents some drawbacks that are described into the documentation of psfrag.

The package psfragX aims to circumvent these drawbacks, as well as to introduce more flexibility into the automatic inclusion mechanism. For example, psfragX allows to define different \psfrag replacements for different languages. If babel is used, replacements will be selected according to the current language of the document. It also allows to make use of color commands that are ignored if the color package is not loaded.

3 How PSfragX works

3.1 PSfrag, PFX, overpic and OVP

The package psfragx allows to embed \psfrag commands into the EPS file, as well as picture objects in overpic environments. In order to sim-

---

[1]This was kindly reminded to me by Michael C. Grant, one of the authors of psfrag, that I would like to thank here.

plify the description, we will refer only to \psfrag inclusions here below. The overpic inclusion mechanism works in the same way and will not be described. The differences between psfrag and overpic inclusions will appear in the syntax of some commands. We use the three letters pfx to prefix things that relate to psfrag and ovp to prefix things that relate to overpic.

The psfragx mechanism can be divided into two parts that are described separately.

3.2 Merging and separating the TeX and EPS documents

We use the result of laprint.m as an example, but all the EPS file could be processed in the same way. laprint outputs Fig1.tex & Fig1.eps files that can be converted into a single file that we call Figure1.eps, which is a copy of Fig1.eps, with additional comments that contain all the interesting lines of Fig1.tex. At this stage, you can throw the original files Fig1.tex and Fig1.eps. BE CAREFUL TO MAKE A BACKUP!!!

The added comments are not read by the PostScript interpreter and should not affect the resulting EPS file. As far as I know, the comments used conform to the Adobe(TM) Document Structuring Convention (ADSC). These comments can be added by hand or using the matlab script psfragx.m that should be accompanying this file. Their structure also conforms to the DocStrip convention (see docstrip.dtx): the part to be copied starts with a comment %<*pfx> and ends with %</pfx>. All the lines between these marks will be taken into account by psfragx.

```
%<*pfx>
%\psfrag{x01}[B][B][1][0]{3.141}
%\psfrag{x02}[B][B][1][0]{6.283}
%\psfrag{x03}[B][B][1][0]{9.425}
%\psfrag{x04}[B][B][1][0]{12.566}
%</pfx>
```

PSfragx looks for these lines into the EPS file and outputs them into a file with the same name, but a .pfx extension. In other words, Figure1.pfx is created with the comments of Figure1.eps. This file is normally created only once, though there is an option to overwrite it. This means that you could edit it by hand without loosing your work next time you run LaTeX.

In the same manner, it is possible to include picture commands using

```
%<*ovp>
%\put(50,50){Middle of the graphic}
%</ovp>
```

These lines will go into the file `Figure1.ovp`.

The process of seeking for `pfx` and `ovp` environments stops as soon as a line starting with `%\endinput` is found. Including such a line will speed the things up.

3.3 Input the right file at the right moment

Once the `Figure1.pfx` file exists, the command `\includegraphicx` includes it and uses the conventional `\includegraphics` from `graphicx` with the same arguments. The result is that all the `\psfrag` replacements are processed before the `Figure1.eps` file is included.

The `\psfrag` commands do not appear into the TEX file. The PFX file could be deleted, and all the replacements would still be performed, because the PFX file would be re-generated on the fly.

Now that all the labels on the axes are perfectly drawn, we could still want to replace the value `3.141` by the tag `$\pi$`. This is why the `\includegraphicx` command has a second facultative argument. Inside this argument, you should issue all the `\psfrag` commands that you want to perform after the inclusion of the PFX file.

```
\includegraphicx[width=\linewidth]
                (\psfrag{x01}[B][B]{$\pi$}%
                 \psfrag{x02}[B][B]{$2\,\pi$})
                {Figure1.eps}
```

The second optional argument is defined with () to avoid an interaction between the brackets of the `\psfrag` command and those of the `\includegraphicx` command.

All that is said about PFX files and `psfrag` replacements can be transposed to OVP files and `overpic picture` commands.

If we want to add `overpic` commands before or after the inclusion of the OVP file, we can use the two other optional arguments of `\includegraphicx`:

```
\includegraphicx[width=\linewidth]
                <\put(50,50){Foreground object}>
                [\put(0,0){Background object}]
                {Figure1.eps}
```

4 Usage

4.1 Package options

The package is input by `\usepackage[`*options*`]{psfragx}`

The *options* are

h!

Table 1: Meaning of the keys for \includegraphics and overpix

key	acceptable values	action
graphicx keys	*usual values*	*usual meanings*
pfx	true/false	allows/disallows the inclusion of the PFX file
overwritepfx	true/false	allows/disallows to overwrite an existing PFX file
pfxadd	*psfrags*	\psfrag commands to be processed after the inclusion of the PFX file
ovp	true/false	allows/disallows the inclusion of the OVP file
overwriteovp	true/false	allows/disallows to overwrite an existing OVP file
ovpbgd	*picture commands*	picture commands to be processed before the inclusion of the OVP file
ovpfgd	*picture commands*	picture commands to be processed after the inclusion of the OVP file

sub, nosub substitute or do not substitute \includegraphicx to \includegraphics;

allcom,selcom copy all or only selected MetaComments from the EPS file to the PFX file (if you do not understand what this means, you can safely ignore it);

ovp, noovp makes use of overpic to (automatically) put picture objects over the graphics.

4.2 Two new commands

\includegraphicx

overpix

- THE command of the package is:

  \includegraphicx [*keys*]
  (*psfrags*)
  <*foreground overpic*>
  [*background overpic*]
  {*file.eps*}

- THE environment of the package is:

  \begin{overpix} [*keys*]
  <*foreground overpic*>
  [*background overpic*]
  {*file.eps*}

  \end{overpix}

The meaning of the keys is explained in table 1.

The item denoted by *psfrags* should consist only in \psfrag{A}[b][c][d][e]{F} commands, where A,b,c,d,e,F can be anything. In addition, the \psfrag commands can be selectively included according to the current language of the document (at the point of inclusion). Two commands are provided. The commands \iflanguage is explained in the babel documentation. If babel is not loaded, \iflanguage is redefined to match the definition of babel, that is:

\iflanguage{*languagename*} {*true case*} {*false case*}. The configuration file psfragxcfg given below, as an example, redefines the main commands

6

of the `color` package so that no error occurs if the *psfrags* contains color commands and the package `color` is not loaded.

The item denoted by *picture commands* should consist only in commands that are allowed in the usual `picture` environment of LaTeX. You can also make use of `\iflanguage` and color commands, provided that the configuration file given below is used.

YOU CANNOT PUT BLANK LINES, that is lines that would consist only in one "%" sign. If you insert such lines, the "%" sign will be removed and some space will be added in front of the included figure.

4.3 Other new commands

Though they are not needed in a normal use of `psfragx`, the following commands are available: `\allmetacomments`, `\selectedmetacomments`, `\copypfxfromto{<EPS file>}{<PFX file>}`, `\setpfxinput{<File>}`, `\setpfxoutput{<File>}` `\copypfxlines`, `\pfxinput`, `\ovpinput`. Their usage can be deduced from the commented source code.

The other commands are internal and start with `\pfx@` or `\ovp@`.

5 Configuration file

The file `psfragx.cfg` will be input by psfrags, if it exists. This file can contain new commands of general use, or commands that must appear just before or just after the inclusion of the PFX/OVP file occurs. To this aim, four commands can be defined. Their names are `\Beforepfxinput`, `\Afterpfxinput`, `\Beforeovpinput`, and `\Afterovpinput`. They can be used as in the example below.

```
 1 ⟨∗cfg⟩
 2 % Example of configuration file for psfragx.sty
 3 % The macros \Beforepfxinput, \Afterpfxinput
 4 % \Beforeovpinput, and \Afterovpinput are executed
 5 % into a group. They should not define global commands to
 6 % avoid side effects.
 7 %
 8 %
 9 % The command \providecolorcommands defines commands that
10 % take the same arguments as the mains commands of the
11 % color package, in case this package is not loaded.
12 %
13 \newcommand{\providecolorcommands}
14         {\def\pfx@gobble@two##1##2{\typeout{Some psfragx
15                                   replacement would appear
16                                   in color ##1{##2}
17                                   if the color package was
18                                   loaded!!!}}%
19         \def\pfx@gobble@three@fbox##1##2##3{\typeout{Some psfragx
```

```
20                                                              replacement would
21                                                              appear
22                                                              in color ##1{##2}
23                                                              and others
24                                                              in color ##1{##3}
25                                                              if the color package
26                                                              was loaded!!!}%
27                                                              \fbox}%
28              \def\pfx@fm@to@mm##1##2##{\csname ##1\endcsname{##2}}%
29              \expandafter\ifx\csname textcolor\endcsname\relax
30                \def\textcolor{\pfx@fm@to@mm{pfx@gobble@two}}\fi
31              \expandafter\ifx\csname color\endcsname\relax
32                \def\color{\pfx@fm@to@mm{pfx@gobble@two}}\fi
33              \expandafter\ifx\csname colorbox\endcsname\relax
34                \def\colorbox{\pfx@fm@to@mm{pfx@gobble@two}}\fi
35              \expandafter\ifx\csname fcolorbox\endcsname\relax
36                \def\fcolorbox{\pfx@fm@to@mm{pfx@gobble@three@fbox}}\fi
37            }
38 % The name of the next four commands are specific to psfragx
39 \def\Beforepfxinput{\providecolorcommands}
40 \def\Afterpfxinput{}
41 \def\Beforeovpinput{\providecolorcommands}
42 \def\Afterovpinput{}
43 ⟨/cfg⟩
```

6   Example of tagged EPS file

We provide here below an example of EPS that uses the language and color features... According to the "Adobe Document Structuring Convention" (ADSC), comments starting with two percent signs have a special meaning. You should therefore avoid to put EXACTLY two percents signs at the beginning of a line. If you respect this rule and avoid very long lines, you should never broke your EPS file.

```
44 ⟨∗example⟩
45 %%!PS-Adobe-2.0 EPSF-1.2
46 %%Creator: Adobe Illustrator(TM) 1.2d4
47 %%Title: tiger.eps
48 %%CreationDate: 4/12/90 3:20 AM
49 %%BoundingBox: 17 171 567 739
50 %<*pfx>
51 %\psfrag{T}[B][B]{\fcolorbox{white}{black}{\color{white}Title}}
52 %\psfrag{t}[t][t]{time (s)}
53 %\psfrag{I}[b][b]{I (W)}
54 %\iflanguage{french}
55 %          {\psfrag{T}[B][B]{Title}%
56 %           \psfrag{t}[t][t]{temps (s)}}
57 %          {}
```

```
58  %\psfrag{T}[B][B]{Title}
59  %</pfx>
60  %<*ovp>
61  %\put(0,80){(a)}
62  %</ovp>
63  %\endinput
64  %%EndComments
65  %%
66  %% [The code of the {\EPS} file should come HERE]
67  %%
68  %% End
69  ⟨/example⟩
```

Full examples should be provided with this package. They are not included into `psfrags.dtx`.

7 Associated matlab scripts

The script `psfragx.m` is written for matlab and can be used in conjunction with `laprint.m` (see URL above) in order to benefit from the advantages of `laprint.m` and mix the resulting .tex and .eps files into a file that contains all the information.

The scripts `pfxprint.m` can be used with the same syntax as `laprint.m` (see documentation of `laprint`). This script invoques `laprint` with the settings contained in the file `laprpfx.mat`, and immediately after, it merges the generated EPS and TEX files. Therefore, you should ensure that the files `laprint.m`, `laprpfx.mat`, and `psfragx.m` are in a directory searched by matlab before using the `pfxprint` command.

At the time of writing, the current version of `laprint` is `3.16`. This version works well with `pfxprint.m` and `psfragx.sty`.

8 Credits

All the code to extract the comments from the EPS file is inspired from `docstrip`. The set of commands was reduced to its minimum, and a `\pfx@` prefix was added to all the commands, in order to avoid any interaction with other packages.

9 Mise en œuvre

```
70  ⟨*package⟩
```

Almost all the internal commands start with `\pfx@`, `\ovp@`, `\ifpfx@`, or `\ifovp@`. Two exceptions are `\@..@overpix` and `\@..@igx`, where `@..@` can be `@`, `@@`, `@@@` or `@@@@`.

9.1 Required packages and options

We offer the option to substitute the new `\includegraphicx` command to the usual `\includegraphics`, and optionally, the overpix environment to the usual overpic one. This could broke things but allows to use `psfragx` with existing documents almost transparently.

`\pfx@subfalse`
`\pfx@subtrue`

```
71 \DeclareOption{sub}{\pfx@subtrue}
72 \DeclareOption{nosub}{\pfx@subfalse}
```

The next option was of some help to debug this package. With `allcom`, all the lines of the EPS file starting with `%%` are copied to the PFX and OVP files. Otherwise, these lines are not copied if they are out of a tagged environment.

```
73 \DeclareOption{allcom}{\allmetacomments}
74 \DeclareOption{selcom}{\selectedmetacomments}
```

The next option specifies that the overpic environment will be used. Therefore, the overpic package should be loaded.

`\pfx@ovptrue`
`\pfx@ovpfalse`

```
75 \DeclareOption{ovp}{\pfx@ovptrue}
76 \DeclareOption{noovp}{\pfx@ovpfalse}
```

We define the new commands needed to process the options.

`\allmetacomments`
`\selectedmetacomments`

```
77        \newif\ifpfx@sub\pfx@subfalse
78        \newif\ifpfx@ovp\pfx@ovptrue
79        \newif\ifpfx@metacomments
80        \pfx@metacommentsfalse
81        \def\allmetacomments{\pfx@metacommentstrue}
82        \def\selectedmetacomments{\pfx@metacommentsfalse}
```

Finally, default options are defined.

```
83 \ExecuteOptions{sub,ovp,selcom}
84 \ProcessOptions*
```

Now, we load the other packages.

```
85 \RequirePackage{graphicx}
86 \RequirePackage{psfrag}
```

The overpic package is not loaded if this was required by the user. Otherwise, we load this package. To ensure proper placement of the objects put into the picture environment, we must always use the same option when loading `overpic`. We choose this option to be `percent`.

```
87 \ifpfx@ovp \RequirePackage[percent]{overpic} \fi
```

9.2 Reading the EPS file and writing PFX or OVP files

The code that follows is highly inspired from that of docstrip.tex.

9.2.1 Copying selected lines from the EPS file

Below, we write the code to copy specific lines contained in the EPS file into an auxiliary file. Comments (single % sign) in front of these lines are automatically removed.

First, we define a few macros of general use.

```
\pfx@gobble
\pfx@percent
\pfx@doublepercent
88 \def\pfx@gobble#1{}
89 {\catcode`\%=12
90  \gdef\pfx@percent{%}
91  \gdef\pfx@doublepercent{%%}
92 }
```

Here we define the extension of the auxiliary file, and the name of the tag associated to this file. The `metaprefix` replaces double percent signs found into the original EPS file.

```
\pfx@ext
\pfx@tag
\pfx@metaprefix
93 \let\pfx@metaprefix\pfx@doublepercent
94 \def\pfx@tag{pfx}
95 \def\pfx@ext{pfx}
96 \def\pfx@tmp{}
```

We also need to define the string after which we will stop to scan the EPS file. This string must appear at the beginning of a line. If this string is not present into the EPS file, the file will be scanned up to the end. This string is defined to be `%\endinput`.

```
\pfx@endinput
97 \edef\pfx@endinput
98     {\pfx@percent\expandafter\pfx@gobble\string\\endinput}
```

And now, we copy the needed code from `docstrip`, with some modifications to throw the leading percent sign when we copy the lines that appear between two tags `<*pfx>...</pfx>`.

We define a command to change catcodes,

```
99 \def\pfx@makeother#1{\catcode`#1=12\relax}
```

another to copy a given token,

```
100 \def\pfx@iden#1{#1}
```

and a few boolean variables.

```
101 \newif\ifpfx@continue
102 \newif\ifpfx@outputtofile
```

The names of the input and output files are contained into the internal variables `\pfx@infile` and `\pfx@outfile`. These named can be accessed from the document through the two commands `\setpfxinput` and `\setpfxoutput`.

`\setpfxinput`
`\setpfxoutput`

```
103 \def\setpfxinput#1{\gdef\pfx@infile{#1}}
104 \def\setpfxoutput#1{\gdef\pfx@outfile{#1}}
105 \gdef\pfx@infile{} \gdef\pfx@outfile{}
```

Two streams are reserved by `psfragx`. I do not know if I should use them locally rather than globally.

```
106 \newread\pfx@in
107 \newwrite\pfx@out
```

The macro `\copypfsfromto` DOES NOT CHECK that the input file exists.

`\copypfxfromto`

```
108 \def\copypfxfromto#1#2{%
109 \setpfxinput{#1}%
110 \setpfxoutput{#2}%
111 \copypfxlines%
112 }
```

The macro `\copypfxlines` does the real job. See `docstrip` to understand how it works.

`\copypfxlines`

```
113 \def\pfx@ignorespaces{\ignorespaces}%
114 \def\copypfxlines{% input and output files are global names
115 \immediate\openin\pfx@in\pfx@infile\relax \ifeof\pfx@in
116        \errmessage{psfragx tried to read from a file that
117                   does not exist. This seems to be a bug!}%
118 \else
119        \immediate\openout\pfx@out=\pfx@outfile\relax
120        \immediate\write\pfx@out{\pfx@ignorespaces}
121        \ifeof\pfx@out
122             \begingroup
123             \pfx@makeother\ \pfx@makeother\\\pfx@makeother\$%
124             \pfx@makeother\#\pfx@makeother\^\pfx@makeother\^^K%
125             \pfx@makeother\_\pfx@makeother\^^A\pfx@makeother\%%
126             \pfx@makeother\~\pfx@makeother\{\pfx@makeother\}%
127             \pfx@makeother\&\endlinechar-1\relax
128             \loop
129                  \read\pfx@in to \pfx@inline
130                  \ifx\pfx@inline\pfx@endinput
131                       \pfx@continuefalse
132                       \typeout{psfragx: \pfx@percent
133                              \expandafter\pfx@gobble
134                                   \string\\endinput was
```

```
135                              found in \pfx@infile.}%
136                  \else
137                      \ifeof\pfx@in
138                          \pfx@continuefalse
139                      \typeout{psfragx: End of file
140                          \pfx@infile was reached.}%
141                      \else
142                          \pfx@continuetrue
143                          \expandafter\pfx@processline
144                          \pfx@inline\pfx@endline
145                      \fi%
146                  \fi%
147          \ifpfx@continue
148          \repeat
149          \endgroup
150      \else
151          \errmessage{psfragx: output file already exists!}%
152      \fi %\pfx@out
153      \immediate\closeout\pfx@out
154 \fi %\pfx@in
155 \immediate\closein\pfx@in
156 }
```

At this stage, all `<pfx>` and `<*pfx>...</pfx>` lines from `\pfx@infile` should be in `\pfx@outfile`.

Each time a new line is found by the previous macro, the line is processed using `\pfx@processline`. This macro scans the beginning of the line and defers the treatment to the right macro. In the `docstrip` code, normal lines are copied without change. In our code, the leading percent of copied lines is removed.

`\pfx@processline`

```
157 \def\pfx@normalline#1\pfx@endline{%
158      \def\pfx@inline{#1}%
159      \ifpfx@outputtofile%
160          \immediate\write\pfx@out{\pfx@inline}%
161      \fi%
162 }
163 %
164 \def\pfx@removecomment#1\pfx@endline{%
165      \def\pfx@inline{#1}%
166      \ifpfx@outputtofile%
167          \immediate\write\pfx@out{\pfx@inline}%
168      \fi%
169 }
170 %
171 \bgroup\catcode`\%=12 \pfx@iden{\egroup
172 \def\pfx@putmetacomment%}#1\pfx@endline{%
173      \edef\pfx@inline{\pfx@metaprefix#1}%
```

```
174        \ifpfx@metacomments
175                \immediate\write\pfx@out{\pfx@inline}%
176        \else
177                \ifpfx@outputtofile
178                        \immediate\write\pfx@out{\pfx@inline}%
179                \fi
180        \fi
181 }
182 %
183 \begingroup
184 \catcode`\%=12 \catcode`\*=14 \gdef\pfx@processline#1{*
185        \ifx%#1*
186                \expandafter\pfx@processlinex
187        \else
188                \expandafter\pfx@normalline
189        \fi
190 #1}
191 \endgroup
192 %
193 \begingroup
194 \catcode`\%=12 \catcode`\*=14
195 \gdef\pfx@processlinex%#1{*
196        \ifcase\ifx%#10\else
197                \ifx<#11\else2\fi\fi\relax
198        \expandafter\pfx@putmetacomment\or
199        \expandafter\pfx@checkoption\or
200        \expandafter\pfx@removecomment\fi
201 #1}
202 \endgroup
203 %
204 \def\pfx@checkoption<#1{%
205        \ifcase\ifx*#10\else
206                \ifx/#11\else2\fi\fi\relax
207        \expandafter\pfx@staroption\or
208        \expandafter\pfx@slashoption\or
209        \expandafter\pfx@tagoption\fi
210 #1}
211 %
212 \def\pfx@staroption*#1>#2\pfx@endline{%
213        \def\pfx@tmp{#1}%
214        \ifx\pfx@tmp\pfx@tag
215                \pfx@outputtofiletrue
216        \fi
217 }
218 %
219 \def\pfx@slashoption/#1>#2\pfx@endline{%
220        \def\pfx@tmp{#1}%
221        \ifx\pfx@tmp\pfx@tag\relax
222                \pfx@outputtofilefalse
```

14

```
223        \fi
224 }
225 %
226 \def\pfx@tagoption#1>#2\pfx@endline{%
227        \def\pfx@tmp{#1}%
228        \ifx\pfx@tmp\pfx@tag\relax
229                \def\pfx@inline{#2}%
230                \immediate\write\pfx@out{\pfx@inline}%
231        \fi
232 }
```

This ends the code to read EPS file and write PFX file. It is clear that
only `\pfx@tag` and `\pfx@ext` should be changed from pfx to ovp in order
to process `overpic` inclusions rather than `psfrag` replacements.

9.3   Code that inputs the PFX and OVP files

This code will add commands to input the PFX and OVP files if they exist.
If they do not, they will be created on the fly and read just after. An option
allows to ignore existing files and generate PFX and OVP files from the EPS
file each time the EPS file is included.

At first, we define commands related to PFX files. Later on, we will
adapt them to OVP files.

9.3.1   Saving and providing commands of other packages

We save the commands that could be redefined later

```
233 \let\pfx@includegraphics=\includegraphics
234 \let\pfx@overpic=\overpic
235 \let\pfx@endoverpic=\endoverpic
```

Even if `overpic` is not loaded, the `overpic` environment should exist.
In this case, the OVP files will not be processed, and no `picture` element
should be put over the graphics. Nonetheless, `\includegraphicx` is defined
to always use the `overpic` environment.

Therefore, we provide a definition of the `overpic` environment that is
partially copied from overpic.sty. We have removed all the code that makes
computations about the size and the position of the grid.

`\pfx@overpic`
`\pfx@endoverpic`
```
236 \@ifundefined{pfx@overpic}{%
237                 \newcommand*{\pfx@overpic}[2][]
238                 {\sbox{\z@}{\includegraphics[#1]{#2}}%
239                 \settodepth{\@tempcnta}{\usebox{\z@}}%
240                 \settoheight{\@tempcntb}{\usebox{\z@}}%
241                 \advance\@tempcntb\@tempcnta%
242                 \settowidth{\@tempcnta}{\usebox{\z@}}%
243                 \begin{picture}(\@tempcnta,\@tempcntb)%
```

```
244                              \put(0,0){\makebox(0,0)[bl]{\usebox{\z@}}}}%
245                           }{}
246 \@ifundefined{pfx@endoverpic}{\def\pfx@endoverpic{\end{input}}}{}
```

\iflanguage   We also have to provide \iflanguage command, in case babel is not loaded. We could have simplified the code, because \pfx@iflanguage should always expand to \@secondoftwo if babel is not loaded. Because this code was also copied from the babel package with some changes, we use it even if babel is loaded. This could cause problems if the internal command l@*language* of babel was redefined. Though we redefine a babel command, this should cause no major problem, because the command \iflanguage will be provided only at time of the file inclusion. This means that \psfrag replacements should contain no reference to \iflanguage as these commands will be evaluated after the file is read. The command \iflanguage should be evaluated at time of inclusion, in order to decide which psfrag or picture commands are to be taken into account.

\onlylanguage   In order to simplify the writing of multilingual EPS files, we also provide
\endonlylanguage   the command \onlylanguage {*language*}... \endonlylanguage, which argument is read only if the current language of the document is *language*.

In order to simplify the writing of multilingual EPS files, we also provide the command \onlylanguage {*language*}... \endonlylanguage, which argument is read only if the current language of the document is *language*.

\pfx@iflanguage
\iflanguage
\onlylanguage
\endonlylanguage
\pfx@save@iflanguage
\pfx@restore@iflanguage
\pfx@firstoftwo
\pfx@secondoftwo
```
247 \long\def\pfx@firstoftwo#1#2{#1\ignorespaces}%
248 \long\def\pfx@secondoftwo#1#2{#2\ignorespaces}%
249 \def\pfx@iflanguage#1{%
250     \ifnum\csname l@#1\endcsname=\language
251       \expandafter\pfx@firstoftwo
252     \else
253       \expandafter\pfx@secondoftwo
254     \fi}
255 \long\def\onlylanguage#1#2\endonlylanguage{\pfx@iflanguage{#1}{#2}{}\ignorespaces}
256 \def\pfx@save@iflanguage{\let\save@pfx@iflanguage=\iflanguage%
257                          \let\iflanguage=\pfx@iflanguage}
258 \def\pfx@restore@iflanguage{\let\iflanguage=\save@pfx@iflanguage}
```

The two commands \pfx@save@iflanguage and \pfx@restore@iflanguage will be called just before and after the inclusion of the PFX file.

9.3.2   New commands to read and write files

We start with some declarations (new commands and new if) The names of the ifGin series are chosen to be easily processed through the keyval package mechanism. Gin is the prefix used by the graphicx package.

16

\ifpfx@generate
\ifovp@generate
\pfx@add
\ovp@add@bgd
\ovp@add@fgd
\ifGin@pfx
\ifGin@overwritepfx
\ifGin@ovp
\ifGin@overwriteovp

```
259 \newif\ifpfx@generate
260 \newif\ifovp@generate
261 \newcommand*\pfx@add{}
262 \newcommand*\ovp@add@bgd{}
263 \newcommand*\ovp@add@fgd{}
264 \newif\ifGin@pfx
265 \newif\ifGin@overwritepfx
266 \newif\ifGin@ovp
267 \newif\ifGin@overwriteovp
```

As the names indicate, these macros are attached to PFX or OVP inclusions. They allow to save information to know if a PFX/OVP file is to be generated, if the PFX/OVP automatic inclusion mechanism is to be used and if existing PFX/OVP files should be overwritten. Finally, three commands will contain the `\psfrag` commands (`\pfx@add`) to be issued after the inclusion of the PFX file, as well as `picture` commands to be issued before (`\ovp@add@bgd`) and after (`\ovp@add@fgd`) the OVP file inclusion.

The new keys will be available through the optional arguments of `\includegraphicx`. This is why they are defined as belonging to the same group as the `graphicx` keys: `Gin`.

The role that we have described for the previous commands is assigned here below. As is common, the boolean keys are set to be true if they are invoked without argument.

```
268 \define@key{Gin}{pfx}[true]%
269          {\lowercase{\Gin@boolkey{#1}}{pfx}}
270 \define@key{Gin}{overwritepfx}[true]%
271          {\lowercase{\Gin@boolkey{#1}}{overwritepfx}}
272 \define@key{Gin}{pfxadd}[]%
273          {\def\pfx@add{#1}}
274 \define@key{Gin}{ovp}[true]%
275          {\lowercase{\Gin@boolkey{#1}}{ovp}}
276 \define@key{Gin}{overwriteovp}[true]%
277          {\lowercase{\Gin@boolkey{#1}}{overwriteovp}}
278 \define@key{Gin}{ovpbgd}[]%
279          {\def\ovp@add@bgd{#1}}
280 \define@key{Gin}{ovpfgd}[]%
281          {\def\ovp@add@fgd{#1}}
```

We will define a handy syntax for the `\includegraphicx` command. This command will mainly convert some of its optional arguments to keys `pfxadd={`*argument*`}`, `ovpbgd={`*argument*`}`, and `ovpfgd={`*argument*`}`.

It is now time to define the commands that will test for the existence of the input and output files and decide if an output file is to be generated. This command makes use of values defined previously for PFX files. This is why we prefix the command with `\pfx`. To understand the code below, it is important to know that the command `\filename@parse{}` defines three commands that are `\filename@area`, `\filename@base` and

`\filename@ext.`

`\pfxinput`

```
282 \newcommand*{\pfxinput}[1]{%
283   \filename@parse{#1}%
284   \IfFileExists{\filename@base.\pfx@ext}
285             {\pfx@generatefalse}
286             {\pfx@generatetrue}%
287   \ifGin@overwritepfx\pfx@generatetrue\fi
288   \IfFileExists{#1}{}{\pfx@generatefalse}%
289   \ifpfx@generate%
290       \copypfxfromto{\filename@area\filename@base.\filename@ext}
291                     {\filename@base.\pfx@ext}%
292   \fi%
293   \pfx@save@iflanguage
294   \csname Before\pfx@tag input\endcsname
295   \InputIfFileExists{\filename@base.\pfx@ext}
296             {\typeout{psfragx: reading commands from
297                                   \filename@base.\pfx@ext}}
298             {\typeout{psfragx: I was not able to read psfrag
299                         definitions from
300                         \filename@base.\pfx@ext}}%
301   \csname After\pfx@tag input\endcsname
302   \pfx@restore@iflanguage
303 }
```

As was announced, we determine if the output file exists. In case this file exists we decide not to generate the output file. If the user required that the output file be overwritten, we ask to generate the output file in any case. Then we test if the input file exists. If not, we cannot generate the ouptut file.

Now that the existence of the input file has been checked, we can call the low level command `\copypfxfromto`. This completes the first step.

The second step is to input the PFX file if it exists. The file is input inside a `\pfx@save@iflanguage` `\pfx@restore@iflanguage` pair. In addition, the commands `\Beforepfxinput` and `\Afterpfxinput` are issued if they exist. Otherwise, they expand to `\relax`. These commands should be defined into the `psfragx.cfg` file in order to customise the behaviour of `psfragx`.

Now, we define variations of `\pfx@` commands in order to work with the `overpic` environment rather than with `psfrag` replacements.

We redefine the tag and the extension, copy the overwrite permission and call `\pfxinput`.

`\ovpinput`

```
304 \newcommand*{\ovpinput}[1]{%
305 \begingroup
306       \def\pfx@ext{ovp}%
307       \def\pfx@tag{ovp}%
```

```
308            \ifGin@overwriteovp\Gin@overwritepfxtrue
309                            \else\Gin@overwritepfxfalse\fi
310          \pfxinput{#1}%
311 \endgroup}
```

9.4   The main command of this package

Here comes the definition of the main command of this package, as seen by the user : `\includegraphicx`. This new command will make use of the new environment called `overpix`, in order to include the graphics.

9.4.1   Internal commands

First, we define two internal commands that perform the required task. Then we define external commands with optional arguments.

`\pfx@includegraphicx`   The macro `\pfx@includegraphics` is just a shortcut to acces the `overpix` environment. We do not call `\begin{pfx@overpix}`... `\end{pfx@overpix}`, in order to save time.

```
312 \def\pfx@includegraphicx#1#2{%
313          \mbox{\pfx@overpix{#1}{#2}\endpfx@overpix}}
```

`\ovp@box@tmp`   In what follows, we need a temporary box. This is called `\ovp@box@tmp`.

```
314 \newbox{\ovp@box@tmp}%
```

`\pfx@overpix`   The environment `overpix` has the same syntax as the `overpic` one. You
`\endpfx@overpix`   can notice that the original version of `\includegraphics` is used. This is important if we decide, later, to let `\includegraphics` be equivalent to `\includegraphicx`. The `\psfrag` commands and the `picture` commands are processed inside this environment. To avoid side effects of command redefinitions inside the included files, we enclose the contents of `overpix` inside a `\begingroup` `\endgroup` pair.

The graphic is included via the original or the lightened version of the `overpic` environment.

All we do is to evaluate the keys of the first argument, then input the PFX file, and process the contents of `\pfx@add`. Thereafter, we call the original or lightened version of the `overpic` environment. We read the keys again and add the background layer of the picture environment, then the layer contained into the OVP file.

When the `\pfx@overpix` command is issued, we end up into a `picture` environment that constitutes yet another layer. Finally, the `picture` environment should be closed by an `\endpfx@overpix` command. Before doing so, the foreground layer of the picture environment is drawn.

It is mandatory to take care that not spurious space is added at the end of the lines. A percent sign should appear each time the line is ending with something else than a command name.

19

```
315 \def\pfx@overpix#1#2{%
316         \begingroup%
317         \begin{lrbox}{\ovp@box@tmp}%
318         \let\includegraphics=\pfx@includegraphics%
319         \Gin@pfxtrue%
320         \Gin@overwritepfxfalse%
321         \def\pfx@add{}%
322         \setkeys{Gin}{#1}%
323         \ifGin@pfx%
324             \pfxinput{#2}%
325         \fi%
326         \pfx@add
327         \pfx@overpic[#1]{#2}
328             \Gin@ovptrue
329             \Gin@overwriteovpfalse
330             \def\ovp@add@bgd{}%
331             \def\ovp@add@fgd{}%
332             \setkeys{Gin}{#1}%
333             \ovp@add@bgd
334             \ifGin@ovp
335                 \ovpinput{#2}%
336             \fi
337         }% \pfx@overpix
338 %
339 \def\endpfx@overpix{%
340             \ovp@add@fgd%
341         \pfx@endoverpic
342         \end{lrbox}%
343         \usebox{\ovp@box@tmp}%
344         \endgroup%
345         }%
```

9.4.2 External commands

The definitions here below ensure that the optional arguments are optional.

\overpix   The syntax of overpix is as follows.

\endoverpix   \overpix   [*keys*]
                             <*foreground layer*>
                             [*background layer*]
                             {*file.eps*}

```
346 \def\overpix{\@ifnextchar[{\@overpix}%
347                           {\@overpix[]}}%
348 \def\@overpix[#1]{\@ifnextchar<{\@@overpix[#1]}%
349                                {\@@overpix[#1]<>}}%
350 \def\@@overpix[#1]<#2>%
351         {\@ifnextchar[{\@@@overpix[#1]<#2>}%
352                       {\@@@overpix[#1]<#2>[]}}%
353 \def\@@@overpix[#1]<#2>[#3]#4%
```

```
354                    {\pfx@overpix{#1,ovpfgd={#2},ovpbgd={#3}}{#4}}
355 \def\endoverpix{\endpfx@overpix}
```

This set of commands converts the optional arguments into keys.

`\includegraphicx` The syntax of `\includegraphicx` is as follows.

> `\includegraphicx` [*keys*]
> (*psfrag replacements*)
> < *foreground layer* >
> [*background layer*]
> { *file.eps* }

```
356 \def\includegraphicx{\@ifnextchar[{\@igx}%
357                                 {\@igx[]}}%
358 \def\@igx[#1]{\@ifnextchar({\@@igx[#1]}%
359                             {\@@igx[#1]()}}%
360 \def\@@igx[#1](#2){\@ifnextchar<{\@@@igx[#1](#2)}%
361                               {\@@@igx[#1](#2)<>}}%
362 \def\@@@igx[#1](#2)<#3>{\@ifnextchar[{\@@@@igx[#1](#2)<#3>}%
363                                   {\@@@@igx[#1](#2)<#3>[]}}%
364 \def\@@@@igx[#1](#2)<#3>[#4]#5%
365     {\pfx@includegraphicx{#1,pfxadd={#2},ovpfgd={#3},ovpbgd={#4}}{#5}}
```

This set of commands converts the optional arguments into keys.

9.5   Overloading includegraphics and overpic

If the user requires so, we let `\includegraphics` and the `overpic` environment act as their counterparts ending in `x`. Though this substitution was tested, it could broke things and should be used with care.

`\includegraphics`
`\overpic`
`\endoverpic`
```
366 \ifpfx@sub
367         \let\includegraphics=\includegraphicx
368         \ifpfx@ovp
369                 \let\overpic=\overpix
370                 \let\endoverpic=\endoverpix
371         \fi
372 \fi
```

9.6   Configuration file

Finally, we input the configuration file if it exists.

```
373 \InputIfFileExists{psfragx.cfg}{}{}
```

This ends the code of `psfragx.sty`.

```
374 ⟨/package⟩
```

21

10 Code of the matlab script

Note that the lines containing the `rm` and `mv` commands should be replaced by their equivalents on the operating system on wich matlab is running. For example, under DOS and its successors the replacements are `del` and `ren`.

```
375 ⟨∗matlab⟩
376 %% psfragx.m [2012/05/02 v0.4 Pascal Kockaert]
377 %%
378 %% function psfragx(TeXname,EPSname,Outname)
379 %% nargin=1 -> EPSname=TeXname
380 %% nargin=2 -> Outname=EPSname
381 %%
382 %% Interleaves lines of TeXname.tex and EPSname.eps
383 %% so that all the lines of EPSname.eps are copied in Outname.tex and
384 %% lines from TeXname.tex starting with
385 %% \psfrag
386 %% and
387 %% %<pfx>
388 %% are written in Outname.eps, as a comment following the
389 %% %%BoundigBox
390 %% line.
391
392 %% This work may be distributed and/or modified under the
393 %% conditions of the LaTeX Project Public License, either version 1.3
394 %% of this license or (at your option) any later version.
395 %% The latest version of this license is in
396 %%   http://www.latex-project.org/lppl.txt
397 %% and version 1.3 or later is part of all distributions of LaTeX
398 %% version 2005/12/01 or later.
399 %%
400 %% This file is currently maintained by Pascal Kockaert
401
402 function psfragx(TeXname,EPSname,Outname)
403 TMPname='psfragx_tmp';
404 if nargin<2, EPSname=TeXname; end
405 if nargin<3, Outname=EPSname; end
406 if Outname==EPSname,
407    eval(['!rm ',TMPname,'.eps'])
408    eval(['!mv ',EPSname,'.eps ',TMPname,'.eps'])
409    EPSname=TMPname;
410 end
411 TeXName=([TeXname,'.tex']);
412 EPSName=([EPSname,'.eps']);
413 OutName=([Outname,'.eps']);
414
415 %% Uncomment for debugging purposes
416 %% eval(['ls ',TeXName])
```

```
417 %% eval(['ls ',EPSName])
418 %% eval(['ls ',OutName])
419
420
421 BeginInput ='%%BoundingBox:';
422 BeginPSFRAG='%<pfx>\pfxbegin[1.0]{laprint}%';
423 EndPSFRAG   ='%<pfx>\pfxend';
424 StartPFX    ='%<*pfx> Inserted where \begin{psfrags}% occured';
425 StopPFX     ='%</pfx> Inserted where \end{psfrags}% occured';
426 EndInput   ='%\endinput';
427 EndOfFile ='%%EOF';
428 ResizeBox ='%<pfx>\def\naturalwidth';
429 StopOn    ={'\psfrag{','<pfx>','\begin{psfrags}','\end{psfrags}','\resizebox'};
430
431 TeXFile=fopen(TeXName,'r');
432 if (TeXFile==-1)
433         error(['I was not able to open ',TeXName,'!']);
434 end
435 EPSFile=fopen(EPSName,'r');
436 if (EPSFile==-1)
437         error(['I was not able to open ',EPSName,'!']);
438 end
439 OutFile=fopen(OutName,'w');
440 if (OutFile==-1)
441         error(['I was not able to open ',OutName,'!']);
442 end
443
444 [sEPS,llEPS,iEPS]=CopyUntil(EPSFile,OutFile,{BeginInput});
445         if sEPS~=1, error(['No line contains ',BeginInput]);
446         else
447                 fprintf(OutFile,'%s\n',llEPS);
448         end
449
450 %%%
451 %%% Write preamble
452 %%%
453 fprintf(OutFile,'%%<*pfx> Begin Preamble\n');
454 fprintf(OutFile,'%%\\providecommand*{\\pfxbegin}[2][]{}%%\n');
455 fprintf(OutFile,'%%\\providecommand{\\pfxend}{}%%\n');
456 fprintf(OutFile,'%%</pfx> End Preamble\n');
457 %%%
458 %%% Copy interesting lines
459 %%%
460 while 1
461         [sTeX,llTeX,iTeX]=ReadUntil(TeXFile,StopOn);
462                 if sTeX~=1, break; end
463         switch iTeX
464             case 1,    %   \psfrag
465                     fprintf(OutFile,'%%%s\n',llTeX);
```

```
466         case 2,     %   %<pfx>
467               fprintf(OutFile,'%s\n',llTeX);
468         case 3,     %   \begin{psfrags}
469               fprintf(OutFile,'%s\n',BeginPSFRAG);
470               fprintf(OutFile,'%s\n',StartPFX);
471         case 4,     %   \end{psfrags}
472               fprintf(OutFile,'%s\n',StopPFX);
473               fprintf(OutFile,'%s\n',EndPSFRAG);
474         case 5,     %   \resizebox
475               tmpbeg=findstr(llTeX,'{');
476               tmpend=findstr(llTeX,'}');
477               if (length(tmpbeg)>0)&(length(tmpend)>0)
478                   if (tmpbeg(1)<tmpend(1))
479                       fprintf(OutFile,'%s%s%%\n',ResizeBox,llTeX(tmpbeg(1):tmpend(1
480                   end
481               end
482       otherwise
483               error('Otherwise should never happen !')
484       end
485 end
486 %%%
487 %%% Write postamble
488 %%%
489 fprintf(OutFile,'%s\n',EndInput);
490 %%%
491 %%% Copy to the end of file
492 %%%
493 [sEPS,llEPS,iEPS]=CopyUntil(EPSFile,OutFile,{''});
494 %%%
495 %%% Close files
496 %%%
497 fclose(OutFile);
498 fclose(TeXFile);
499 fclose(EPSFile);
500 return
501
502 function [OK,lastline,elt]=CopyUntil(fidIn,fidOut,linebeg);
503 sl=length(linebeg);
504 if sl==0, OK=-2; return, end
505 llb=zeros(sl);
506 for ii=1:sl
507       llb(ii)=length(linebeg{ii});
508 end
509 lastline='';
510 OK=0;
511 elt=0;
512 while 1
513       Line=fgetl(fidIn);
514       if ~isstr(Line),
```

```
515                    OK=-1;
516                    return,
517          end   %EndOfFile
518          for ii=1:sl
519 %%%                 fprintf('Seeking for line starting with %s.\n',linebeg{ii});
520                 if llb==0,    %%% Copying to the end of file
521                 else
522                        if length(Line)>=llb(ii)
523 %%%                           fprintf('This line counts more than %i chars.\n',llb(i
524                            if Line(1:llb(ii))==linebeg{ii},
525                                   OK=1;
526                                   elt=ii;
527                                   lastline=Line;
528                                   break
529                               end
530                        end
531                 end
532          end     %%% No matching string
533          if OK==1, break, end
534          if ~isempty(fidOut)
535                 fprintf(fidOut,'%s\n',Line);
536          end
537 end
538 return
539
540 function [OK,lastline,elt]=ReadUntil(fidIn,linebeg);
541          [OK,lastline,elt]=CopyUntil(fidIn,[],linebeg);
542 return
543 ⟨/matlab⟩
544 ⟨∗pfxprint⟩
545 %%% pfxprint [ -*- Matlab -*- ] Time-stamp: <2004-08-12 18:20:57 Pascal Kockaert>
546 %%%
547 % function pfxprint(fig,name,'optA','valA','optB','valB',...)
548 %
549 % TO USE THIS FUNCTION, THE FILE laprpfx.mat SHOULD BE IN THE MATLAB PATH
550 %
551 % This function is to be used like laprint.m
552 % The EPS and TeX files resulting from the call to laprint with the given arguments
553 % are automatically merged into one EPS file that contains the
554 % psfrags replacements as comments.
555 % These comments can be automatically used in LaTeX, with the help of the psfragx pac
556 %
557 % This file is subject to the LPPL licence (see other files in the source archive or
558 % Copyright 2004, Pascal Kockaert
559 %
560
561 function pfxprint(fig,name,varargin)
562  deftxtint=get(0,'DefaultTextInterpreter');
563  set(0,'DefaultTextInterpreter','none');
```

```
564
565   laprint(fig,name,'options','laprpfx',varargin{:});
566   psfragx(name);
567
568  set(0,'DefaultTextInterpreter',deftxtint)
569
570 % Default options are
571 % LAPRINTOPT =
572 %             figno: 2
573 %          filename: 'laprint'
574 %             width: 12
575 %            factor: 0.8
576 %         scalefonts: 1
577 %      keepfontprops: 0
578 %          asonscreen: 0
579 %     keepticklabels: 0
580 %     mathticklabels: 0
581 %               head: 0
582 %            comment: 'Test de laprint'
583 %            caption: ''
584 %       extrapicture: 0
585 %             nzeros: 3
586 %            verbose: 'off'
587 %            figcopy: 1
588 %           printcmd: 'print('-f<figurenumber>','-depsc2','<filename.eps>')'
589 %            package: 'graphicx'
590 %              color: 0
591 %         createview: 0
592 %       viewfilename: 'unnamed_'
593 %        processview: 0
594 %               cmd1: 'latex -halt-on-error -interaction nonstopmode <viewfile>.tex'
595 %               cmd2: 'dvips -D600 -E* -o<viewfile>.eps <viewfile>.dvi'
596 %               cmd3: 'epstool --bbox --copy --output <filename>_final.eps <viewfile>
597 %               cmd4: 'rm <viewfile>.eps <viewfile>.dvi <viewfile>.aux <viewfile>.log
598 %               cmd5: 'ghostview <filename>_final.eps&'
599 %               cmd6: ''
600 %               cmd7: ''
601 %               cmd8: ''
602 ⟨/pfxprint⟩
```

Change History

v0.1

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.