

# The cooking-units package\*

Ben Vitecek  
b.vitecek@gmx.at

August 31, 2016

## Abstract

This package enables user to globally format units and to switch between them. It should be used for light-hearted things like cookery books (and not e.g. scientific texts).<sup>1</sup>

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Important note</b>	<b>2</b>
<b>3</b>	<b>The Commands</b>	<b>2</b>
<b>4</b>	<b>Predefined units &amp; some notes</b>	<b>4</b>
<b>5</b>	<b>Defining units</b>	<b>4</b>
<b>6</b>	<b>Defining options</b>	<b>4</b>
<b>7</b>	<b>Language support</b>	<b>9</b>
<b>8</b>	<b>Options</b>	<b>11</b>
	8.1 Load time options . . . . .	12
	8.2 Normal options . . . . .	12
	8.3 Weird options . . . . .	17
<b>9</b>	<b>Bugs &amp; Feedback</b>	<b>17</b>

---

\*This document corresponds to Benedikt Vitecek v1.00, dated 2016/08/31.

<sup>1</sup>I did hide some grammatical and spelling errors for easter egg hunters ☺.

## 1 Introduction

While writing on a cookery book I used – for reasons whatsoever – three different units for weight: kilogram (kg), gram (g) and decagram (dag). Later my mother told me that she doesn't like it if a cookery book uses more than two different units (for weight in this case). Happily I hardly used Decagram and therefore didn't have many problems changing the units. But, well ... I am using  $\text{\LaTeX}$  and changing those units by hand seemed not very  $\text{\LaTeX}$ like, so I started writing some code to convert units. I expanded the code, rewrote it in  $\text{\LaTeX}3$  (which is much more pleasant than  $\text{\LaTeX}2_{\epsilon}$ ) and here it is.

## 2 Important note

This package uses the `translator` package to be able to switch between different languages (german, english and french by now). To do that `translator` has to know which languages are used. The (I think) easiest way is to specify used languages using the optional argument of the `documentclass` (you can do this for both `babel` and `polyglossia`<sup>2</sup>):

```
\documentclass[english,french,ngerman]{class}
\usepackage[main=english]{babel}
\usepackage{cooking-units}
...
```

or if you are using `polyglossia`

```
\documentclass[french,ngerman,english]{class}

\usepackage{polyglossia}
\setmainlanguage{english}
\setotherlanguages{german,french}

\usepackage{cooking-units}
...
```

At least I hope that this works, dealing with languages is a pain in the ass<sup>3</sup>.

## 3 The Commands

This package offers the following commands for unit printing (and converting):

- `\cunum` [*options*]{*amount*}[*space*]{*unit-key*}
- `\cutext` [*options*]{*amount*}{*unit-key*}
- `\Cutext` [*options*]{*amount*}{*unit-key*}

---

<sup>2</sup>I know that `polyglossia` doesn't support the `babel`-way of setting the language by optional argument, but it doesn't harm.

<sup>3</sup>If you excuse me being blunt about this

- `\cfrac{⟨input⟩}`

Numbers and units are printed using `\cunum`. The numerical part can interpret `_` and `/` as (mixed) fractions and `--` as a separator for ranges; to convert units use the option `⟨old-unit⟩=⟨new-unit⟩`<sup>4</sup>. It furthermore allows the sign `?` to be used as a placeholder for not known amounts and raises a warning to remind that this amount needs a checkup<sup>5</sup>. [`⟨space⟩`] adds a space between the number and the unit using `\phantom`.

For a list of predefined units have a look at [table 2 on page 5](#).

1 kg	<code>\cunum{1}{kg}\</code>
2.3 kg	<code>\cunum{2.3}{kg}\</code>
2.3 kg	<code>\cunum{2,3}{kg}\</code>
2–3 kg	<code>\cunum{2--3}{kg}\</code>
2.5–3.5 kg	<code>\cunum{2.5--3.5}{kg}\</code>
2500–3500 g	<code>\cunum[kg=g]{2.5--3.5}{kg}\</code>
392 °F	<code>\cunum[C=F]{200}{C}\</code>
356–392 °F	<code>\cunum[C=F]{180--200}{C}\</code>
$\frac{1}{2}$ m	<code>\cunum{1/2}{m}\</code>
$1\frac{1}{2}$ m	<code>\cunum{1_1/2}{m}\</code>
$1\frac{1}{2}$ m	<code>\cunum[m=cm]{1_1/2}{m}\</code>
? l	<code>\cunum{?}{l}\</code>
50 dag	<code>\cunum{50}{dag}\</code>
5 dag	<code>\cunum{5}[0]{dag}\</code>
1.12 m	<code>\cunum{1.1234}{m}</code>

Decimal numbers are automatically rounded to 2 digits after the colon, temperatures (C, F, K and Re) are automatically rounded to integers.<sup>6</sup>

`\cutext` and `\Cutext` print the number and the written name of the unit. Furthermore, if the package option `use-numerals` is used, integers below a specific integer (by default 13; see `use-numerals-below`) are also written out with `\Cutext` capitalizing the first letter (using package `fmtcount`). Conversion between units is not supported. Without `use-numerals`:

1 litre	<code>\cutext{1}{l}\</code>
1 litre	<code>\Cutext{1}{l}\</code>
12 litres	<code>\cutext{12}{l}\</code>
13 litres	<code>\Cutext{13}{l}</code>

and using package option `use-numerals=true`

one litre	<code>\cutext{1}{l}\</code>
One litre	<code>\Cutext{1}{l}\</code>
twelve litres	<code>\cutext{12}{l}\</code>
13 litres	<code>\Cutext{13}{l}</code>

<sup>4</sup>New keys can be added and defined, see [4 on the following page](#) and [5 on the next page](#) for further information.

<sup>5</sup>You can customize this behavior, see [8 on page 11](#)

<sup>6</sup>You can – of course – change this behavior, see [8 on page 11](#).

`\cfrac` can be used to print fractions. Like in `\cunum` `_` and `/` are used to imply a (mixed) fraction:

$\frac{3}{2/3}$	<code>\cfrac{3}\{2/3\}</code>
$1\frac{2}{3}$	<code>\cfrac{1_2/3}</code>

## 4 Predefined units & some notes

In [table 2 on the next page](#) you can find all predefined units.

I did not include a separate key for “Prise” (pinch), but you can use `Msp` (Messerspitze) and modify it to fit your purpose<sup>7</sup>. In other languages the nearest equivalent of “Messerspitze” is “pinch” which I used.

## 5 Defining units

New units can be defined using `\newcookingunit`:

---

<code>\newcookingunit</code>	<code>\newcookingunit[\langle symbol \rangle]{\langle new-unit-key \rangle}</code>
------------------------------	--

---

This command defines the new unit  $\langle new-unit-key \rangle$ . If the key is not the same as the printed symbol use  $[\langle symbol \rangle]$ .

Some examples (note: the definition of the printed degree Celsius is directly copied & pasted from [a maybe older version of] `siunitx`):

```
\newcookingunit{kg}
\newcookingunit{g}
\newcookingunit[Msp.] {Msp}
\newcookingunit[\ensuremath{\{ }^{\circ}\kern-\scriptspace C}]{C}
```

## 6 Defining options

Options (to change units) can be newly defined or added to already existing keys using

- `\cdefinekeys`
- `\cdefinesinglekey`
- `\cuaddkeys`
- `\cuaddsinglekeys`
- `\cuaddtokeys`

---

<sup>7</sup>See [7 on page 9](#).

unit name	unit-key	symbol
kilogramme	kg	kg
decagramme	dag	dag
gramme	g	g
ounce	oz	oz
pound	lb	lb
degree Celsius	C	°C
degree Fahrenheit	F	°F
degree Réaumur	Re	°Ré
kelvin	K	K
day	d	d
hour	h	h
minute	min	min
second	s	s
metre	m	m
decimetre	dm	dm
centimetre	cm	cm
millimetre	mm	mm
inch	in	in
litre	l	l
decilitre	dl	dl
centilitre	cl	cl
millilitre	ml	ml
calorie	cal	cal
kilocalorie	kcal	kcal
joule	J	J
kilojoule	kJ	kJ
electron volt	eV	eV
pinch	Msp	pn
tablespoon	EL	tsp.
teaspoon	TL	tbsp.

Table 1: List of predefined unit-keys. Note that “electron volt” exists just for fun.

unit-key	symbol
eVc-2	$eV/c^2$
hbareV-1	$\hbar/eV$
chbareV-1	$c\hbar/eV$
(chbareV-1)3	$c^3\hbar^3/eV^3$

Table 2: List of nonsense units (exist just for fun, there will be no support for those units).

---

<code>\cdefinekeys</code> <code>\cdefinesinglekey</code>	<code>\cdefinekeys{⟨unit-key-1⟩}</code> <code>{</code> <code>  {⟨unit-key-2⟩} {⟨1 unit-key-1 are ... unit-key-2⟩}</code> <code>  {⟨unit-key-3⟩} {⟨1 unit-key-1 are ... unit-key-3⟩}</code> <code>  {⟨unit-key-4⟩} {⟨1 unit-key-1 are ... unit-key-4⟩}</code> <code>  ...</code> <code>}</code> <code>\cdefinesinglekey{⟨unit-key-1⟩}</code> <code>{</code> <code>  {⟨unit-key-2⟩} {⟨1 unit-key-2 are ... unit-key-1⟩}</code> <code>  {⟨unit-key-3⟩} {⟨1 unit-key-3 are ... unit-key-1⟩}</code> <code>  ...</code> <code>}</code>
---	--

---

If you define new units (see 5 on page 4) and cannot add them to already existing keys you can use `\cdefinekeys` bzw. `\cdefinesinglekey` to define new keys.

`\cdefinekeys` takes the  $\{⟨unit-key-1⟩\}$  as a “basis”, defines a key with the name  $⟨unit-key-1⟩$  and adds the values  $⟨unit-key-1⟩$ ,  $⟨unit-key-2⟩$ ,  $⟨unit-key-3⟩$ , etc. Furthermore this command also defines the keys  $⟨unit-key-2⟩$ ,  $⟨unit-key-3⟩$ , etc. with the same values as  $⟨unit-key-1⟩$ . Please note that  $⟨...⟩$  has to be a number.

Sometimes it is not that easy and the conversion of one unit into another needs are more complicated formula (see for example temperatures). If that is the case use `\cdefinesinglekey`. As the name says it defines *only* the key  $⟨unit-key-1⟩$  with the values  $⟨unit-key-1⟩$ ,  $⟨unit-key-2⟩$ , etc. The advantage of this command is that now  $⟨...⟩$  can be a formula and the numerical input can be placed explicitly using **#1**.

**Example:** This example defines following keys with their respective value:

- the key **kg** with the values **kg**, **dag**, **g** and **oz**
- the key **dag** with the values **kg**, **dag**, **g** and **oz**
- the key **g** with the values **kg**, **dag**, **g** and **oz**
- the key **oz** with the values **kg**, **dag**, **g** and **oz**
- the key **d** with the values **d**, **h**, **min** and **s**
- ...

1 kg = 1 kg	1 kg = 100 dag	1 kg = 1000 g
1 kg = 35.273 99 oz	1 kg = 2.204 622 6 lb	

```
\cdefinekeys {kg}
{
  {dag}{ 100 } %% 1 kg are 100 dag
  {g} { 1000 } %% 1 kg are 1000 g
  {oz} { 35.27399 } %% 1 kg are 35.27399 oz
}
```

```

    {lb} { 2.204 622 6 } %% 1 kg are 2.204 622 6 lb
  }
\cdefinekeys {d}
{
  {h} { 24 } %% 1 day are 24 hours
  {min}{ 1440 } %% 1 day are 1440 minutes
  {s} { 86400 } %% 1 day are 86400 seconds
}

```

To convert degree Fahrenheit to degree Celsius, kelvin and degree Réaumur one needs the formulas

$$T_C = (T_F - 32) \cdot \frac{5}{9}$$

$$T_K = (T_F - 459.67) \cdot \frac{5}{9}$$

$$T_{Re} = (T_F - 32) \cdot \frac{4}{9}$$

with  $T_F$  being the input temperature in degree Fahrenheit and  $T_C$  being the same temperature in degree Celsius, etc. Using `\cdefinesinglekey` the key **F** and the values **C**, **K** and **Re** are defined:

```

\cdefinesinglekey {F}
{
  {C} { ( #1 - 32 ) * 5/9 } %% see formulas above
  {K} { ( #1 + 459.67 ) * 5/9 }
  {Re} { ( #1 - 32 ) * 4/9 }
}

```

This defines the key **F** with the values **F**, **C**, **K** and **Re**.

---

```

\cuaddkeys      \cuaddkeys{<unit-key-1>}
\cuaddsinglekeys {
    {<unit-key-2>} {<1 unit-key-1 are ... unit-key-2>}
    {<unit-key-3>} {<1 unit-key-1 are ... unit-key-3>}
    {<unit-key-4>} {<1 unit-key-1 are ... unit-key-4>}
    ...
}
\cuaddsinglekeys{<unit-key-1>}
{
    {<unit-key-2>} {<1 unit-key-2 are ... unit-key-1>}
    {<unit-key-3>} {<1 unit-key-3 are ... unit-key-1>}
    ...
}

```

---

These commands add  $\langle unit-key-2 \rangle$ , etc. to the already defined key  $\langle unit-key-1 \rangle$ .

`\cuaddkeys` takes the already defined key  $\{\langle unit-key-1 \rangle\}$  as a “basis”, and adds  $\langle unit-key-2 \rangle$ ,  $\langle unit-key-3 \rangle$ , etc. to its values. Furthermore it adds those new values to other keys linked to  $\langle unit-key-1 \rangle$  and defines the new keys  $\langle unit-key-2 \rangle$ , etc. with the same values as  $\langle unit-key-1 \rangle$ .

If the conversion is more complicated use `\cuaddsinglekeys`. It adds  $\langle unit-key-2 \rangle$ , etc. as values to  $\langle unit-key-1 \rangle$ . The numerical input can be placed using `#1` (see `\cundefinesinglekey`). This command neither defines new keys nor does it add values to other keys than  $\langle unit-key-1 \rangle$ .

**Example:** Suppose you are british (I am sorry, I can’t think of another reason to use those units) and you want to implement ‘stone’ (yes, I was surprised myself that such a unit exists, but it even appears in a Sherlock-Holmes story). You exactly know that 1 st equals 14 lb, well ... now you have two choices. `\cuaddkeys` or `\cuaddtokeys` (use the one best fitting). This example uses the first, the next the latter one.

```

\newcookingunit {st} %% defining new unit 'stone'
\cuaddkeys {lb} %% adding st to lb (could also add to kg, dag and oz)
{
    {st} { 1/14 } %% 1 lb are 1/14 st as 14 lb are 1 st
}

```

0.07 st	<code>\cunum[lb=st]{1}{lb}\\</code>
14 lb	<code>\cunum[st=lb]{1}{st}\\</code>
6350.29 g	<code>\cunum[st=g]{1}{st}\\</code>
6.35 kg	<code>\cunum[st=kg]{1}{st}\\</code>
0.16 st	<code>\cunum[kg=st]{1}{kg}\\</code>

**Example:** Now you want to add degree Rømer and convert Celsius to degree Rømer:

$$T_{R\varnothing} = T_C * \frac{21}{40} + 7.5$$



```

%% defining new unit 'degree R{\o}mer'
\newcookingunit [\ensuremath{ {}^{\circ} } \kern-\scriptspace R{\o} ] {Ro}
\cuaddsinglekeys {C} %% adds value 'Ro' to 'C'.
{
  {Ro} { #1 * 21/40 + 7.5 }
}
\cusetup %% round to integer automatically
{
  set-option-for-Ro = { round-to-int = true }
}

10°C \cunum{10}{C}\
13°Rø \cunum[C=Ro]{10}{C}\

```

---

`\cuaddtokeys` `\cuaddtokeys {\langle unit-key-1 \rangle} {\langle unit-key-2 \rangle} {\langle 1 unit-key-2 are ... unit-key-1 \rangle}`

---

Works similar to `\cuaddkeys` regarding the definition of keys.

**Example:** Continuing the example from before, this time with `\cuaddtokeys`:

```

\newcookingunit {st} %% defining (again) new unit 'stone'
\cuaddtokeys {lb} {st} { 14 } %% 1 st are 14 lb

0.07st \cunum[lb=st]{1}{lb}\
14lb \cunum[st=lb]{1}{st}\
6350.29g \cunum[st=g]{1}{st}\
6.35kg \cunum[st=kg]{1}{st}\
0.16st \cunum[kg=st]{1}{kg}\

```

## 7 Language support

The unit names and symbols depend on the language. To change the name depending on the language you can use `\cudefinename` and to only change symbols use `\cudefinesymbol`.

---

`decimal-mark`  
`one(m)`  
`one(f)`  
`one(n)`

---

Those are special keys (as they cannot be used as units). Not only are printed units language depending, but as is the decimal mark (“.” or “,”). To set the decimal mark use `decimal-mark` (see examples below).

Furthermore if you are using the package-option `use-numerals` you may also use the keys `one(m)`, `one(f)` and `one(n)`. If you use this option, integers below a certain value (see option `use-numerals-below`) are written-out. The only problem is the written-out “1” mostly depends on the gender of the following word (e.g. “ein Baum” (m), “eine Pflanze” (f) and “ein Auto” (n)). To set the written-out 1 to be correct with the gender of the used unit, use this key (see also examples below)

---



---

`\cudefinename`    `\cudefinename{⟨Language⟩}`

```
{
  {⟨unit-key-1⟩} [⟨symbol-1⟩] {⟨singular-1⟩} [⟨plural-1⟩] ⟨gender⟩
  {⟨unit-key-2⟩} [⟨symbol-2⟩] {⟨singular-2⟩} [⟨plural-2⟩] ⟨gender⟩
  ...
}
```

This command defines the names (and optionally the symbol) of the commands printed in `\cutext` and `\Cutext` (and `\cunum` regarding the symbol) for the specific `⟨Language⟩`. For details regarding `⟨language⟩` see the `translator`-documentation.

If the plural form of the name differs from the singular form use `[⟨plural⟩]` to specify the plural form, if no `[⟨plural⟩]` is given the plural will be set equal to its singular. The singular is only used if the number in `\cutext` and `\Cutext` is equal to 1.

`⟨gender⟩` can be `m` (maskulin), `f` (feminin) or `n` (neutrum). If not given `m` is used as default.

```
\cudefinename {English}
{
  {kg}   {kilogramme}
  {oz}   {ounce}
  {h}    {hour} [hours]
  {C}    {degree \space Celsius} [degrees \space Celsius]
  {decimal-marker} {\.}
  {one(m)} {one}
  {one(f)} {one}
  {one(n)} {one}
}
```

```
\cudefinename {German}
{
  {kg}   {Kilogramm} <n>
  {oz}   {Unze} <f>
  {d}    {Tag} [Tage]
  {h}    {Stunde} [Stunden] <f>
  {C}    {Grad\space Celsius}
  {decimal-marker} {,}
  {one(m)} {ein}
  {one(f)} {eine}
  {one(n)} {ein}
}
```

---

```
\cudefinesymbol \cudefinesymbol{<Language>}
{
  {<unit-key-1>} {<symbol-1>}
  {<unit-key-2>} {<symbol-2>}
  ...
}
```

---

This command defines the symbols of the units printed in `\cunum` for the specific *<language>*. It works similar as `\cudefinename`, but only the symbols (and no names) can be set. For details regarding *<language>* see the `translator`-documentation.

```
\cudefinesymbol {English}
{
  {decimal-mark} {.,}
  {one(m)} {one}
  {one(f)} {one}
  {one(n)} {one}
}
\cudefinesymbol {German}
{
  {decimal-mark} {,,}
  {one(m)} {ein}
  {one(f)} {eine}
  {one(n)} {ein}
}
\cudefinesymbol {French}
{
  {l} {L}
  {dl} {dL}
  {cl} {cL}
  {ml} {mL}
  {decimal-mark} {.,}
  {one(m)} {un}
  {one(f)} {une}
  {one(n)} {un}
}
```

## 8 Options

Options in `cooking-units` can mostly be set globally using `\cusetup` or locally using the optional argument of the respective command (but *not* as a package option). The only exception is the option given in [8.1 on the following page](#) which needs to be used as a package option.

## 8.1 Load time options

---

<code>use-numerals</code>	<code>\usepackage[use-numerals=&lt;true/false&gt;]{cooking-units}</code>
---------------------------	--

---

If set to `true` loads package `fmtcount` and uses `\numberstringnum` for `\cutext` and `\Numberstringnum` for `\Cutext` to write-out numbers below `use-numerals-below` (13 by default), integers above are printed as numbers. Please note the keys `one(m)`, `one(f)` and `one(n)` to change the printed “one” (as “one” is in many languages dependent on the gender of the following word. E.g in German: Maskulin: ein Baum, Feminin: eine Pflanze, Neutrum: ein Auto).

one kilogramme	<code>\cutext{1}{kg}\</code>
One kilogramme	<code>\Cutext{1}{kg}\</code>
two kilogramme	<code>\cutext{2}{kg}\</code>
Two kilogramme	<code>\Cutext{2}{kg}\</code>
13 kilogramme	<code>\cutext{13}{kg}\</code>
14 kilogramme	<code>\Cutext{14}{kg}</code>

## 8.2 Normal options

This option can only be set as local options or using `\cusetup`, but *not* as load time options.

---

<code>\cusetup</code>	Options can be set globally using <code>\cusetup</code> .
-----------------------	---

---



---

<code>unit</code>	<code>&lt;unit-key-1&gt; = &lt;unit-key-2&gt;</code>
-------------------	--

---

Convert units from `<unit-key-1>` to `<unit-key-2>` (see 6 on page 4 to define new options).

---

<code>set-option-for-&lt;unit-key&gt;</code>	<code>set-option-for-&lt;unit-key&gt; = &lt;key1=value1,...&gt;</code>
<code>add-option-for-&lt;unit-key&gt;</code>	<code>add-option-for-&lt;unit-key&gt; = &lt;key1=value1,...&gt;</code>
<code>erase-all-options</code>	<code>erase-all-options</code>

---

Sets and adds `<key1=value1,...>`, for a specific `<unit-key>` `erase-all-options` is used to erase all options for all `<unit-key>`s.

You may want to attach some options to a special `<unit-key>`. Those options are automatically activated if (and only if) the specific `<unit-key>` is used (or changed into this unit). Setting options overwrites old options. Adding options, well ... adds the options to the old ones.

The following rounds the values to integers for F, C, K and Re.

```
\cusetup
{
  set-option-for-F   = { round-to-int = true } ,
  set-option-for-C   = { round-to-int = true } ,
  set-option-for-K   = { round-to-int = true } ,
  set-option-for-Re  = { round-to-int = true }
}
```

You can “delete” the options by setting an empty value for a specific  $\langle unit-key \rangle$  (or use `erase-all-options` to erase all options for all  $\langle unit-key \rangle$ s)

---

**eval-fraction**    `eval-fraction =  $\langle true/false \rangle$`

---

This option takes `true` or `false` as values. If set to `true` fractions are evaluated. Please note that divisions through zero are not allowed.

	<code>\cusetup{eval-fraction=true}</code>
0.33 kg	<code>\cunum{1/3}{kg}\\</code>
0.5 kg	<code>\cunum{1/2}{kg}\\</code>
500 g	<code>\cunum[kg=g]{1/2}{kg}\\</code>
1.5 kg	<code>\cunum{1_1/2}{kg}\\</code>
1500 g	<code>\cunum[kg=g]{1_1/2}{kg}</code>

---

**round-precision**    `round-precision =  $\langle integer \rangle$`

---

Rounds the amount automatically to  $\langle integer \rangle$  digits after the colon. Note that units like C, F, K and Re are still rounded to integers due to `set-option-for- $\langle unit-key \rangle$` .

	<code>\cusetup{round-precision=5}</code>
1.23457 kg	<code>\cunum{1.23456789}{kg}\\</code>
0.01259 kg	<code>\cunum[g=kg]{12.587}{g}\\</code>
194 kg	<code>\cunum{194}{kg}\\</code>
392–410 °F	<code>\cunum[C=F]{200--210}{C}\\</code>
–273 °C	<code>\cunum[K=C]{0.0012}{K}\\</code>
	<code>\cusetup{round-precision=1}</code>
1.2 kg	<code>\cunum{1.23456789}{kg}\\</code>
12.6 kg	<code>\cunum{12.58}{kg}\\</code>
0.2 kg	<code>\cunum[g=kg]{194}{g}\\</code>
392–410 °F	<code>\cunum[C=F]{200--210}{C}\\</code>
–273 °C	<code>\cunum[K=C]{0.0012}{K}</code>

---

**round-to-int**    `round-to-int =  $\langle true/false \rangle$`

---

Rounds the amount to an integer if set `true`.

	<code>\cusetup{round-to-int=true}</code>
1 kg	<code>\cunum{1.23456789}{kg}\\</code>
13 kg	<code>\cunum{12.58}{kg}\\</code>
0–0 kg	<code>\cunum[g=kg]{194--294}{g}\\</code>
1235 g	<code>\cunum[kg=g]{1.23456789}{kg}\\</code>

---

<code>range-sign</code>	<code>range-sign = <math>\langle string \rangle</math></code>
	<code>cunum-range-sign = <math>\langle string \rangle</math></code>
	<code>cutext-range-sign = <math>\langle string \rangle</math></code>

---

The second sets the *printed* range-sign used in `\cunum` to  $\langle string \rangle$ , the third sets the printed range-sign used in `\cutext`/`\Cutext` to  $\langle string \rangle$ .

Use the `range-sign` to set the printed range-signs for both `\cunum` and `\cutext`/`\Cutext` to  $\langle string \rangle$ .

The default for  $\langle string \rangle$  is `--` (for both).

	<code>\cusetup{cunum-range-sign={~to~}}</code>
1 to 2 kg	<code>\cunum{1--2}{kg}\</code>
1-2 kilogramme	<code>\cutext{1--2}{kg}\</code>
1-2 kilogramme	<code>\Cutext{1--2}{kg}\</code>
	 <code>\cusetup{cutext-range-sign={~to~}}</code>
1-2 kg	<code>\cunum{1--2}{kg}\</code>
1 to 2 kilogramme	<code>\cutext{1--2}{kg}\</code>
1 to 2 kilogramme	<code>\Cutext{1--2}{kg}\</code>
	 <code>\cusetup{range-sign={~to~}}</code>
1 to 2 kg	<code>\cunum{1--2}{kg}\</code>
1 to 2 kilogramme	<code>\cutext{1--2}{kg}\</code>
1 to 2 kilogramme	<code>\Cutext{1--2}{kg}\</code>

---

<code>fraction-command</code>	<code>fraction-command = <math>\langle \backslash command \rangle</math></code>
-------------------------------	---

---

Sets the command used for printing fractions equal to  $\langle \backslash command \rangle$ .  $\langle \backslash command \rangle$  has to take two arguments. By default it is equal to `\sfrac` from `xfrac`.

Please note that the amount is *not* printed inside a math environment by default.

	<code>\newcommand\myfrac[2]{\#1/\#2}</code>
	<code>\cusetup{fraction-command=\myfrac}</code>
1/2 kg	<code>\cunum{1/2}{kg}\</code>
4/5 °C	<code>\cunum{4/5}{C}\</code>
12/3 kg	<code>\cunum{12/3}{kg}\</code>
	<code>\cusetup{fraction-command=\nicefrac}</code>
1/2 kg	<code>\cunum{1/2}{kg}\</code>
4/5 °C	<code>\cunum{4/5}{C}\</code>
12/3 kg	<code>\cunum{12/3}{kg}</code>

---

<code>fraction-inline</code>	<code>fraction-inline = <math>\langle input \text{ containing } \#1 \text{ and } \#2 \rangle</math></code>
------------------------------	--

---

Similar to `fraction-command` only that you don't have to define a command to alter the output of the fraction.

$\frac{1}{2}$ kg	<code>\csetup{fraction-inline={#1/#2}}</code>
$\frac{4}{5}$ °C	<code>\cunum{1/2}{kg}\\</code>
$1\frac{2}{3}$ kg	<code>\cunum{4/5}{C}\\</code>
	<code>\cunum{1_2/3}{kg}\\</code>
$\frac{2}{1}$ kg	<code>\csetup{fraction-inline={\nicefrac{#2}{#1}}}</code>
$\frac{5}{4}$ °C	<code>\cunum{1/2}{kg}\\</code>
$1\frac{3}{2}$ kg	<code>\cunum{4/5}{C}\\</code>
	<code>\cunum{1_2/3}{kg}</code>

---

**mixed-fraction-space**


---

`mixed-fraction-space =  $\langle length \rangle$`

Sets the length between the fraction and the number in a mixed-fraction, default is 0.1em.

$1\frac{2}{3}$ kg	<code>\cunum{1_2/3}{kg}\\</code>
$10\frac{2}{3}$ kg	<code>\cunum{10_2/3}{kg}\\</code>
	<code>\csetup{mixed-fraction-space=1em}</code>
$1\frac{2}{3}$ kg	<code>\cunum{1_2/3}{kg}\\</code>
$10\frac{2}{3}$ kg	<code>\cunum{10_2/3}{kg}\\</code>
	<code>\csetup{mixed-fraction-space=0em}</code>
$1\frac{2}{3}$ kg	<code>\cunum{1_2/3}{kg}\\</code>
$10\frac{2}{3}$ kg	<code>\cunum{10_2/3}{kg}\\</code>

---

**set-special-sign**  
**add-special-sign**


---

`set-special-sign =  $\langle character(s) \rangle$`   
`add-special-sign =  $\langle character(s) \rangle$`

Allows  $\langle character(s) \rangle$  to be used in the first mandatory argument of `\cunum` without raising an error (you can customize this behavior, see `set-unknown-message`). By default it is set to ?.

? kg	<code>\cunum{?}{kg}\\</code>
10?–20? kg	<code>\cunum[g=kg]{10?--20?}{kg}\\</code>
	<code>\csetup{add-special-sign={xX}}</code>
x kg	<code>\cunum{x}{kg}\\</code>
X–? kg	<code>\cunum{X--?}{kg}\\</code>
	<code>\csetup{set-special-sign={}}</code>
1 kg	<code>\cunum{1}{kg}\\</code>
1–2 kg	<code>\cunum{1--2}{kg}\\</code>

---

**parse-number**


---

`parse-number =  $\langle true/false \rangle$`

If set to **false** prints the number of `\cunum`, `\cutext` and `\Cutext` as they are (after some ... well ... parsing due to “\_”). It is **true** by default.

1 kg	<code>\cusetup{parse-number=false}</code>
1-2 kg	<code>\cunum[kg=g]{1}{kg}\\</code>
1-----2 kg	<code>\cunum{1--2}{kg}\\</code>
1.2 kg	<code>\cunum{1-----2}{kg}\\</code>
1,2 kg	<code>\cunum{1.2}{kg}\\</code>
1/2 kg	<code>\cunum[kg=g]{1,2}{kg}\\</code>
1_2/3 kg	<code>\cunum{1/2}{kg}\\</code>
1/2_3 kg	<code>\cunum{1_2/3}{kg}\\</code>
qwertzuiop kg	<code>\cunum{1/2_3}{kg}\\</code>
1 kilogramme	<code>\cunum{qwertzuiop}{kg}\\</code>
100 kilogramme	<code>\cutext{1}{kg}\\</code>
gjfak kilogramme	<code>\cutext{100}{kg}\\</code>
	<code>\cutext{gjfak}{kg}\\</code>

---

**use-numerals-below**    `use-numerals-below =  $\langle integer \rangle$`

---

Only usable if the package option `use-numerals` is active. Prints the name of the numbers for integers used in `\cutext` and `\Cutext` smaller than  $\langle integer \rangle$ .  $\langle integer \rangle$  it is by default 13.

one kilogramme	<code>\cutext{1}{kg}\\</code>
two kilogramme	<code>\cutext{2}{kg}\\</code>
twelve kilogramme	<code>\cutext{12}{kg}\\</code>
13 kilogramme	<code>\cutext{13}{kg}\\</code>
	<code>\cusetup{use-numerals-below=10}</code>
one kilogramme	<code>\cutext{1}{kg}\\</code>
two kilogramme	<code>\cutext{2}{kg}\\</code>
12 kilogramme	<code>\cutext{12}{kg}\\</code>
13 kilogramme	<code>\cutext{13}{kg}\\</code>
	<code>\cusetup{use-numerals-below=0}</code>
1 kilogramme	<code>\cutext{1}{kg}\\</code>
2 kilogramme	<code>\cutext{2}{kg}\\</code>
12 kilogramme	<code>\cutext{12}{kg}\\</code>
13 kilogramme	<code>\cutext{13}{kg}\\</code>

---

**set-unknown-message**    `set-unknown-message =  $\langle error/warning/none \rangle$`

---

Using a special sign (? by default) causes a warning to be raised. Set this option to **error** if you want an error (as an extra emphasis), **warning** if you want a warning (default) and **none** if you don't want to know anything about it.



### 8.3 Weird options

---

<code>check-temperature</code>	<code>check-temperature = <math>\langle true/false \rangle</math></code>
--------------------------------	--

---

Checks if the used temperature is below the absolute zero point. Currently C, F, K and Re are supported. While `\cunum{0}{K}` is ok, `\cunum{-1}{K}` raises an error, same for the others. Is set to `false` by default.

---

<code>convert-to-eV</code>	<code>convert-to-eV = <math>\langle true/false \rangle</math></code>
----------------------------	--

---

Converts (nearly) every unit in [table 2 on page 5](#) to electron volt or the respective derivative. Note that this option is: a) experimental and probably will forever be and b) just a joke, you are not supposed to use this units in a cookery book.

	<code>\cusetup{convert-to-eV=true}</code>
$5609589206809303 \text{ eV}/c^2$	<code>\cunum{1}{kg}\\</code>
$130148943000000000 c^3 \hbar^3 / \text{eV}^3$	<code>\cunum{1}{1}\\</code>
$6241509125883258000 \text{ eV}$	<code>\cunum{1}{J}\\</code>
$5067730.93 \text{ ch}/\text{eV}$	<code>\cunum{1}{m}\\</code>
$0.02 \text{ eV}$	<code>\cunum{1}{C}\\</code>
$1519267460691082 \text{ h}/\text{eV}$	<code>\cunum{1}{s}\\</code>

## 9 Bugs & Feedback

Bug reports are always welcome. If you are sending a bug report please include a minimal working example showing the bug and a short description. Furthermore please add “cooking-units” to the e-mail header. GMX has the habit of putting e-mails into the spam account and adding “cooking-units” to the header makes it easier to recognize those e-mails.

Feedback and requests are most welcome. Please also add (if possible) an example of the desired output into the minimal example (and also add “cooking-units” to the header).

Furthermore, as you can see I am not able to speak too many languages (german and english to be precise; I managed to add french with the help of the internet, which is not optimal) so if you are able to speak a language not yet implemented and would like to help you can send me a list of the translations of the units given in [table 2 on page 5](#). I would need

- their singular (and plural) form,
- the gender,
- the printed symbol (if different),
- `decimal-mark` and `one(m)`, `one(f)`, `one(n)`

Oh yeah, if someone has a better idea of how to deal with languages I am happy to know.