`constants.sty`, a package for automatic numbering of constants *

Antoine Lejay

Released 2008/03/25

Abstract

This set of macros aims to provide a way to number automatically constants in a mathematical proof with a system for labelling/referencing. In addition, several families of constants (with different symbols) may be defined.

Contents

1 Introduction

Some domains of mathematics such as the theory of (partial) differential equations heavily rely on inequalities, and many proofs are done by writing down a long sequence of inequalities involving constants that may change from line to line. Thus, a standard redaction trick consists in starting a proof by the statement

> Let C denotes a constants that varies from line to line.

However, it is sometimes necessary to get more informations about the involved constants. Thus, either one has to finds different symbols, to the risk of introduce some confusions in the notations, or to labels the constants. Of course, manual labelling is not that easy, especially at the early stages of redaction.

The aim of this package is then to provide an automatic way to incrementally label the constants with numbers. In addition, it offers the possibility to label the constants with a mechanism similar to `\label` and `\ref`. Finally, it is possible to define several families of constants with different rules for printing.

*This file describes version v.1, last revised 2008/03/25.

Dependencies: this package uses the `keyval` package of the *graphic bundle* (which is in every standard distribution).

2 Basic usage

`\C` The most basic usage of this package consists in calling the macro `\C` in math mode, that prints C_1 for the first time, C_2 for the second time, ...

`\Cl` The macro `\Cl` is similar to `\C` except that it takes a mandatory argument which is a label's name, for example `\Cl{cst-1}`. The number of the constant which is then printed is recorded and written in the `.aux` file. Note that `\label{cst-1}` does not interact with and `\Cl{cst-1}`, that is the labels for constants are treated in a specific way and can be only accessed through the command `\Cr` below.

`\Cr` The macro `\Cr` takes a mandatory argument which shall be the name of a label defined by `\Cl`. As for standard labels in LaTeX, if `\Cr` calls a label that have not yet been defined are, then the result is replaced by "C^st**??**". A message is written at the end of the compilation to indicate the need of a second compilation to get all the references right.

`\pagerefconstant` This macros takes the label of a constant as a mandatory argument and returns the page were the constant is.

`\resetconstant` This macros resets the counter of constants, so that the next call to `\C` will print again C_1.

3 Family of constants

What if one whishes to use two families of constants $K_{(1)}$, $K_{(2)}$, ... and C_1, C_2, ... in the same document, and that the $K_{(i)}$'s are reseted at each new section? A possibility consists then in defining a new family of constant using the following code in the preamble.

```
\newcommand{\parenthezises}[1]{(\arabic{#1})}
\newconstantfamily{example1}{
symbol=K,
format=\parenthezises,
reset={section}
}
```

3.1 Defining a family

`\newconstantfamily` The macro `\newconstantfamily` allows one to define a new family of constants. Its first argument is the name of the family, while the second argument uses the key-value principle.

The key `symbol` is for the symbol that is used. The key `format` shall takes as argument a command with one argument that can be applied to a counter (such as `\alph`, `\roman`, ...), or any user-defined command such a `\parenthezises` above. The key `reset` takes the name of a counter and thus the counter for this family of constants is reseted each time the corresponding counter is stepped by one. Note that several `reset` key may be used (technically, it adds the name of the counter

to a list of counters to be reseted each time a given counter is stepped). Here, possible names are `section`, `subsection`, `equation`, ...

By default, there is one family which is defined, whose name is `normal`.

If a key is missing, then the corresponding standard values are used (that is `symbol=C`, `format=\arabic` and no `reset` key).

`\renewconstantfamily` If `\newconstantfamily` is called upon a family name that has been already defined, then the compilers stops and returns an error message. The `\renewconstantfamily` (still to be used in the preamble) allows one to override the behavior of a family without getting an arror message. Using `normal` as a family name, one can get replace the standard behavior of `\C`.

3.2 Using a family

`\C` `\Cl` `\Cr` `\pagerefconstant` Using a family is simple: it is sufficient to call `\C` and `\Cl` with an optional argument which is the family name (if the name is incorrect, an error message is printed), that is `\C[example1]` and `\Cl[example1]{cst-2}` to get $K_{(3)}$ and $K_{(4)}$. For `\Cr`, there is no need to call the family, as it is stored with the family name. Thus, `\Cr{cst-2}` will print $K_{(4)}$. The macro `\pagerefconstant` also do not need any reference to a family name.

`\resetconstant` The command `\resetconstant` also accepts the family name as an optional argument.

4 The code

The mechanism of labelling constants is similar to the mechanism used by the standard LaTeX `\label` and `\ref` mechanism (see the files `source2e.pdf`, `latex.ltx` and `ltxref.dtx`), excepted that the family of the constant is taken into account.

This works the following way:

1. The `aux` file is read first. If a line of type

   `\newlabelconstant{`⟨*label*⟩`}{{`⟨*number*⟩`}{`⟨*page*⟩`}{`⟨*family*⟩`}}`

   is found, then a new macro `\cstr@`⟨*label*⟩ is created. This macro shall expand into

   `{{`⟨*number*⟩`}{`⟨*page*⟩`}{`⟨*family*⟩`}}`

   where ⟨*number*⟩ is the number of the constant, ⟨*page*⟩ is the page were it appears and ⟨*family*⟩ is its family name.

2. When a reference ⟨*label*⟩ to a constant is found (call of `\Cr{`⟨*label*⟩`}`), then TeX determines if `\cstr@`⟨*label*⟩ is already defined. If not, then it prints a symbol for unknown constants and send a warning message.

3. When a label to a constant is defined (call of `\Cr[\family]{`⟨*label*⟩`}`), then TeX writes into the `aux` file the corresponding `\newlabelconstant` line.

4. At the end of the document, the `aux` file is read once to determined wether or not there the source file has to be compiled or not.

4.1 Initialization

The initializaton call the package `keyval`.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{constants}%
  [2008/03/25 v.1 Labeling and numbering constants]
\PackageInfo{constants}
  {This package aims to label et number constants in a mathematical proof.}
\RequirePackage{keyval}
```

4.2 Macros associated to constants

When a family ⟨*family*⟩ of constants is defined, several macros are created, that are `\cst@family@`⟨*family*⟩, to check the existence of a family, `\cst@format@`⟨*family*⟩, that takes a counter name as argument, and `\cst@symbol@`⟨*family*⟩, that expand into the symbol of the constant. In addition, a counter `cst@counter@`⟨*family*⟩ is created.

4.3 Macros for testing existence of families

`\@if@constant@exists` When a new family ⟨*family*⟩ is defined, a macro `\cst@family@`⟨*family*⟩ is defined, that expands to nothing.

```
\def\@if@constant@exists#1#2#3{\@ifundefined{cst@family@#1}{#3}{#2}}
```

4.4 The family of constants *normal*

The normal family is the family by default. The corresponding macros are then defined.

```
\global\@namedef{cst@family@normal}{}
\newcounter{cst@counter@normal}
\def\cst@format@normal#1{\arabic{#1}}
\def\cst@symbol@normal{C}
```

4.5 Definition of families

`\newconstantfamily` The macro `\newconstantfamily` defines the macros and counter associated to a family.

```
\newcommand{\newconstantfamily}[2]{
\@if@constant@exists{#1}{%
\PackageError{constants}{The family of constants '#1' already exists}{%
Use \protect\renewconstantfamily\space to override}}%
{%
\expandafter\def\csname cst@family@#1\endcsname{}
\expandafter\def\csname cst@format@#1\endcsname{\cst@format@normal}
\expandafter\def\csname cst@symbol@#1\endcsname{\cst@symbol@normal}
\expandafter\newcounter{cst@counter@#1}
\define@key{constants}{format}%
{\expandafter\def\csname cst@format@#1\endcsname{##1}}
\define@key{constants}{symbol}%
{\expandafter\def\csname cst@symbol@#1\endcsname{##1}}
\define@key{constants}{reset}{\@addtoreset{cst@counter@#1}{##1}}
```

```
\setkeys{constants}{#2}
}}
```

\renewconstantfamily The \renewconstantfamily also redefines the macros associated to constants, but does not re-create the counter.

```
\newcommand{\renewconstantfamily}[2]{
\@if@constant@exists{#1}{
\define@key{constants}{format}{%
\expandafter\def\csname cst@format@#1\endcsname{##1}}
\define@key{constants}{symbol}{%
\expandafter\def\csname cst@symbol@#1\endcsname{##1}}
\define@key{constants}{reset}{\@addtoreset{cst@counter@#1}{##1}}
\setkeys{constants}{#2}
}{%
\PackageError{constants}{The family of constants '#1' already exists}{%
Use \protect\renewconstantfamily\space to override}}%
}
```

The macros \newconstantfamily and \renewconstantfamily can only be called in the preamble.

```
\@onlypreamble\newconstantfamily
\@onlypreamble\renewconstantfamily
```

4.6 Reading constants informations

The following macros aims to read the informations from the macro \cstr@⟨*label*⟩ (See Section 4).

When constants informations are read from the aux file, the format, deduced from the family name ⟨*family*⟩, is put in \cst@tmp@format and the symbol is put in \cst@tmp@symbol. By default, these macros expand into \cst@undefined@format and \cst@undefined@symbol.

```
\def\G@refundefinedconstanttrue{%
  \gdef\@refundefinedconstant{%
    \@latex@warning@no@line{%
There were undefined references to constants}}}
\let\@refundefinedconstant\relax
\def\cst@tmp@format{cst@undefined@format}
\def\cst@tmp@symbol{cst@undefined@symbol}
```

\@firstofthree \@secondofthree \@thirdofthree These macros return respectively their first, second and third argument.

```
\long\def\@firstofthree#1#2#3{#1}
\long\def\@secondofthree#1#2#3{#2}
\long\def\@thirdofthree#1#2#3{#3}
```

If a reference to a macro is not defined, then the counter value is not expanded, while a symbol "Cst??" is returned.

```
\def\cst@undefined@format#1{}
\def\cst@undefined@symbol{%
\nfss@text{\reset@font\textrm{C\textsuperscript{st}\textbf{??}}}}
```

\@setrefconstant This macro expands into one of the arguments of the constant. The argument #1 shall be of type cstr@⟨*label*⟩, the argument #2 shall then be \@firstofthree, \@secondofthree and \@thirdofthree. The third argument #3 expands into

{{⟨*number*⟩}{⟨*page*⟩}{⟨*family*⟩}}. If `cstr@`⟨*label*⟩ is not defined, then the package send a warning and the macro expands to "**??**".

```
\def\@setrefconstant#1#2#3{%
  \ifx#1\relax
   \protect\G@refundefinedconstanttrue
   \nfss@text{\reset@font\bfseries ??}%
   \@latex@warning{Reference to constant `#3' on page \thepage \space
             undefined}%
  \else
   \expandafter#2#1\null
  \fi}
```

`\@setfamconstant` This macro is similar to `\@setrefconstant` but initializes `\cst@tmp@format` and `\cst@tmp@symbol` to `\cst@format@`⟨*family*⟩ and `\cst@fsymbol@`⟨*family*⟩.

```
\def\@setfamconstant#1#2#3{%
\ifx#1\relax%
\def\cst@tmp@format{cst@undefined@format}\null
\def\cst@tmp@symbol{cst@undefined@symbol}\null
\else%
\def\cst@tmp@format{cst@format@\expandafter#2#1}\null
\def\cst@tmp@symbol{cst@symbol@\expandafter#2#1}\null
\fi}
```

`\@setcounterconstant` This macro is similar to `\@setrefconstant` but set the temporary counter `cst@tmp@counter` to the value of ⟨*number*⟩.

```
\newcounter{cst@tmp@counter}
\def\@setcounterconstant#1#2#3{%
  \ifx#1\relax
   \protect\G@refundefinedconstanttrue
   \@latex@warning{Reference to constant `#3' on page \thepage \space
             undefined}%
  \else
    \setcounter{cst@tmp@counter}{\expandafter#2#1}\null
\fi}
```

`\refconstant` `\familyconstant` `\counterconstant` These macros are used to call `\@setrefconstant`, `\@setfamconstant` and `\@setcounterconstant`.

```
\def\refconstant#1{%
\expandafter\@setrefconstant\csname cstr@#1\endcsname\@firstofthree{#1}}
\def\familyconstant#1{%
\expandafter\@setfamconstant\csname cstr@#1\endcsname\@thirdofthree{#1}}
\def\counterconstant#1{%
\expandafter\@setcounterconstant%
\csname cstr@#1\endcsname\@firstofthree{#1}}
```

4.7 Calling constants

`\Cr` The macro is used to call a referenced constant. Using `\counterconstant` and `\familyconstant`, the macros `\cst@tmp@symbols`, `\cst@tmp@format` and the counter `cst@tmp@counter` have been properly initialized.

```
\newcommand{\Cr}[1]{%
\counterconstant{#1}%
\familyconstant{#1}%
{\@nameuse{\cst@tmp@symbol}}_{\@nameuse{\cst@tmp@format}{cst@tmp@counter}}}
```

\C This macro checks the existence of a family of constants and then expand to the symbol with the value of the counter.

```
\newcommand{\C}[1][normal]{%
\@if@constant@exists{#1}{%
\expandafter\refstepcounterconstant{cst@counter@#1}%
{\@nameuse{cst@symbol@#1}}_{\@nameuse{cst@format@#1}{cst@counter@#1}}}%
{\PackageError{constants}{Family for constants '#1' not defined}{
Check the name or use \protect\newconstantfamily}}%
}
```

\Cl This macro calls \C and then call \labelconstant to add the constant label into the aux file.

```
\newcommand{\Cl}[2][normal]{\C[#1]\labelconstant{#2}{\string #1}}
```

\pagerefconstant This macro is similar to refconstant but returns the page number.

```
\def\pagerefconstant#1{%
\expandafter\@setrefconstant\csname cstr@#1\endcsname
   \@secondofthree{#1}}
```

\refstepcouterconstant This macro is used to increment the number of the constants.

```
\def\refstepcounterconstant#1{\stepcounter{#1}%
    \protected@edef\@currentlabelconstant
       {\csname p@#1\endcsname\csname the#1\endcsname}%
}
```

4.8 Writing informations in the aux file

\@newl@belconstant This macro checks the existence of #1@#2 (here cstr is passed as argument #1 and ⟨*label*⟩ is passed as argument #2). If #1@#2 has already been defined, then it claims that the label already exists. then it creates this macros that expands into #3, which shall then expand into

{{⟨*number*⟩}{⟨*page*⟩}{⟨*family*⟩}}

This macro is called by \newlabelconstant and is executed only when reading the aux file at the beginning of the page processing. In addition, it can only be called in the preamble.

```
\def\@newl@belconstant#1#2#3{{%
  \@ifundefined{#1@#2}%
    \relax
    {\gdef \@multiplelabelsconstant {%
       \@latex@warning@no@line{%
There were multiply-defined labels for constants}}%
     \@latex@warning@no@line{Label for constant `#2' multiply defined}}%
  \global\@namedef{#1@#2}{#3}
}}
\def\@currentlabelconstant{}
\def\newlabelconstant{\@newl@belconstant{cstr}}
\@onlypreamble\@newl@belconstant
\let \@multiplelabelsconstant \relax
```

\labelconstant This macro is called when a new label is created and writes the corresponding informations into the aux file.

```
\def\labelconstant#1#2{\@bsphack
  \protected@write\@auxout{}%
```

```
{\string\newlabelconstant{#1}{{\@currentlabelconstant}{\thepage}{#2}}}%
 \@esphack}
```

The following codes corresponds to the final check to determine if a supplementary compilation is needed.

```
\AtEndDocument{
\clearpage
\let\@newl@belconstant\@empty
\begingroup
    \if@filesw
      \immediate\closeout\@mainaux
      \let\@setckpt\@gobbletwo
      \let\@newl@belconstant\@testdef
      \let\@newl@bel\@testdef
      \@tempswafalse
      \makeatletter \input\jobname.aux
    \fi
\@refundefinedconstant
    \if@filesw
      \ifx \@multiplelabelsconstant \relax
        \if@tempswa
          \@latex@warning@no@line{%
Label(s) for constants may have changed.
                Rerun to get cross-references right}%
        \fi
      \else
        \@multiplelabelsconstant
      \fi
    \fi
\endgroup
\deadcycles\z@\@@end
}
```

4.9 Reseting constants

\resetconstant The macro \resetconstant set the counter cst@counter@⟨*family*⟩ to 0.

```
\newcommand{\resetconstant}[1][normal]{%
\@if@constant@exists{#1}{%
\setcounter{cst@counter@#1}{0}}{%
{\PackageError{constants}{Family for constants '#1' not defined}{%
Check the name or use \protect\newconstantfamily}}%
}}
```

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.