# PSfragX: one graphic in one file [*]

Pascal Kockaert

2004/12/10

## 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 v1.0. It was processed on 2004/12/10.

# 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.

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

# 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 simplify 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}
```

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 |

# 4 Usage

## 4.1 Package options

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

The *options* are

**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 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.

```
⟨*cfg⟩
% Example of configuration file for psfragx.sty
% The macros \Beforepfxinput, \Afterpfxinput
% \Beforeovpinput, and \Afterovpinput are executed
% into a group. They should not define global commands to
% avoid side effects.
%
%
% The command \providecolorcommands defines commands that
% take the same arguments as the mains commands of the
% color package, in case this package is not loaded.
%
\newcommand{\providecolorcommands}
           {\def\pfx@gobble@two##1##2{\typeout{Some psfragx
                                       replacement would appear
                                       in color ##1{##2}
                                       if the color package was
                                       loaded!!!}}%
            \def\pfx@gobble@three@fbox##1##2##3{\typeout{Some psfragx
                                                replacement would
                                                appear
                                                in color ##1{##2}
                                                and others
                                                in color ##1{##3}
                                                if the color package
                                                was loaded!!!}%
                                                \fbox}%
            \def\pfx@fm@to@mm##1##2##{\csname ##1\endcsname{##2}}%
            \expandafter\ifx\csname textcolor\endcsname\relax
              \def\textcolor{\pfx@fm@to@mm{pfx@gobble@two}}\fi
            \expandafter\ifx\csname color\endcsname\relax
              \def\color{\pfx@fm@to@mm{pfx@gobble@two}}\fi
            \expandafter\ifx\csname colorbox\endcsname\relax
              \def\colorbox{\pfx@fm@to@mm{pfx@gobble@two}}\fi
            \expandafter\ifx\csname fcolorbox\endcsname\relax
              \def\fcolorbox{\pfx@fm@to@mm{pfx@gobble@three@fbox}}\fi
            }
% The name of the next four commands are specific to psfragx
\def\Beforepfxinput{\providecolorcommands}
\def\Afterpfxinput{}
\def\Beforeovpinput{\providecolorcommands}
\def\Afterovpinput{}
⟨/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.

```
⟨*example⟩
 %%!PS-Adobe-2.0 EPSF-1.2
 %%Creator: Adobe Illustrator(TM) 1.2d4
 %%Title: tiger.eps
 %%CreationDate: 4/12/90 3:20 AM
```

```
%%BoundingBox: 17 171 567 739
%<*pfx>
%\psfrag{T}[B][B]{\fcolorbox{white}{black}{\color{white}Title}}
%\psfrag{t}[t][t]{time (s)}
%\psfrag{I}[b][b]{I (W)}
%\iflanguage{french}
%          {\psfrag{T}[B][B]{Title}%
%           \psfrag{t}[t][t]{temps (s)}}
%          {}
%\psfrag{T}[B][B]{Title}
%</pfx>
%<*ovp>
%\put(0,80){(a)}
%</ovp>
%\endinput
%%EndComments
%%
%% [The code of the {\EPS} file should come HERE]
%%
%% End
⟨/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

⟨*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

```
\DeclareOption{sub}{\pfx@subtrue}
\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.

```
\DeclareOption{allcom}{\allmetacomments}
\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

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

We define the new commands needed to process the options.

\allmetacomments
\selectedmetacomments

```
        \newif\ifpfx@sub\pfx@subfalse
        \newif\ifpfx@ovp\pfx@ovptrue
        \newif\ifpfx@metacomments
        \pfx@metacommentsfalse
        \def\allmetacomments{\pfx@metacommentstrue}
        \def\selectedmetacomments{\pfx@metacommentsfalse}
```

Finally, default options are defined.

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

Now, we load the other packages.

```
\RequirePackage{graphicx}
\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`.

```
\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`

```
\def\pfx@gobble#1{}
{\catcode`\%=12
 \gdef\pfx@percent{%}
 \gdef\pfx@doublepercent{%%}
}
```

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`

```
\let\pfx@metaprefix\pfx@doublepercent
\def\pfx@tag{pfx}
\def\pfx@ext{pfx}
\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`

```
\edef\pfx@endinput
     {\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,

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

another to copy a given token,

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

and a few boolean variables.

```
\newif\ifpfx@continue
\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

```
\def\setpfxinput#1{\gdef\pfx@infile{#1}}
\def\setpfxoutput#1{\gdef\pfx@outfile{#1}}
\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.

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

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

\copypfxfromto

```
\def\copypfxfromto#1#2{%
\setpfxinput{#1}%
\setpfxoutput{#2}%
\copypfxlines%
}
```

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

\copypfxlines

```
\def\pfx@ignorespaces{\ignorespaces}%
\def\copypfxlines{% input and output files are global names
\immediate\openin\pfx@in\pfx@infile\relax \ifeof\pfx@in
      \errmessage{psfragx tried to read from a file that
                  does not exist. This seems to be a bug!}%
\else
      \immediate\openout\pfx@out=\pfx@outfile\relax
      \immediate\write\pfx@out{\pfx@ignorespaces}
      \ifeof\pfx@out
            \begingroup
            \pfx@makeother\ \pfx@makeother\\\pfx@makeother\$%
            \pfx@makeother\#\pfx@makeother\^\pfx@makeother\^^K%
            \pfx@makeother\_\pfx@makeother\^^A\pfx@makeother\%%
            \pfx@makeother\~\pfx@makeother\{\pfx@makeother\}%
            \pfx@makeother\&\endlinechar-1\relax
            \loop
```

```
                    \read\pfx@in to \pfx@inline
                    \ifx\pfx@inline\pfx@endinput
                           \pfx@continuefalse
                           \typeout{psfragx: \pfx@percent
                                   \expandafter\pfx@gobble
                                   \string\\endinput was
                                   found in \pfx@infile.}%
                    \else
                           \ifeof\pfx@in
                                  \pfx@continuefalse
                           \typeout{psfragx: End of file
                                   \pfx@infile was reached.}%
                           \else
                                  \pfx@continuetrue
                                  \expandafter\pfx@processline
                                  \pfx@inline\pfx@endline
                           \fi%
                    \fi%
            \ifpfx@continue
            \repeat
            \endgroup
      \else
             \errmessage{psfragx: output file already exists!}%
      \fi %\pfx@out
      \immediate\closeout\pfx@out
\fi %\pfx@in
\immediate\closein\pfx@in
}
```

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

```
\def\pfx@normalline#1\pfx@endline{%
      \def\pfx@inline{#1}%
      \ifpfx@outputtofile%
             \immediate\write\pfx@out{\pfx@inline}%
      \fi%
}
%
\def\pfx@removecomment#1\pfx@endline{%
      \def\pfx@inline{#1}%
      \ifpfx@outputtofile%
             \immediate\write\pfx@out{\pfx@inline}%
```

```
        \fi%
}
%
\bgroup\catcode‘\%=12 \pfx@iden{\egroup
\def\pfx@putmetacomment%}#1\pfx@endline{%
        \edef\pfx@inline{\pfx@metaprefix#1}%
        \ifpfx@metacomments
                \immediate\write\pfx@out{\pfx@inline}%
        \else
                \ifpfx@outputtofile
                        \immediate\write\pfx@out{\pfx@inline}%
                \fi
        \fi
}
%
\begingroup
\catcode‘\%=12 \catcode‘\*=14 \gdef\pfx@processline#1{*
        \ifx%#1*
                \expandafter\pfx@processlinex
        \else
                \expandafter\pfx@normalline
        \fi
#1}
\endgroup
%
\begingroup
\catcode‘\%=12 \catcode‘\*=14
\gdef\pfx@processlinex%#1{*
        \ifcase\ifx%#10\else
                \ifx<#11\else2\fi\fi\relax
        \expandafter\pfx@putmetacomment\or
        \expandafter\pfx@checkoption\or
        \expandafter\pfx@removecomment\fi
#1}
\endgroup
%
\def\pfx@checkoption<#1{%
        \ifcase\ifx*#10\else
                \ifx/#11\else2\fi\fi\relax
        \expandafter\pfx@staroption\or
        \expandafter\pfx@slashoption\or
        \expandafter\pfx@tagoption\fi
#1}
%
\def\pfx@staroption*#1>#2\pfx@endline{%
        \def\pfx@tmp{#1}%
        \ifx\pfx@tmp\pfx@tag
                \pfx@outputtofiletrue
        \fi
```

```
}
%
\def\pfx@slashoption/#1>#2\pfx@endline{%
        \def\pfx@tmp{#1}%
        \ifx\pfx@tmp\pfx@tag\relax
                \pfx@outputtofilefalse
        \fi
}
%
\def\pfx@tagoption#1>#2\pfx@endline{%
        \def\pfx@tmp{#1}%
        \ifx\pfx@tmp\pfx@tag\relax
                \def\pfx@inline{#2}%
                \immediate\write\pfx@out{\pfx@inline}%
        \fi
}
```

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 pfxto 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

```
\let\pfx@includegraphics=\includegraphics
\let\pfx@overpic=\overpic
\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

```
\@ifundefined{pfx@overpic}{%
                        \newcommand*{\pfx@overpic}[2][]
```

```
                         {\sbox{\z@}{\includegraphics[#1]{#2}}%
                         \settodepth{\@tempcnta}{\usebox{\z@}}%
                         \settoheight{\@tempcntb}{\usebox{\z@}}%
                         \advance\@tempcntb\@tempcnta%
                         \settowidth{\@tempcnta}{\usebox{\z@}}%
                         \begin{picture}(\@tempcnta,\@tempcntb)%
                         \put(0,0){\makebox(0,0)[bl]{\usebox{\z@}}}}%
                        }{}
\@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
\endonlylanguage

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*.

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

```
\long\def\pfx@firstoftwo#1#2{#1\ignorespaces}%
\long\def\pfx@secondoftwo#1#2{#2\ignorespaces}%
\def\pfx@iflanguage#1{%
    \ifnum\csname l@#1\endcsname=\language
      \expandafter\pfx@firstoftwo
    \else
      \expandafter\pfx@secondoftwo
    \fi}
\long\def\onlylanguage#1#2\endonlylanguage{\pfx@iflanguage{#1}{#2}{}\ignorespaces}
\def\pfx@save@iflanguage{\let\save@pfx@iflanguage=\iflanguage%
                           \let\iflanguage=\pfx@iflanguage}
\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.

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

```
\newif\ifpfx@generate
\newif\ifovp@generate
\newcommand*\pfx@add{}
\newcommand*\ovp@add@bgd{}
\newcommand*\ovp@add@fgd{}
\newif\ifGin@pfx
\newif\ifGin@overwritepfx
\newif\ifGin@ovp
\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.

```
\define@key{Gin}{pfx}[true]%
           {\lowercase{\Gin@boolkey{#1}}{pfx}}
\define@key{Gin}{overwritepfx}[true]%
           {\lowercase{\Gin@boolkey{#1}}{overwritepfx}}
\define@key{Gin}{pfxadd}[]%
           {\def\pfx@add{#1}}
\define@key{Gin}{ovp}[true]%
           {\lowercase{\Gin@boolkey{#1}}{ovp}}
\define@key{Gin}{overwriteovp}[true]%
           {\lowercase{\Gin@boolkey{#1}}{overwriteovp}}
\define@key{Gin}{ovpbgd}[]%
           {\def\ovp@add@bgd{#1}}
\define@key{Gin}{ovpfgd}[]%
           {\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

```
\newcommand*{\pfxinput}[1]{%
  \filename@parse{#1}%
  \IfFileExists{\filename@base.\pfx@ext}
               {\pfx@generatefalse}
               {\pfx@generatetrue}%
  \ifGin@overwritepfx\pfx@generatetrue\fi
  \IfFileExists{#1}{}{\pfx@generatefalse}%
  \ifpfx@generate%
       \copypfxfromto{\filename@area\filename@base.\filename@ext}
                     {\filename@base.\pfx@ext}%
  \fi%
  \pfx@save@iflanguage
  \csname Before\pfx@tag input\endcsname
  \InputIfFileExists{\filename@base.\pfx@ext}
                    {\typeout{psfragx: reading commands from
                                       \filename@base.\pfx@ext}}
                    {\typeout{psfragx: I was not able to read psfrag
                              definitions from
                              \filename@base.\pfx@ext}}%
  \csname After\pfx@tag input\endcsname
  \pfx@restore@iflanguage
}
```

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

```
\newcommand*{\ovpinput}[1]{%
\begingroup
      \def\pfx@ext{ovp}%
      \def\pfx@tag{ovp}%
      \ifGin@overwriteovp\Gin@overwritepfxtrue
                        \else\Gin@overwritepfxfalse\fi
      \pfxinput{#1}%
\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.

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

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

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

\pfx@overpix
\endpfx@overpix The environment overpix has the same syntax as the overpic one. You 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.

```
\def\pfx@overpix#1#2{%
     \begingroup%
     \begin{lrbox}{\ovp@box@tmp}%
     \let\includegraphics=\pfx@includegraphics%
     \Gin@pfxtrue%
     \Gin@overwritepfxfalse%
     \def\pfx@add{}%
     \setkeys{Gin}{#1}%
     \ifGin@pfx%
        \pfxinput{#2}%
     \fi%
     \pfx@add
     \pfx@overpic[#1]{#2}
             \Gin@ovptrue
             \Gin@overwriteovpfalse
             \def\ovp@add@bgd{}%
             \def\ovp@add@fgd{}%
             \setkeys{Gin}{#1}%
             \ovp@add@bgd
             \ifGin@ovp
                     \ovpinput{#2}%
             \fi
     }% \pfx@overpix
%
\def\endpfx@overpix{%
             \ovp@add@fgd%
     \pfx@endoverpic
     \end{lrbox}%
     \usebox{\ovp@box@tmp}%
     \endgroup%
     }%
```

### 9.4.2 External commands

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

\overpix
\endoverpix

The syntax of overpix is as follows.

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

```
\def\overpix{\@ifnextchar[{\@overpix}%
                          {\@overpix[]}}%
\def\@overpix[#1]{\@ifnextchar<{\@@overpix[#1]}%
                               {\@@overpix[#1]<>}}%
\def\@@overpix[#1]<#2>%
              {\@ifnextchar[{\@@@overpix[#1]<#2>}%
                            {\@@@overpix[#1]<#2>[]}}%
\def\@@@overpix[#1]<#2>[#3]#4%
               {\pfx@overpix{#1,ovpfgd={#2},ovpbgd={#3}}{#4}}
\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*}

```
\def\includegraphicx{\@ifnextchar[{\@igx}%
                                  {\@igx[]}}%
\def\@igx[#1]{\@ifnextchar({\@@igx[#1]}%
                           {\@@igx[#1]()}}%
\def\@@igx[#1](#2){\@ifnextchar<{\@@@igx[#1](#2)}%
                                {\@@@igx[#1](#2)<>}}%
\def\@@@igx[#1](#2)<#3>{\@ifnextchar[{\@@@@igx[#1](#2)<#3>}%
                                     {\@@@@igx[#1](#2)<#3>[]}}%
\def\@@@@igx[#1](#2)<#3>[#4]#5%
    {\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

```
\ifpfx@sub
       \let\includegraphics=\includegraphicx
       \ifpfx@ovp
              \let\overpic=\overpix
              \let\endoverpic=\endoverpix
       \fi
```

```
\fi
```

## 9.6 Configuration file

Finally, we input the configuration file if it exists.

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

This ends the code of `psfragx.sty`.

```
⟨/package⟩
```

# 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`.

```
⟨*matlab⟩
% psfragx.m                      %%% [-*- Matlab -*-]
%
% function psfragx(NomTeX,NomEPS)
% nargin=1 -> NomTeX=NomEPS
%
% Copy lines of NomTeX.tex
% starting with
% \psfrag
% and
% %<pfx>
% to the file NomEPS.eps, as a comment following the
% %%BoundigBox
% line.

function psfragx(TeXname,EPSname,Outname)
TMPname='psfragx_tmp';
if nargin<2, EPSname=TeXname; end
if nargin<3, Outname=EPSname; end
if Outname==EPSname,
   eval(['!rm ',TMPname,'.eps'])
   eval(['!mv ',EPSname,'.eps ',TMPname,'.eps'])
   EPSname=TMPname;
end
TeXName=([TeXname,'.tex']);
EPSName=([EPSname,'.eps']);
OutName=([Outname,'.eps']);

BeginInput ='%%BoundingBox:';
BeginPSFRAG='%<pfx>\pfxbegin[1.0]{laprint}%';
EndPSFRAG  ='%<pfx>\pfxend';
```

```
StartPFX   ='%<*pfx> Inserted where \begin{psfrags}% occured';
StopPFX    ='%</pfx> Inserted where \end{psfrags}% occured';
EndInput  ='%\endinput';
EndOfFile ='%%EOF';
ResizeBox ='%<pfx>\def\naturalwidth';
StopOn    ={'\psfrag{','<pfx>','\begin{psfrags}','\end{psfrags}','\resizebox'};

TeXFile=fopen(TeXName,'r');
if (TeXFile==-1)
        error(['I was not able to open ',TeXName,'!']);
end
EPSFile=fopen(EPSName,'r');
if (EPSFile==-1)
        error(['I was not able to open ',EPSName,'!']);
end
OutFile=fopen(OutName,'w');
if (OutFile==-1)
        error(['I was not able to open ',OutName,'!']);
end

[sEPS,llEPS,iEPS]=CopyUntil(EPSFile,OutFile,{BeginInput});
       if sEPS~=1, error(['No line contains ',BeginInput]);
       else
                 fprintf(OutFile,'%s\n',llEPS);
       end

%%%
%%% Write preamble
%%%
fprintf(OutFile,'%%<*pfx> Begin Preamble\n');
fprintf(OutFile,'%%\\providecommand*{\\pfxbegin}[2][]{}%%\n');
fprintf(OutFile,'%%\\providecommand{\\pfxend}{}%%\n');
fprintf(OutFile,'%%</pfx> End Preamble\n');
%%%
%%% Copy interesting lines
%%%
while 1
       [sTeX,llTeX,iTeX]=ReadUntil(TeXFile,StopOn);
              if sTeX~=1, break; end
       switch iTeX
          case 1,    %   \psfrag
                 fprintf(OutFile,'%%%s\n',llTeX);
          case 2,    %   %<pfx>
                 fprintf(OutFile,'%s\n',llTeX);
          case 3,    %   \begin{psfrags}
                 fprintf(OutFile,'%s\n',BeginPSFRAG);
                 fprintf(OutFile,'%s\n',StartPFX);
          case 4,    %   \end{psfrags}
                 fprintf(OutFile,'%s\n',StopPFX);
                 fprintf(OutFile,'%s\n',EndPSFRAG);
          case 5,    %   \resizebox
                 tmpbeg=findstr(llTeX,'{');
                 tmpend=findstr(llTeX,'}');
                 if (length(tmpbeg)>0)&(length(tmpend)>0)
                     if (tmpbeg(1)<tmpend(1))
                        fprintf(OutFile,'%s%s%%\n',ResizeBox,llTeX(tmpbeg(1):tmpend(1)));
                     end
                 end
       otherwise
              error('Otherwise should never happen !')
       end
end
%%%
%%% Write postamble
%%%
fprintf(OutFile,'%s\n',EndInput);
%%%
%%% Copy to the end of file
%%%
[sEPS,llEPS,iEPS]=CopyUntil(EPSFile,OutFile,{''});
%%%
%%% Close files
%%%
fclose(OutFile);
fclose(TeXFile);
fclose(EPSFile);
return

function [OK,lastline,elt]=CopyUntil(fidIn,fidOut,linebeg);
sl=length(linebeg);
if sl==0, OK=-2; return, end
llb=zeros(sl);
for ii=1:sl
       llb(ii)=length(linebeg{ii});
end
lastline='';
OK=0;
elt=0;
while 1
       Line=fgetl(fidIn);
       if ~isstr(Line),
              OK=-1;
              return,
       end  %EndOfFile
       for ii=1:sl
%%%              fprintf('Seeking for line starting with %s.\n',linebeg{ii});
              if llb==0,    %%% Copying to the end of file
              else
```

```
                     if length(Line)>=llb(ii)
%%%                            fprintf(’This line counts more than %i chars.\n’,llb(ii));
                            if Line(1:llb(ii))==linebeg{ii},
                                   OK=1;
                                   elt=ii;
                                   lastline=Line;
                                   break
                            end
                     end
              end
       end    %%% No matching string
       if OK==1, break, end
       if ~isempty(fidOut)
              fprintf(fidOut,’%s\n’,Line);
       end
end
return

function [OK,lastline,elt]=ReadUntil(fidIn,linebeg);
         [OK,lastline,elt]=CopyUntil(fidIn,[],linebeg);
return
⟨/matlab⟩
⟨*pfxprint⟩
%%% pfxprint [ -*- Matlab -*- ] Time-stamp: <2004-08-12 18:20:57 Pascal Kockaert>
%%%
% function pfxprint(fig,name,’optA’,’valA’,’optB’,’valB’,...)
%
% TO USE THIS FUNCTION, THE FILE laprpfx.mat SHOULD BE IN THE MATLAB PATH
%
% This function is to be used like laprint.m
% The EPS and TeX files resulting from the call to laprint with the given arguments
% are automatically merged into one EPS file that contains the
% psfrags replacements as comments.
% These comments can be automatically used in LaTeX, with the help of the psfragx package
%
% This file is subject to the LPPL licence (see other files in the source archive or www.
% Copyright 2004, Pascal Kockaert
%

function pfxprint(fig,name,varargin)
 deftxtint=get(0,’DefaultTextInterpreter’);
 set(0,’DefaultTextInterpreter’,’none’);

  laprint(fig,name,’options’,’laprpfx’,varargin{:});
  psfragx(name);

 set(0,’DefaultTextInterpreter’,deftxtint)

% Default options are
% LAPRINTOPT =
%              figno: 2
%           filename: 'laprint'
%              width: 12
%             factor: 0.8
%         scalefonts: 1
%      keepfontprops: 0
%         asonscreen: 0
%     keepticklabels: 0
%     mathticklabels: 0
%               head: 0
%            comment: 'Test de laprint'
%            caption: ''
%       extrapicture: 0
%             nzeros: 3
%            verbose: 'off'
%            figcopy: 1
%           printcmd: 'print('-f<figurenumber>','-depsc2','<filename.eps>')'
%            package: 'graphicx'
%              color: 0
%         createview: 0
%       viewfilename: 'unnamed_'
%        processview: 0
%               cmd1: 'latex -halt-on-error -interaction nonstopmode <viewfile>.tex'
%               cmd2: 'dvips -D600 -E* -o<viewfile>.eps <viewfile>.dvi'
%               cmd3: 'epstool --bbox --copy --output <filename>_final.eps <viewfile>.eps
%               cmd4: 'rm <viewfile>.eps <viewfile>.dvi <viewfile>.aux <viewfile>.log <vi
%               cmd5: 'ghostview <filename>_final.eps&'
%               cmd6: ''
%               cmd7: ''
%               cmd8: ''
⟨/pfxprint⟩
```

# 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.