The export package

Jean-Pierre F. Drucbert
drucbert@onecert.fr

2000/10/16

Abstract

This package[1] allows you to export or import the values of various LaTeX registers like counters and lengths (rigid or rubber). This could be useful to fabricate "composite" documents or to pass some data from a document to another one.

1 The export package

This package is designed to transmit data of type ⟨*counter*⟩ or ⟨*length*⟩ from a LaTeX document to another. You must be aware that this package must be used *with extreme care*. The passing of data is done via a file. A first document *exports* the data by invoking the following sequence of commands:

\openexport
\closeexport
\Export
\ExportLength
\PreciseExportLength
\ExportMuskip
\ExportParameter
\ExportIf

```
\openexport[⟨extension⟩]{⟨file⟩}
\Export{⟨counter⟩}
\Export{⟨counter⟩}
.
.
.
\ExportLength{⟨length⟩}
\PreciseExportLength{⟨length⟩}
\ExportMuskip{⟨muskip⟩}
\ExportParameter{⟨parameter⟩}
\ExportIf{⟨ifname⟩}
.
.
.
\closeexport
```

where ⟨*file*⟩ is the file used to carry the data (its name will have the `.xpt` extension by default, but you must not give it here; an optionnal argument may be used to specify the extension), ⟨*counter*⟩ is the name of a LaTeX counter (i.e. one of the well known counters `chapter`, `section`, ..., `equation`, etc.), or a counter you have declared by `\newcounter`. Also, ⟨*length*⟩ is the name (*without the backslash*) of a LaTeX length like `parskip` or of a length declared by `\newlength`. In the "importing" document, we will find a command `\Import{`⟨*file*⟩`}` or `\Import[`⟨*extension*⟩`]{`⟨*file*⟩`}`. The `\ExportLength` command exports a length expressed in *points* (`pt`) which some fractionnal digits (5 is the maximum), while `\PreciseExportLength` exports the length expressed in *scaled points* units (`sp`, 65536 `sp` make a point), so it avoids possible rounding errors. David Carlisle has pointed out that using that macro with a *rubber* length as argument would lose the shrinking and stretching parts and that `\ExportLength` does not make any rounding error in fact. You can trust him! Always use `\ExportLength` in case of doubt.

`\ExportMuskip` is like `\ExportLength` but for a muskip register. In fact, you could use `\ExportLength` for a muskip register: the two commands are identical, except the message written in the `.log` file. But you can not use `\PreciseExportLength` with a muskip register: another pretext to not use that superfluous command.

`\ExportParameter` allows you to export parameters defined by simple commands, like `\textfraction`. Do not try to export complex things. It would nor work, at best, or crash, at worst.

`\ExportIf` allows you to export the state of an "if" condition, i.e. the command `\ExportIf{foo}` will export the state of `\iffoo`, just by writing `\foofalse` or `\footrue` in the export file. *Note that the* ⟨*ifname*⟩ *does not contains the letters if at the beginning, nor the backslash.*

`\ExportPageLayout` If you want to export *all* the dimensions (lengths) describing the page layout, just use the `\ExportPageLayout` command. In a standard way, you should export only that into a given file, which should be imported in the preamble of the importing document. This method allows to have two documents with similar page layouts.

`\ExportArrayParams` This macro export the dimensions and parameters related to the layout of `array` and `tabular` environments.

1.1 An Example of Application

This package is very useful when associated which the dvipaste package [2], which allows you to prepare rectangular page pieces and to "paste" these page pieces into your document. **Warning:** if a such pasted piece modifies an imported counter or length, you must take account of that alteration in the document receiving the pasted piece.

The dvipaste.sty package is now available from CTAN archives, in the same directory than the export package. But you should get from CTAN the whole

[2] This package is now obsoleted by the nice package grfpaste from David Carlisle, but the same remarks still apply.

directory containing dvipaste.tex is. Get it (and the accompagning files), and read the documentation of dvipaste: you will also need the dvipaste.c program. The grfpaste package is also available from CTAN archives; it needs the dvipaste.c program. A simple example is described in the next pages.

1. the main document `a.tex` exports the counters `equation` and `chapter`, via the file `x.xpt`, toward the document `b.tex`. Adding to that, `a.tex` imports the labels defined in `b.tex` (using the xr package, from David Carlisle).

a.tex	b.tex
`\documentclass{article}%`	`\documentclass{article}%`
`\usepackage%`	`\usepackage%`
`  {...,export,dvipaste,xr,...}`	`  {...,export,dvipaste,...}`
`\externaldocument{b}`	
`\begin{document}`	`\begin{document}`
⋮ ⋮	⋮ ⋮
⋮ ⋮	⋮ ⋮
`\openexport{x}`	
`\Export{equation}`	
`\Export{chapter}`	
⋮ ⋮	⋮ ⋮
⋮ ⋮	⋮ ⋮
`\closeexport`	

2. The document `b.tex` imports the data contained in the file `x.xpt`, i.e. the values of the `equation` and `chapter` counters.

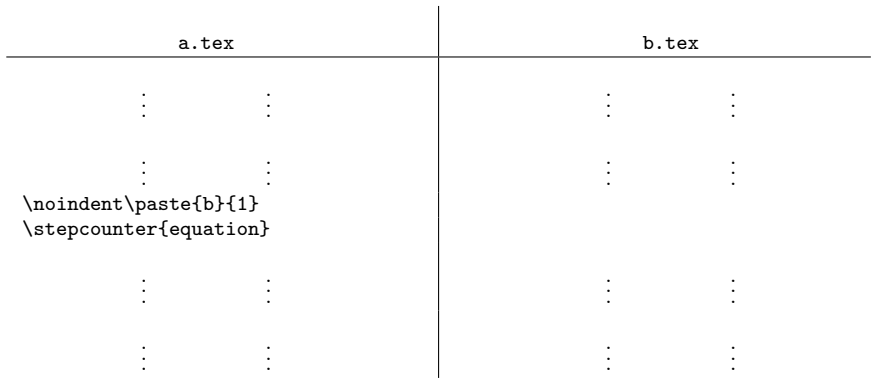
a.tex	b.tex
	`\Import{x}`

3. the it uses this counter in preparing an equation in a logical box:

a.tex	b.tex
	`\setbox0=\hbox{%`
	`  \begin{minipage}{\textwidth}`
	`\begin{equation}`
	`a=b  \label{eq-a}`
	`\end{equation}`
	`\end{minipage}}`
	`\sendout{\box0}`
⋮ ⋮	⋮ ⋮
⋮ ⋮	⋮ ⋮
	`\end{document}`

4. The document `a.tex` "pastes" the piece created by `b.tex`, then explicitely increments the `equation` counter, since the pasted piece contains an unaccounted equation [3]:

[3] An other, more general, method would be to "export back" the counter via an other file. This method is used when several counters are involved.



In fact, the following sequence of actions must be performed here:

latex b	to create the files `b.dat`, which contain the dimensions of the pieces to "paste" and `b.dvi` contains the pieces themselves. But `b.tex` will not be able to import anything, because `x.xpt` has not yet been created;
latex a	to create `x.xpt`;
latex b	to create again `b.dat` with the right dimensions and `b.dvi` containing correct values for equation numbers, since importation could be done;
latex a	to take into account the new dimensions in `b.dat`;
dvipaste a	to "paste" into `a.dvi` the pieces coming from `b.dvi`.

Note that the `xr` package, from David Carlisle, allows you to pass ⟨*label*⟩ information (i.e. to use `\ref{`⟨*label*⟩`}` or `\pageref{`⟨*label*⟩`}`) when the ⟨*label*⟩ is defined by `\label{`⟨*label*⟩`}` in an other LaTeX document.

1.2 Figures and Tables

`\xcaptionf` `\xcaptiont`

In the case of a figure or table which is "pasted" this way into a document *with* its caption, this one will not be present in the list of figures or tables. An easy solution is to use a `\xcaptionf{`⟨*caption*⟩`}` (or `\xcaptiont{`⟨*caption*⟩`}` for a table) command. These commands will increment the figure or table counter.

1.3 Inserting several pages

Sometimes, you will need to separately typeset several pages of a document. To accomplish that, you prepare a second document containing only the desired pages. This second document will need, by exemple, some packages not used in the main document (often because they require too much resources). To get correctly numbered pages, figures, tables, notes, etc., in the main document and in the secondary document, you just have to export the corresponding counters from the main document *before* the insertion, to import them at the beginning of the secondary document, to export them again (via an other file) at the end of the secondary

document, and at least to import them (via this second file) *after* the insertion into the main document.

> It will certainly be necessary to transmit the page counter, which is named "page". You must then add a `\clearpage` command *before* the exportation. This is mandatory only if the exported value must be used to number the page which follows the exportation. Do not export the page counter to an auxiliary document in which you are creating pieces to be "pasted" into your document (in such an auxiliary document, the "page counter" is in fact counter used to number the pieces to be pasted).

The insertion is only logical: you will have to do it manually in the printed paper sheets, or to use software tools manipulating `.dvi` files (e.g. **dvidvi**, **dviselect**, and **dviconcat**). A serious problem subsists: if they are sectionning commands and floating elements (like figures and tables) with captions in the secondary document, what to do to have a correct table of contents, a correct list of figures and tables in the main document? A similar problem arises for the bibliography. You just need to add, in the main document, at the insertion point, a command:

`\AddInputInAux`

```
\AddInputInAux{⟨secondary document⟩.aux}
```

and so the `.aux` file of the secondary document will be considered as an auxiliary file (of first or second level) of the main document. This method has a small drawback: if you declare the main document as an external document (as defined by the xr package) of the secondary document in order to be able to reference labels of the main document (or of parts of it) from the secondary one, you will have to use the optionnal argument in the command:

```
\usepackage{xr}
\externaldocument[⟨optiannal arg.⟩]{⟨main document⟩}
\externaldocument[⟨optionnal arg.⟩]{⟨part of main document⟩}
```

else the labels in the main document (or parts of it) would be considered as defined twice during the processing of the secondary document. This happens not often, but if you have this problem, the solution is simple but not trivial.

2 Acknowledgments

I must thanks, once more, David Carlisle for his precious comments, for the grfpaste package, and his great work on the xr package.

3 Problems

- This package is certainly to be used with care; the user must understand its mechanism.

- I did not found a way to determine automatically the type of a LaTeX register. If it is not defined, you get an error. But exporting a register with the wrong type (i.e. a rubber length as a counter) may lead to confusion or loss of data.
- The documentation is written in a very poor english. Do not hesitate to send corrections.
- Do not confuse this `export` package (and its `\Import` command) with the `import` package (and its `\import` command) from Donald Arseneau. The two packages have no relation except similarity of names.

4 To Do

- Other parameter sets could be exported the same way as the page layout parameters. Ask if you have needs.
- Export other kinds of parameters (strings, macros, etc.).
- Improve the documentation.

5 Implementation

```
⟨*package⟩
```

`\export` First, we declare the `\export` write channel:

```
\newwrite\export
```

`\openexport` The command `\openexport` announce the beginning of a sequence of exportations and opens the file which will be used to carry the data:

```
\newcommand{\openexport}[2][xpt]{%
\PackageInfo{export}{\MessageBreak
Exportations into #2.#1.\MessageBreak}%
\immediate\openout\export #2.#1}
```

The optionnal argument is the extension added to the name of the file. Default: `xpt`. Using an other extension may be useful for package writers needing more specific names.

`\closeexport` The command `\closeexport` terminates a sequence of exportations by closing the file and announcing. Opening and closing are `\immediate`.

```
\newcommand{\closeexport}{\immediate\closeout\export%
   \PackageInfo{export}{\MessageBreak
   End of exportations.\MessageBreak}}
```

`\ExportLength` And now, the exporting commands. To export a length in points, we just use `\the`⟨*length*⟩ and write it as an argument of a `\setlength` command. This command may be used for rigid and rubber lengths, because `\the` gives the shrinking and stretching parts of a rubber length.

```
\newcommand{\ExportLength}[1]{%
\PackageInfo{export}{Exportation of length #1}%
\immediate\write\export{\string\setlength{\csname#1\endcsname}%
   {\the\csname#1\endcsname}}}
```

`\PreciseExportLength` But for rigid lengths (for rubber ones, you would lose the skrinking and stretching parts), you could use `\PreciseExportLength`, which export the length expressed in scaled points. The trick is to use `\number` in place of `\the`, and adding `sp` for units.

```
\newcommand{\PreciseExportLength}[1]{%
\PackageInfo{export}{Precise exportation of length #1}%
\immediate\write\export{\string\setlength{\csname#1\endcsname}%
   {\number\csname#1\endcsname sp}}}
```

`\ExportMuskip` To export a muskip in mu-s, we just use `\the`⟨*length*⟩ and write it as an argument of a `\setlength` command.

```
\newcommand{\ExportMuskip}[1]{%
\PackageInfo{export}{Exportation of muskip #1}%
\immediate\write\export{\string\setlength{\csname#1\endcsname}%
   {\the\csname#1\endcsname}}}
```

Note that \ExportLength works well with muskips, but \PreciceExportLength does nait and gives an error: another reason to not use that superfluous command.

\Export Exporting a counter is rather trivial, using \@arabic to get its value from the internal name of the LaTeX counter.

```
\newcommand{\Export}[1]{%
\PackageInfo{export}{Exportation of counter #1}%
\immediate\write\export{\string\setcounter{#1}%
   {\@arabic\csname c@#1\endcsname}}}
```

\ExportParameter Exporting a simple parameter is utterly complex: hence contorsions with \expandafter and \noexpand. The \providecommand should avoid errors.

```
\newcommand{\ExportParameter}[1]{%
\PackageInfo{export}{Exportation of parameter #1}%
\immediate\write\export%
   {\string\providecommand{\expandafter\noexpand\csname#1\endcsname}%
   {}}
\immediate\write\export%
   {\string\renewcommand{\expandafter\noexpand\csname#1\endcsname}%
   {\csname#1\endcsname}}}
```

\ExportIf Exporting an "if" condition is far from trivial. We need to protect the contition name and to avoid some expansions.

```
\newcommand{\ExportIf}[1]{%
\PackageInfo{export}{Exportation of the if condition ``#1''}
\expandafter\csname if#1\endcsname
\immediate\write\export{\expandafter\noexpand\csname #1true\endcsname}
\else
\immediate\write\export{\expandafter\noexpand\csname #1false\endcsname}
\fi}
```

\Import Importing the data is easy: open the file for reading, read it: the \setcounter and \setlength commands are executed, then close it.

```
\newcommand{\Import}[2][xpt]{%
\PackageInfo{export}{\MessageBreak
Importations from #2.#1\MessageBreak}%
\InputIfFileExists{#2.#1}{\relax}{\relax}}
```

\xcaptionf Now, the two caption commands. They increment the figure or table counter, then call the internal macro \@xcaption. We provide \xcaption if you have other classes of floats.

```
\def\xcaptionf{\refstepcounter{figure} \@dblarg{\@xcaption{figure}}}
```

\xcaptiont

```
\def\xcaptiont{\refstepcounter{table} \@dblarg{\@xcaption{table}}}
```

\xcaption

```
\def\xcaption#1{\refstepcounter{#1} \@dblarg{\@xcaption{#1}}}
```

\@xcaption

```
\long\def\@xcaption#1[#2]#3{\par\addcontentsline{\csname
  ext@#1\endcsname}{#1}{\protect\numberline{\csname
  the#1\endcsname}{\ignorespaces #2}}%
%  ^^A \begingroup
%  ^^A  \@parboxrestore
%  ^^A  \normalsize
%  ^^A  \@makecaption{\csname fnum@#1\endcsname}{\ignorespaces #3}\par
%  ^^A \endgroup
}
```

\iprotected@write We need these two internal macros for `\AddInputInAux`. They are slighty modified copies of similar macros in the LaTeX kernel.

```
\def\iprotected@write#1#2#3{%
      \begingroup
       \let\thepage\relax
       #2%
       \let\protect\@unexpandable@protect
       \edef\reserved@a{\immediate\write#1{#3}}%
       \reserved@a
      \endgroup
      \if@nobreak\ifvmode\nobreak\fi\fi
}
```

\@inputx

```
\def\@inputx#1{{\def\bibcite##1##2{\relax}\@input{#1}}}
```

This macro write an `\@inputx` of the given file into the `.aux` file.

\AddInputInAux

```
\newcommand{\AddInputInAux}[1]{%
  \iprotected@write\@auxout
      {%
      \let\label\@gobble \let\index\@gobble \let\glossary\@gobble}%
      {\string\@inputx{#1}%
           }%
   }%
```

\ExportPageLayout This macro is just a list of exportations of the lengths defining the page layout, because it would be so boring to type all that! It also exports some parameters and counters related to page layout and float management. Note that `\mathindent` is exported only if it is defined, i.e. when the `fleqn` class option is used.

```
\newcommand{\ExportPageLayout}{%
\PackageInfo{export}{\MessageBreak
Exportation of the PageLayout\MessageBreak}%
\PreciseExportLength{hsize}
\PreciseExportLength{vsize}
\PreciseExportLength{hoffset}
\PreciseExportLength{voffset}
```

```
\PreciseExportLength{linewidth}
\PreciseExportLength{columnwidth}
\PreciseExportLength{columnsep}
\PreciseExportLength{columnseprule}
\PreciseExportLength{parindent}
\ExportLength{parskip}
\PreciseExportLength{hoffset}
\PreciseExportLength{voffset}
\PreciseExportLength{oddsidemargin}
\PreciseExportLength{evensidemargin}
\PreciseExportLength{headheight}
\PreciseExportLength{headsep}
\PreciseExportLength{textheight}
\PreciseExportLength{textwidth}
\PreciseExportLength{paperheight}
\PreciseExportLength{paperwidth}
\PreciseExportLength{marginparsep}
\PreciseExportLength{marginparwidth}
\PreciseExportLength{marginparpush}
\PreciseExportLength{footskip}
\PreciseExportLength{topmargin}
\PreciseExportLength{topskip}
\PreciseExportLength{jot}
\@ifundefined{mathindent}{}{\PreciseExportLength{mathindent}}
\ExportLength{abovedisplayskip}
\ExportLength{belowdisplayskip}
\ExportLength{abovedisplayshortskip}
\ExportLength{belowdisplayshortskip}
\ExportLength{floatsep}
\ExportLength{textfloatsep}
\ExportLength{dblfloatsep}
\ExportLength{dbltextfloatsep}
\ExportLength{intextsep}
\Export{topnumber}
\Export{dbltopnumber}
\Export{bottomnumber}
\Export{totalnumber}
\PreciseExportLength{footnotesep}
\ExportParameter{topfraction}
\ExportParameter{bottomfraction}
\ExportParameter{textfraction}
\ExportParameter{floatpagefraction}
\ExportParameter{dbltopfraction}
\ExportParameter{dblfloatfraction}
\ExportLength{baselineskip}
\ExportLength{normalbaselineskip}
\ExportParameter{baselinestretch}
\PackageInfo{export}{\MessageBreak
End of exportation of the PageLayout\MessageBreak}}%
```

`\ExportArrayParams` This macro is just a list of exportations of the lengths and parameters defining the layout in `array` and `tabular` environments.

```
\newcommand{\ExportArrayParams}{%
\PackageInfo{export}{\MessageBreak
Exportation of the ArrayParams\MessageBreak}%
\PreciseExportLength{arraycolsep}
\PreciseExportLength{tabcolsep}
\PreciseExportLength{arrayrulewidth}
\PreciseExportLength{doublerulesep}
\ExportParameter{arraystretch}
\PackageInfo{export}{\MessageBreak
End of exportation of the ArrayParams\MessageBreak}}%
```