

units.sty – nicefrac.sty

Axel Reichert

1998-06-25

Abstract

`units.sty` is a package for setting units in a typographically correct way. It is based upon `nicefrac.sty`, a package for nice fractions. See the files `README` and `COPYING` for additional information.

1 Loading

Only nice fractions: `\usepackage{nicefrac}`

Only units or both: `\usepackage{units}`

2 Options

Tight spacing for units (default): `\usepackage[tight]{units}`

Loose spacing for units: `\usepackage[loose]{units}`

“Nice” fractions (default): `\usepackage[nice]{nicefrac}`

“Ugly” fractions: `\usepackage[ugly]{nicefrac}`

The options `nice` and `ugly` can also be used for the `units` package, they will simply be passed to the `nicefrac` package, so you can combine the options, e. g.: `\usepackage[loose,ugly]{units}`

Tight spacing means `\`, for the space between the value and the dimension, loose spacing uses `~`, like 1 m and 1 m. Nice fractions look like $\frac{m}{s}$, ugly fractions look like m/s in text mode and $\frac{m}{s}$ in math mode.

3 Commands

Units: $\backslash\text{unit}[\langle val \rangle]\{\langle dim \rangle\}$
Fractions of units: $\backslash\text{unitfrac}[\langle val \rangle]\{\langle num \rangle\}\{\langle denom \rangle\}$
Nice fractions: $\backslash\text{nicefrac}[\langle fontcmd \rangle]\{\langle num \rangle\}\{\langle denom \rangle\}$

In these list, $\langle val \rangle$ and $\langle dim \rangle$ denote the value and the dimension of the unit, respectively. $\langle num \rangle$ and $\langle denom \rangle$ are the numerator and denominator of the fraction, and $\langle fontcmd \rangle$ can be an author command for fonts or a math alphabet, see `fntguide.dvi`.

Typically, $\langle val \rangle$ is only a number and $\langle num \rangle$ and $\langle denom \rangle$ are relatively simple L^AT_EX expressions. If you really feel the need for putting whole paragraphs or complex formula into such a fraction, you have misunderstood the purpose of this package. (-;

It is very important to be aware of the fact that all these commands distinguish between text mode and math mode. Within text mode, the font of the surrounding text will be used by default, for math mode `\mathrm` is the default.

This is quite sensible, because you would not want your collection of delicious recipes (typeset in a mega-cool ultra condensed bold italic calligraphical font) contain those spindle Computer Modern Roman just for half a litre of milk. (-;

Otherwise, when working on scientific papers, strict and consistent notation is really a virtue, and, like it or not, units are typeset with upright fonts. So take great care when deciding about math mode or not.

4 Examples

<code>\sffamily\bfseries\unit{m}</code>	m
<code>\sffamily\bfseries\$\unit{m}\$</code>	m
<code>\sffamily\itshape\unit[1]{m}</code>	<i>1 m</i>
<code>\sffamily\itshape\$\unit[1]{m}\$</code>	1 m
<code>\sffamily\bfseries\unitfrac{m}{s}</code>	m/s
<code>\sffamily\bfseries\$\unitfrac{m}{s}\$</code>	m/s
<code>\sffamily\itshape\unitfrac[1]{m}{s}</code>	<i>1 m/s</i>
<code>\sffamily\itshape\$\unitfrac[1]{m}{s}\$</code>	1 m/s

As you can see, font changes are ignored in math mode ...

<code>\scriptsize\sffamily\itshape\unitfrac[1]{m}{s}</code>	<i>1 m/s</i>
<code>\scriptsize\sffamily\itshape\$\unitfrac[1]{m}{s}\$</code>	1 m/s

... except for the fontsize.

<code>\bfseries\itshape\nicefrac{1}{2}</code>	$1/2$
<code>\bfseries\itshape\$\nicefrac{1}{2}\$</code>	$1/2$
<code>\nicefrac[\texttt]{1}{2}</code>	$1/2$
<code>\nicefrac[\texttt]{\textit{1}}{2}</code>	$1/2$
<code>\$\nicefrac[\mathcal]{A}{B}\$</code>	\mathcal{A}/\mathcal{B}

The `\nicefrac` command can deal even with quite strange font changing commands.

5 Typography

Why should units be typeset in upright fonts, not in italics? Because they have to be distinguished from normal variables: “m” is meter, “*m*” is a variable, for example a mass.

Why should the space between the value and the dimension be non-breakable? Because the reader is disturbed by linebreaks like 1 m.

Why should the space between the value and the dimension be a half word space only? Because things belonging together are typeset tighter. Compare 1 m with 1 m and the normal word spacing, which can vary from line to line.

Why should nice fractions be typeset so that the numerator does not extend above the height of the letter “M” and the denominator does not extend below the baseline? Because otherwise a stretching of the baselineskip could be necessary due to descenders.

By the way, a very common mistake is to place units into brackets, like [N]. The correct notation is $[F] = N$. The brackets indeed are a function with a variable as an argument. The value returned is the unit. If you need to specify units in table headings, then use a single row for the units, where you put them into *parentheses* instead.

6 Bugs

None, are you kidding?

7 Features

- Consistent and logical markup of units is enhanced instead of fiddling around with spacing and fonts.
- The same command works in text mode *and* math mode, font and size are adjusted automatically for nice integration within text mode while forcing strict notation in math mode.
- Basic requirements of typography are fulfilled: Forbidden line-breaks, correct spacing and the numerator automatically aligning with the height of an “M”.
- Easy configuration by use of package options.

8 Bugs again (-;

Ok, ok, you got me.

- Fonts without “M” do not work correctly. Do you know one?
- Fractions in `\scriptscriptstyle` look ugly because they exceed the height of an “M”. As far as I know this is a \LaTeX problem, because there are no smaller math fonts available.
- The kerning between numerator and slash or denominator and slash is bad. In fact, there is none.
- The combination of the `ugly` option with text mode can lead to ambiguous fractions. Be happy that a warning is issued. Why do you use this option? Your boss? Ah, I see.

9 Implementation

9.1 Documentation Driver

```
1 <*driver>
2 \documentclass[12pt,a4paper]{ltxdoc} \usepackage{units}
3 \begin{document}
4   \DocInput{units.dtx}
5 \end{document}
6 </driver>
```

9.2 units.sty

```
7 <*units>
```

9.2.1 Identification

As this package uses the `\RequirePackage` command, it does not work with older L^AT_EX 2_ε versions.

```
8 \NeedsTeXFormat{LaTeX2e}[1995/12/01]
```

The package identifies itself with its release date, a version number, and a short description.

```
9 \ProvidesPackage{units}[1998/06/25 v0.9a Typesetting units]
```

9.2.2 Initialization

The `ifthen` package is loaded because a new boolean variable has to be declared.

```
10 \RequirePackage{ifthen}
11 \newboolean{B@UnitsLoose}
```

9.2.3 Option Declaration

This boolean variable is set according to the package options.

```
12 \DeclareOption{tight}{\setboolean{B@UnitsLoose}{false}}
13 \DeclareOption{loose}{\setboolean{B@UnitsLoose}{true}}
```

Other options will be passed to the `nicefrac` package.

```
14 \DeclareOption*{%
15   \PassOptionsToPackage{\CurrentOption}{nicefrac}%
16 }
```

9.2.4 Option Processing

If no options are specified, tight spacing between the value and the dimension of a unit is used by default. Otherwise the options are processed in the order given by the calling command.

```
17 \ExecuteOptions{tight}
18 \ProcessOptions*
```

9.2.5 Loading Files

Because the macro used for fractions of units is also helpful for other nice fractions, it is build into a stand-alone package called `nicefrac`, which therefore is required by `units.sty`.

```
19 \RequirePackage{nicefrac}[1998/06/25]
```

9.2.6 Defining Commands

In order to make debugging easier for the package user (missing braces or brackets), all commands are declared with the `*-form` of `\DeclareRobustCommand`. The `*-form` does not allow “long” arguments for the commands, so e.g. `\par` commands are forbidden inside the arguments. This should be no serious restriction. (-;

`\unit` First, it is checked if the optional argument is empty.¹

```
20 \DeclareRobustCommand*\unit}[2] [] {%
21   \begingroup
22   \def\0{#1}%
23   \expandafter
24   \endgroup
```

If it is, nothing happens.

```
25 \ifx\0\@empty
```

Otherwise it is typeset and followed by a space depending on the boolean declared above.

```
26 \else
27   #1%
28   \ifthenelse{\boolean{B@UnitsLoose}}{~}{\,%}
29 \fi
```

The use of upright fonts for the dimension is forced in math mode. Units in text mode are typeset with the currently active font.

```
30 \ifthenelse{\boolean{mmode}}{\mathrm{#2}}{#2}%
31 }
```

`\unitfrac` Fractions of units are typeset by means of the `\nicefrac` command. Upright fonts are forced in math mode by use of the optional argument. The rest of code is identical to the former macro.

```
32 \DeclareRobustCommand*\unitfrac}[3] [] {%
33   \begingroup
34   \def\0{#1}%
35   \expandafter
36   \endgroup
37   \ifx\0\@empty
38   \else
39     #1%
40     \ifthenelse{\boolean{B@UnitsLoose}}{~}{\,%}
41   \fi
```

¹Credits go to BERND RAICHLE for supplying this method.

```

42 \ifthenelse{\boolean{mmode}}{%
43   \nicefrac[\mathrm]{#2}{#3}%
44 }%
45 {%
46   \nicefrac{#2}{#3}%
47 }%
48 }
49 </units>

```

9.3 nicefrac.sty

```
50 <*nicefrac>
```

9.3.1 Identification

```

51 \NeedsTeXFormat{LaTeX2e}[1995/12/01]
52 \ProvidesPackage{nicefrac}[1998/06/25 v0.9a Nice fractions]

```

9.3.2 Initialization

First, some new lengths are allocated.

```

53 \newlength{\L@UnitsRaiseDisplaystyle}
54 \newlength{\L@UnitsRaiseTextstyle}
55 \newlength{\L@UnitsRaiseScriptstyle}

```

9.3.3 Loading Files

The `ifthen` package is loaded for easier handling of decisions.

```
56 \RequirePackage{ifthen}
```

9.3.4 Defining Commands

`\@UnitsNiceFrac` Now the code used for nice fractions in math mode:

```

57 \DeclareRobustCommand*\@UnitsNiceFrac[3] []{%
58   \ifthenelse{\boolean{mmode}}{%

```

For each of the four different math styles the “M” height is measured, taking specified font changing commands into account.

```

59   \settoheight{\L@UnitsRaiseDisplaystyle}{%
60     \ensuremath{\displaystyle#1{M}}%
61   }%
62   \settoheight{\L@UnitsRaiseTextstyle}{%
63     \ensuremath{\textstyle#1{M}}%
64   }%
65   \settoheight{\L@UnitsRaiseScriptstyle}{%

```

```

66     \ensuremath{\scriptstyle#1{M}}}%
67   }%
68   \settoheight{\@tempdima}{%
69     \ensuremath{\scriptscriptstyle#1{M}}}%
70   }%

```

The raise height is calculated by taking the difference between the height of a normal “M” and the height of an “M” set in the correct numerator size.

```

71   \addtolength{\L@UnitsRaiseDisplaystyle}{%
72     -\L@UnitsRaiseScriptstyle%
73   }%
74   \addtolength{\L@UnitsRaiseTextstyle}{%
75     -\L@UnitsRaiseScriptstyle%
76   }%
77   \addtolength{\L@UnitsRaiseScriptstyle}{-\@tempdima}%

```

The height of the numerator above the baseline is dependent on the actual fontsize within the math environment.² The numerator is put into an appropriately raised box, within math mode (the additional `\ensuremath` is necessary, because the `\raisebox` command leaves math mode).

The fontsize for the numerator part is `\scriptstyle` within `\displaystyle` and `\textstyle`, but `\scriptscriptstyle` within `\scriptstyle` and `\scriptscriptstyle` context.

Then the numerator is typeset either with the upright math font or using the font command specified in the optional argument (if it is given). Only the math alphabet commands may be used: `\mathnormal`, `\mathrm`, `\mathbf`, `\mathsf`, `\mathit`, `\mathtt`, and `\mathcal`.

```

78   \mathchoice
79     {%
80       \raisebox{\L@UnitsRaiseDisplaystyle}{%
81         \ensuremath{\scriptstyle#1{#2}}}%
82       }%
83     }%
84     {%
85       \raisebox{\L@UnitsRaiseTextstyle}{%
86         \ensuremath{\scriptstyle#1{#2}}}%
87       }%
88     }%
89     {%

```

²Credits go to WOLFGANG MAY for telling me of the `\mathchoice` command.

```

90     \raisebox{\L@UnitsRaiseScriptstyle}{%
91     \ensuremath{\scriptscriptstyle#1{#2}}}%
92     }%
93     }%

```

Within `\scriptscriptstyle` context, L^AT_EX does *not* use even smaller fonts for numerator or denominator. Following the above calculation scheme would lead to a raise height of 0pt. Because this could make fractions at least ambiguous (if not wrong, like J/kg · K for specific thermal capacity), the numerator in `\scriptscriptstyle` context is raised by the same amount as in `\scriptstyle` context and so exceeds the height of an adjacent “M”. This is ugly, but cannot be avoided.

```

94     {%
95     \raisebox{\L@UnitsRaiseScriptstyle}{%
96     \ensuremath{\scriptscriptstyle#1{#2}}}%
97     }%
98     }%

```

The slash is typeset with reduced spacing to the numerator and to the denominator.

```

99     \mkern-2mu/\mkern-1mu%

```

The denominator needs a separate group to delimit the range of the font commands. The choice for the fontsize and the font changing are made just like for the numerator.

```

100    \begingroup
101    \mathchoice
102    {\scriptstyle}%
103    {\scriptstyle}%
104    {\scriptscriptstyle}%
105    {\scriptscriptstyle}%
106    #1{#3}%
107    \endgroup
108    }%

```

Now the code for nice fractions in text mode:

```

109    {%

```

First, the height of a normal “M” is measured.

```

110    \settoheight{\L@UnitsRaiseTextstyle}{#1{M}}%

```

L^AT_EX offers no commands for relative font sizes. To make the macro work for example within footnotes or chapter headings, it is necessary to avoid absolute sizing commands like `\normalsize` or `\scriptsize`. To obtain a font size that suits for the numerator and denominator,

some code was taken from the L^AT_EX kernel (`latex.ltx`, definition of `\textsuperscript`).

In all these measurements the font changing effect of the optional argument is taken into account. Only the following font commands may be used: `\textnormal`, `\textrm`, `\textsf`, `\texttt`, `\textbf`, `\textmd`, `\textup`, `\textit`, `\textsc`, and `\textsl`.

```

111   \settoheight{\@tempdima}{%
112     \ensuremath{%
113       \mbox{\fontsize\sf@size\z@\selectfont#1{M}}}%
114     }%
115   }%

```

Both lengths are subtracted. The result is used for the raise of the numerator.

```

116   \addtolength{\L@UnitsRaiseTextstyle}{-\@tempdima}%

```

Finally, the numerator is typeset in “relative scriptsize” either with the normal text font or using the font command specified in the optional argument (if it is given).

```

117   \raisebox{\L@UnitsRaiseTextstyle}{%
118     \ensuremath{%
119       \mbox{\fontsize\sf@size\z@\selectfont#1{#2}}}%
120     }%
121   }%

```

The code for the slash and the denominator should now be straight forward.

```

122   \ensuremath{\mkern-2mu}/\ensuremath{\mkern-1mu}%
123   \ensuremath{%
124     \mbox{\fontsize\sf@size\z@\selectfont#1{#3}}}%
125   }%
126 }%
127 }

```

`\@UnitsUglyFrac` Ugly fractions in math mode ...

```

128 \DeclareRobustCommand*\@UnitsUglyFrac[3] []{%
129   \ifthenelse{\boolean{mmode}}{%
... are typeset just like normal fractions.
130     \frac{#1{#2}}{#1{#3}}%
131   }%

```

In text mode ...

```
132  {%
```

...they are typeset like J/kg·K.

```
133    #1{#2}/#1{#3}%
```

Because the fraction can get ambiguous or even wrong, a warning message is issued.

```
134    \PackageWarning{nicefrac}{%
```

```
135      You used \protect\nicefrac\space or
```

```
136      \protect\unitfrac\space in text mode\MessageBreak
```

```
137      and specified the ‘‘ugly’’ option.\MessageBreak
```

```
138      The fraction may be ambiguous or wrong.\MessageBreak
```

```
139      Please make sure the denominator is
```

```
140      correct.\MessageBreak
```

```
141      If it is, you can safely ignore\MessageBreak
```

```
142      this warning
```

```
143    }%
```

```
144  }%
```

```
145 }
```

9.3.5 Option Declaration

Dependent on the package options, the `\nicefrac` command typesets either nice or ugly fractions by means of the corresponding macros.

```
146 \DeclareOption{nice}{%
```

```
147   \DeclareRobustCommand*\nicefrac{\@UnitsNiceFrac}%
```

```
148 }
```

```
149 \DeclareOption{ugly}{%
```

```
150   \DeclareRobustCommand*\nicefrac{\@UnitsUglyFrac}%
```

```
151 }
```

9.3.6 Option Processing

If no options are specified, nice fractions are used by default. Otherwise the options are processed in the order given by the calling command.

```
152 \ExecuteOptions{nice}
```

```
153 \ProcessOptions*
```

```
154 </nicefrac>
```

References

- [1] GOOSSENS, M.; MITTELBACH, F.; SAMARIN, A.: *Der L^AT_EX-Begleiter*. Addison-Wesley, Bonn, 1st edn., 1994.
- [2] KOPKA, H.: *Einführung*, vol. 1 of *L^AT_EX*. Addison-Wesley, Bonn, 1st edn., 1994.
- [3] KOPKA, H.: *Ergänzungen – mit einer Einführung in METAFONT*, vol. 2 of *L^AT_EX*. Addison-Wesley, Bonn, 1st edn., 1995.
- [4] KOPKA, H.: *Erweiterungen*, vol. 3 of *L^AT_EX*. Addison-Wesley-Longman, Bonn, 1997.
- [5] WILLBERG, H. P.; FORSSMAN, F.: *Lesetypographie*. Schmidt, Mainz, 1997.