The cooking-units package*

Ben Vitecek
b.vitecek@gmx.at

October 17, 2017

Abstract

This package enables user to globally format units, to switch between them and since v1.10 you can also change your recipes for a given number of persons. It should be used for light-hearted things like cookery books (and not e.g. scientific texts).[1] Please read through the section "Important Changes"

Contents

*This document corresponds to Benedikt Vitecek v1.11, dated 2017/03/10.
[1]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, or older: dkg). 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 LaTeX and changing those units by hand seemed not very LaTeXlike, so I started writing some code to convert units. I expanded the code, rewrote it in LaTeX3 (which is much more pleasant than LaTeX 2_ε) and here it is.

1.1 Important Changes

Language I am now using the translations package and I hope it makes things easier. As such, declaring the used language through class-options shouldn't be necessary anymore.

Phrases This package now supports the usage of "phrases" (words used instead of certain integers) (which I think are called "counting measures" in english, but I am not sure).

`\cutext` and `\Cutext` If no translation is found for a specific language, `\cutext` and `\Cutext` are replaced by `\cunum` with a warning is given.

Commands Currently, it seems that alowing ⟨*label*⟩ to be set by arrow-brackets was not the best idea as it leads to problems if they are made active (e.g. babel and option `spanish`). As such, `<` is not allowed as a "special-sign" anymore as this package tries to "fix" this (at least make it work). If any problems occur (for this specific case or in general) please feel free to contact me.

1.2 Supported languages

- German
- English
- French (currently suboptimal[2])

Have another language to add or a correction of an existing one? See section 10 for more details. Wanna just check the existing translations? See appendix A.

2 The Commands

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

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

[2] You can only get limited information from the internet.

- \cuam⟨*label*⟩[⟨*options*⟩] {⟨*amount*⟩}
- \cusetup{⟨*options*⟩}

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*⟩[3]. 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[4]. [⟨*space*⟩] adds a space between the number and the unit using \phantom.

For a list of predefined units have a look at table 1.

⟨*label*⟩ is explained in section 3.

1 kg	\cunum{1}{kg}\\
2.3 kg	\cunum{2.3}{kg}\\
2.3 kg	\cunum{2,3}{kg}\\
2–3 kg	\cunum{2--3}{kg}\\
2.5–3.5 kg	\cunum{2.5--3.5}{kg}\\
2500–3500 g	\cunum[kg=g]{2.5--3.5}{kg}\\
392 °F	\cunum[C=F]{200}{C}\\
356–392 °F	\cunum[C=F]{180--200}{C}\\
½ m	\cunum{1/2}{m}\\
1 ½ m	\cunum{1_1/2}{m}\\
1 ½ m	\cunum[m=cm]{1_1/2}{m}\\
? ℓ	\cunum{?}{l}\\
50 dag	\cunum{50}{dag}\\
5 dag	\cunum{5}[0]{dag}\\
1.12 m	\cunum{1.1234}{m}

Decimal numbers are automatically rounded to 2 digits after the colon, temperatures (C, F, K and Re) are automatically rounded to integers.[5]

\cutext and \Cutext print the number and the written name of the unit. Since v1.10 it works similar[6] to \cunum: it allows the conversion between units and interprets the numerical part (again _ and / are used for (mixed) fractions and -- for ranges). Furthermore, if the package option use-numerals is used, integers below a specific integer (by default 13; see use-numerals-below) are written out with \Cutext capitalizing the first letter (using package fmtcount).

1 litre	\cutext{1}{l}\\
1 litre	\Cutext{1}{l}\\
1–2 litres	\Cutext{1--2}{l}\\
12 litres	\cutext{12}{l}\\
13 litres	\Cutext{13}{l}

and using package option use-numerals=true

one litre	\cutext{1}{l}\\
One litre	\Cutext{1}{l}\\
one–two litres	\cutext{1--2}{l}\\
twelve litres	\cutext{12}{l}\\
13 litres	\Cutext{13}{l}

[3] New keys can be added and defined, see section 5 and section 6 for further information.
[4] You can customize this behavior, see section 9
[5] You can – of course – change this behavior, see section 9.
[6] One could also say “exactly like”.

Furthermore, since v1.10 `\cutext` and `\Cutext` also allows their units to be changed (this behavior can be altered using `cutext-change-unit`):

```
                                \cusetup{l=ml}
1000 millilitres                \cutext{1}{l}\\
1000 millilitres                \Cutext{1}{l}\\
1000–2000 millilitres           \cutext{1--2}{l}\\
12000 millilitres               \cutext{12}{l}\\
13000 millilitres               \Cutext{13}{l}\\
? litres                        \Cutext{?}{l}\\
½ litres                        \Cutext{1/2}{l}\\
```

`\cuam` works like `\cunum`, but without a unit, so changing units doesn't affect it. Like `\cunum` `_` and `/` are used to imply a (mixed) fraction and `--` is used for ranges.

```
3                               \cuam{3}\\
2–3                             \cuam{2--3}\\
⅔                               \cuam{2/3}\\
1⅔                              \cuam{1_2/3}
```

Furthermore it allows the concept of "phrases" (replacing a positive integer by a word, such as "12" becoming "dozen"[7]) which can be activated by the option `use-phrases` (as I don't know any english phrases, I switched the language to german for the following examples)

```
                                \cusetup{use-phrases=true}
1 Dutzend                       \cuam{12}\\
13                              \cuam{13}\\
2 Dutzend                       \cuam{24}\\
1–2 Dutzend                     \cuam{12--24}\\
12–13                           \cuam{12--13}\\
18                              \cuam{18}\\
5 Dutzend                       \cuam{60}
```

3 Label & refs: Changing the amount of the recipe

What if you don't want to change units, but the amounts of the recipe because you cook not for 4 persons, but for 2 and don't like to do the math? Simple, use the following commands:

- `\culabel{`⟨*label*⟩`}{`⟨*number of persons*⟩`}`
- `\curef{`⟨*label*⟩`}`

The first one is the important one: It defines a ⟨*label*⟩ for a recipe which is initially for ⟨*number of persons*⟩. Afterwards ⟨*label*⟩ can be used to tell the commands from section 2 that the given amounts are for ⟨*number of persons*⟩. Each ⟨*label*⟩ must be unique and an error is raised if a ⟨*label*⟩ is already defined.

If you would like to print the number of persons this recipe is for, use `\curef`, which is fully expandable.

The following example uses `\culabel` to specify that the recipe is initially intended for 2 persons:

[7] At least I think

recipe for 2 persons:
10–20 dag flour,
½ ℓ water,
10 gramme nuts,
2–3 eggs,
180 °C (356 °F) open fire

```
\culabel{recipe}{2}
recipe for \curef{recipe} persons:\\
\cunum<recipe>{10--20}{dag} flour,\\
\cunum<recipe>{1/2}{l} water,\\
\cutext[ref=recipe]{10}{g} nuts,\\
\cuam<recipe>{2--3} eggs,\\
\cunum{180}{C} (\cunum[C=F]{180}{C})
open fire
```

Now with combination of the option `set-number-of-persons` and setting `recalculate-amount` to `true` you can have this recipe changed to four persons:

```
\culabel{recipe}{2}
%% adding options:
\cusetup{set-number-of-persons=4,recalculate-amount=true}
```

recipe for 4 persons:
20–40 dag flour,
1 ℓ water,
20 gramme nuts,
4–6 eggs,
180 °C (356 °F) open fire

```
recipe for \curef{recipe} persons:\\
\cunum<recipe>{10--20}{dag} flour,\\
\cunum<recipe>{1/2}{l} water,\\
\cutext[ref=recipe]{10}{g} nuts,\\
\cuam<recipe>{2--3} eggs,\\
\cunum{180}{C}
(\cunum[C=F]{180}{C}) open fire
```

Note that fractions are automatically evaluated and that only values with a ⟨*label*⟩ are changed (`\cunum{180}{C}` for example stays the same which also makes sense as the heat should be the same).

4 Some Interesting options

This package has some options which might be of interest and to highlight them, this section exists. All options can be found in section 9.

`use-numerals`
`use-numerals-below`
`print-numerals`

As seen above, you can use the *package*-option `use-numerals` to print integers used by `\cutext` and `\Cutext` below `use-numerals-below` (13 by default) by fmtcount. You can still decide if numerals should be printed or not with `print-numerals`.

Note: `use-numerals` is a package option as it needs to load fmtcount which is not loaded by default.

`use-phrases`

In (I presume) all languages there exist phrases for a given amount or a number of things (think it is called “counting measurement”). In German you may say instead of “12”: “ein Dutzend”. Using this option you can tell this package to replace predefined integers used in `\cuam` by phrases for given language (to define new ones, see section 8.1)

Using (for example) langauge `ngerman` (or `naustrian`, etc.) with package option `use-phrases=true` gives:

```
                                \cusetup{use-phrases=true}
1 Dutzend                       \cuam{12}\\
2 Dutzend                       \cuam{24}\\
1–2 Dutzend                     \cuam{12--24}\\
12–13                           \cuam{12--13}\\
18                              \cuam{18}\\
5 Dutzend                       \cuam{60}
```

This of course also works with the *package*-option `use-numerals`:

```
                                \cusetup{use-phrases=true}
ein Dutzend                     \cuam{12}\\
zwei Dutzend                    \cuam{24}\\
ein–zwei Dutzend                \cuam{12--24}\\
12–13                           \cuam{12--13}\\
18                              \cuam{18}\\
fünf Dutzend                    \cuam{60}
```

Note: Curently only the lower-case variant for `use-numerals` is supported. Furthermore this feature is only available for `\cuam`.

5 Predefined units & some notes

In table 1 and table 2 (and table 3) you can find all predefined units. In appendix A all translations available are listed.

6 Defining units

New units can be defined using `\declarecookingunit`, `\newcookingunit` and `\providecookingunit`:

`\declarecookingunit` `\declarecookingunit[⟨symbol⟩]{⟨unit-key⟩}`
`\newcookingunit` `\newcookingunit[⟨symbol⟩]{⟨new-unit-key⟩}`
`\providecookingunit` `\providecookingunit[⟨symbol⟩]{⟨new-unit-key⟩}`

These commands define the unit ⟨*unit-key*⟩. If the key is not the same as the printed symbol use `[⟨symbol⟩]`. Note that ⟨*unit-key*⟩ should neither contain `/` nor `,`.

`\newcookingunit` raises an error if the unit is already defined, `\declarecookingunit` creates or (if given) overwrites ⟨*symbol*⟩ and `\providecookingunit` does nothing if the unit is already defined.

Some examples:

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

Note: The definition of the printed degree Celsius is directly copied and pasted from (a maybe older version of) siunitx

Table 1: The first column shows a list of predefined unit-keys. The column "default-symbol" shows the abbreviation used if for given language no translation is defined. The third column "unitname" *is* language dependent and shows the name printed while using `\cutext` and `\Cutext`. Note that "electron volt" exists just for fun.

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

Table 2: A (not only) spoonful of (more or less) country and language dependent units. Please note that sometimes a translation is nearly impossible as a unit (e.g. "saltspoonful") may not exist in another language (like german; at least I never heard of it). So please only use units known to you.

unitname	unit-key	default symbol
pn	pinch	pinch
EL	EL	tablespoon
TL	TL	teaspoon
dsp	dsp.	dessertspoonful
csp	csp.	coffeespoonful
ssp	ssp.	saltspoonful
Msp	Msp.	Messerspitze

Table 3: List of nonsense units (exist just for fun, there will be no support for those units).

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

7 Defining options to change units

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

- \cudefinekeys
- \cudefinesinglekey
- \cuaddkeys
- \cuaddsinglekeys
- \cuaddtokeys

I apologize for the (name) inconsistency between \cudefinekeys and \cudefinesinglekey (although they are named similarly they work different).

\cudefinekeys
\cudefinesinglekey

```
\cudefinekeys{⟨unit-key-1⟩}
  {
    {⟨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⟩}
    ...
  }
\cudefinesinglekey{⟨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⟩}
    ...
  }
```

If you define new units (see section 6) and cannot add them to already existing keys you can use `\cudefinekeys` bzw. `\cudefinesinglekey` to define new keys.

`\cudefinekeys` takes `{`⟨*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 `\cudefinesinglekey`. 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	

```
\cudefinekeys {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
  }

\cudefinekeys {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 Réamur one needs the formulas[8]

$$T_C = (T_F - 32) \cdot \frac{5}{9}$$
$$T_K = (T_F - 459.67) \cdot \frac{5}{9}$$
$$T_{Re} = (T_F - 32) \cdot \frac{4}{9}$$

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

```
\cudefinesinglekey {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`
`\cuaddsinglekeys`

```
\cuaddkeys{⟨unit-key-1⟩}
  {
    {⟨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 ⟨*unit-key-2*⟩, etc. to the already defined key ⟨*unit-key-1*⟩.

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

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

[8] See Wikipedia.

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	`\cunum[lb=st]{1}{lb}\\`
14 lb	`\cunum[st=lb]{1}{st}\\`
6350.29 g	`\cunum[st=g]{1}{st}\\`
6.35 kg	`\cunum[st=kg]{1}{st}\\`
0.16 st	`\cunum[kg=st]{1}{kg}\\`
101.6 kg	`\cunum[st=kg]{16}{st}`

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

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

```
%% 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 {⟨unit-key-1⟩} {⟨unit-key-2⟩} {⟨1 unit-key-2 are ... unit-key-1⟩}`

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
```

0.07 st	`\cunum[lb=st]{1}{lb}\\`
14 lb	`\cunum[st=lb]{1}{st}\\`
6350.29 g	`\cunum[st=g]{1}{st}\\`
6.35 kg	`\cunum[st=kg]{1}{st}\\`
0.16 st	`\cunum[kg=st]{1}{kg}\\`
101.6 kg	`\cunum[st=kg]{16}{st}`

8 Language support

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 these[9] keys (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 translations 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 translations 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}
  }
```

Example: Imagine that instead of the abbreviation “dag” for “decagramm” you want to use “ducks” (because ... I don’t know). You can easily do this via

```
\cudefinesymbol {English}
  {
    {dag} {ducks}
  }
```

As you can see it may be a bit suboptimal as there is no plural version allowed. You do it anyway and end up with:

12 ducks weed	`\cunum{12}{dag} weed\\`
3 ducks nuts	`\cunum{3}{dag} nuts\\`
10 ducks duckmeat	`\cunum{10}{dag} duckmeat`

8.1 Phrases

Each language has synonyms for certain (integer) numbers. This package supports those phrases and they can be implemented with the following commands and used by `\cuam`:

`\cudefinephrase`

```
\cudefinesymbol{⟨Language⟩}
  {
    {⟨integer-1⟩}   {⟨phrase-1⟩} [⟨phrase-1-plural⟩] ⟨gender-1⟩
    {⟨integer-2⟩} * {⟨phrase-2⟩} [⟨phrase-2-plural⟩] ⟨gender-2⟩
    ...
  }
```

This command pairs for a given `{⟨Language⟩}` the number `{⟨integer-1⟩}` with `{⟨phrase-1⟩}` (plural and gender). The package then checks if the amount given in `\cuam` is either this number or a *multiple* of it.

If the behavior of checking for a multiple is not wanted, you can use the optional star `*` for a given `{⟨integer⟩}`

⟨*gender*⟩ can be `m`, `f` or `n`. It is `m` by default.

Afterwards the numbers are ordered from highest to lowest so that the phrase with the highest number is used (if used at all).

Furthermore, it chooses star (`*`) phrases over non-star phrases.

Note: Numbers with the optional star `*` are stored as negative numbers.

Example: The following example creates some phrases for the language “German”:

```
\cudefinephrase {German}
  {
    { 12 }  {Dutzend} <n> %% implemented by default
    { 60 }  {Schock} <n>
    { 6  }* {halbes\ Dutzend} <n>
  }
```

Let’s just use them (activating the german language):

	`\cusetup{use-phrases=true}`
1 Dutzend	`\cuam{12}\\`
2 Dutzend	`\cuam{24}\\`
1 Schock	`\cuam{60}\\`
2 Schock	`\cuam{120}\\`
1 halbes Dutzend	`\cuam{6}\\`
18	`\cuam{18}`

As you can see, “Schock” (60) is preferred over “Dutzend” (12) as it linked to the higher number. Furhtermore, for 6 the phrase “halbes Dutzend” (half a dozen) is used, but because it is a star version it is *not* used for 18.

9 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 section 9.1 which needs to be used as a package option.

9.1 Load time options

use-numerals `\usepackage[use-numerals=⟨true/false⟩]{cooking-units}`

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. You can decide to not print any numerals by setting `print-numerals` to `false`.

Note: `use-numerals` is a package option as it needs to load fmtcount which is not loaded by default.

Note: 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: Masculine: ein Baum, Feminin: eine Pflanze, Neutrum: ein Auto).

one kilogramme	`\cutext{1}{kg}\\`
One kilogramme	`\Cutext{1}{kg}\\`
two kilogramme	`\cutext{2}{kg}\\`
Two kilogramme	`\Cutext{2}{kg}\\`
twelve kilogramme	`\cutext{12}{kg}\\`
13 kilogramme	`\cutext{13}{kg}\\`
13 kilogramme	`\cutext{13}{kg}\\`
14 kilogramme	`\Cutext{14}{kg}`

9.2 Normal options

Options in this subsection can only be set as local options or using \cusetup, but *not* as load time options.

\cusetup Options can be set using `\cusetup{⟨options⟩}`.

9.2.1 Unit Specific options

unit ⟨*unit-key-1*⟩ = ⟨*unit-key-2*⟩

Change unit ⟨*unit-key-1*⟩ to ⟨*unit-key-2*⟩ (see section 7 to define new options).

set-option-for-<unit-key>
add-option-for-<unit-key>
erase-all-options

`set-option-for-`⟨*unit-key*⟩ `=` ⟨*key1=value1,...*⟩
`add-option-for-`⟨*unit-key*⟩ `=` ⟨*key1=value1,...*⟩
`erase-all-options`

Sets and adds ⟨*key1=value1,...*⟩ to 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 ⟨*unit-key*⟩ (or use `erase-all-options` to erase all options for all ⟨*unit-key*⟩s)

9.2.2 Command behavior

cutext-to-cunum

`cutext-to-cunum` = ⟨*true/false*⟩

Want to get rid of all `\cutext` and `\Cutext`? Set this option to `true` and all `\cutext` and `\Cutext` are changed into `\cunum`.

1 kilogramme	`\cutext{1}{kg}\\`
2 kilogramme	`\Cutext{2}{kg}\\`
½ kilogramme	`\cutext{1/2}{kg}\\`
? kilogramme	`\cutext{?}{kg}\\`
1000–2000 gramme	`\cutext[kg=g]{1--2}{kg}\\`
	`\cusetup{cutext-to-cunum=true}`
1 kg	`\cutext{1}{kg}\\`
2 kg	`\Cutext{2}{kg}\\`
½ kg	`\cutext{1/2}{kg}\\`
? kg	`\cutext{?}{kg}\\`
1000–2000 g	`\cutext[kg=g]{1--2}{kg}`

cutext-change-unit

`cutext-change-unit` = ⟨*true/false*⟩

Set this option to `true` if you do *not* want the units of `\cutext` and `\Cutext` to be changed. Set to `true` by default

1000 gramme	`\cutext[kg=g]{1}{kg}\\`
½ kilogramme	`\cutext[kg=g]{1/2}{kg}\\`
1000–2000 gramme	`\cutext[kg=g]{1--2}{kg}\\`
	`\cusetup{cutext-change-unit=false}`
1 kilogramme	`\cutext[kg=g]{1}{kg}\\`
½ kilogramme	`\cutext[kg=g]{1/2}{kg}\\`
1–2 kilogramme	`\cutext[kg=g]{1--2}{kg}`

`cuam-version` `cutext-version`

`cuam-version` = ⟨*old/new*⟩
`cutext-version` = ⟨*old/new*⟩

Since v1.10 this package also parses and checks the input of `\cutext` and `\Cutext` and `\cuam`. If you want to restore the old behavior, set this option to `old`, but note that then you can neither change the amounts for a given number of persons nor change the unit of `\cutext` and `\Cutext`. Both of them are set to `new` by default.

9.2.3 Input and Outputs

`set-special-sign` `add-special-sign`

`set-special-sign` = ⟨*character(s)*⟩
`add-special-sign` = ⟨*character(s)*⟩

Allows ⟨*character(s)*⟩ to be used in the first mandatory argument of `\cunum`, `\cuam`, `\cutext` and `\Cutext` without raising an error (you can customize this behavior, see `set-unknown-message`). By default it is set to `?`. Please note that the sign `<` is not allowed as a special sign.

? kg	`\cunum{?}{kg}\\`
10?–20? kg	`\cunum[g=kg]{10?--20?}{kg}\\`
	`\cusetup{add-special-sign={xX}}`
x kg	`\cunum{x}{kg}\\`
X–? kg	`\cunum{X--?}{kg}\\`
	`\cusetup{set-special-sign={}}`
1 kg	`\cunum{1}{kg}\\`
1–2 kg	`\cunum{1--2}{kg}`

`set-unknown-message`

`set-unknown-message` = ⟨*error/warning/none*⟩

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.

`set-cutext-translation-message`

`set-cutext-translation-message` = ⟨*error/warning/none*⟩

If a translation for `\cutext` and `\Cutext` is not avalalable the commands are replaced by `\cunum`. Currently – if this is happening – a warning is shown, you may change the bahvior of the message (error, warning or not showing at all) using this option.

`print-numerals`

`print-numerals` = ⟨*true/false*⟩

If the package option `use-numerals` is set to `true` you can deactivate the printing of numerals by setting `print-numerals` to `false` and activate them by setting it to `true`.

Note that this option is automatically set to `true` if `use-numerals` is used.

one kilogramme	`\cutext{1}{kg}\\`
two kilogramme	`\cutext{2}{kg}\\`
twelve kilogramme	`\cutext{12}{kg}\\`
13 kilogramme	`\cutext{13}{kg}\\`
	`\cusetup{print-numerals=false}`
1 kilogramme	`\cutext{1}{kg}\\`
2 kilogramme	`\cutext{2}{kg}\\`
12 kilogramme	`\cutext{12}{kg}\\`
13 kilogramme	`\cutext{13}{kg}\\`

`use-numerals-below` `use-numerals-below =` ⟨*integer*⟩

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 ⟨*integer*⟩. ⟨*integer*⟩ is by default 13. Package fmtcount is used for this purpose.

one kilogramme	`\cutext{1}{kg}\\`
two kilogramme	`\cutext{2}{kg}\\`
twelve kilogramme	`\cutext{12}{kg}\\`
13 kilogramme	`\cutext{13}{kg}\\`
	`\cusetup{use-numerals-below=10}`
one kilogramme	`\cutext{1}{kg}\\`
two kilogramme	`\cutext{2}{kg}\\`
12 kilogramme	`\cutext{12}{kg}\\`
13 kilogramme	`\cutext{13}{kg}\\`
	`\cusetup{use-numerals-below=0}`
1 kilogramme	`\cutext{1}{kg}\\`
2 kilogramme	`\cutext{2}{kg}\\`
12 kilogramme	`\cutext{12}{kg}\\`
13 kilogramme	`\cutext{13}{kg}\\`
	`\cusetup{use-numerals-below=12001}`
one thousand gramme	`\cutext[kg=g]{1}{kg}\\`
two thousand gramme	`\cutext[kg=g]{2}{kg}\\`
twelve thousand gramme	`\cutext[kg=g]{12}{kg}\\`
13000 gramme	`\cutext[kg=g]{13}{kg}\\`

`parse-number` `parse-number =` ⟨*true/false*⟩

If set to `false` prints the number of `\cunum`, `\cutext`, `\Cutext` and `\cuam` as they are (after some ... well ... parsing due to "_"). Is set to `true` by default.

```
                                \cusetup{parse-number=false}
1 kg                            \cunum[kg=g]{1}{kg}\\
1–2 kg                          \cunum{1--2}{kg}\\
1———-2 kg                       \cunum{1----------2}{kg}\\
1.2 kg                          \cunum{1.2}{kg}\\
1,2 kg                          \cunum[kg=g]{1,2}{kg}\\
1/2 kg                          \cunum{1/2}{kg}\\
1_2/3 kg                        \cunum{1_2/3}{kg}\\
1/2_3 kg                        \cunum{1/2_3}{kg}\\
qwertzuiop kg                   \cunum{qwertzuiop}{kg}\\
1 kilogramme                    \cutext{1}{kg}\\
100 kilogramme                  \cutext{100}{kg}\\
gjfak kilogramme                \cutext{gjfak}{kg}\\
12 kilogramme                   \cutext[kg=g]{12}{kg}\\
1———-2                          \cuam{1----------2}\\
1,2                             \cuam{1,2}\\
1_1/2                           \cuam{1_1/2}\\
kwflk                           \cuam{kwflk}\\
```

`range-sign`

`range-sign` = ⟨*string*⟩
`cunum-range-sign` = ⟨*string*⟩
`cutext-range-sign` = ⟨*string*⟩

The second sets the *printed* range sign used in `\cunum` (and `\cuam`) to ⟨*string*⟩, the third sets the printed range sign used in `\cutext` and `\Cutext` to ⟨*string*⟩. Using the first option sets the range signs for both `\cunum` (and `\cuam`) and `\cutext`/`\Cutext` to ⟨*string*⟩.

The default for ⟨*string*⟩ is `--` (for both).

```
                                \cusetup{cunum-range-sign={~to~}}
1 to 2 kg                       \cunum{1--2}{kg}\\
1 to 2                          \cuam{1--2}\\
1–2 kilogramme                  \cutext{1--2}{kg}\\
1–2 kilogramme                  \Cutext{1--2}{kg}

                                \cusetup{cutext-range-sign={~to~}}
1–2 kg                          \cunum{1--2}{kg}\\
1–2                             \cuam{1--2}\\
1 to 2 kilogramme               \cutext{1--2}{kg}\\
1 to 2 kilogramme               \Cutext{1--2}{kg}

                                \cusetup{range-sign={~to~}}
1 to 2 kg                       \cunum{1--2}{kg}\\
1 to 2                          \cuam{1--2}\\
1 to 2 kilogramme               \cutext{1--2}{kg}\\
1 to 2 kilogramme               \Cutext{1--2}{kg}
```

9.2.4 Rounding options

round-precision

`round-precision = ⟨integer⟩`

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

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

round-to-int

`round-to-int = ⟨true/false⟩`

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

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

round-half

`round-half = ⟨default/commercial⟩`

This option is only important for half-way numbers (e.g. 0.005). By setting it to `default` the value will be rounded to the nearest even number. Setting it to `commercial` rounds the value away from zero.

It is set to `default` by ... default.

Note: `default` actually refers to the fact that it is the default rounding algorightm used by `\fp_eval:n { round( ) }` without a third argument.

```
                                \cusetup{round-half=default}
0 kg                            \cunum{0.005}{kg}\\
−0 kg                           \cunum{-0.005}{kg}\\
1.24 kg                         \cunum{1.245}{kg}\\
                                \cusetup{round-half=commercial}
0.01 kg                         \cunum{0.005}{kg}\\
−0.01 kg                        \cunum{-0.005}{kg}\\
1.25 kg                         \cunum{1.245}{kg}
```

9.2.5 Fractions

`eval-fraction` `eval-fraction = ⟨true/false⟩`

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

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

`fraction-command` `fraction-command = ⟨\command⟩`

Sets the command used for printing fractions equal to ⟨`\command`⟩. ⟨`\command`⟩ 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.

```
                                \newcommand\myfrac[2]{#1/#2}
                                \cusetup{fraction-command=\myfrac}
1/8                             \cuam{1/8}\\
1/2 kg                          \cunum{1/2}{kg}\\
4/5 °C                          \cunum{4/5}{C}\\
1 2/3 kg                        \cunum{1_2/3}{kg}\\
                                \cusetup{fraction-command=\nicefrac}
1/8                             \cuam{1/8}\\
1/2 kg                          \cunum{1/2}{kg}\\
4/5 °C                          \cunum{4/5}{C}\\
1 2/3 kg                        \cunum{1_2/3}{kg}
```

`fraction-inline` `fraction-inline = ⟨input containing #1 and #2⟩`

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

```
                                \cusetup{fraction-inline={#1/#2}}
1/8                             \cuam{1/8}\\
1/2 kg                          \cunum{1/2}{kg}\\
4/5 °C                          \cunum{4/5}{C}\\
1 2/3 kg                        \cunum{1_2/3}{kg}\\
                                \cusetup{fraction-inline={\nicefrac{#2}{#1}}}
8/1                             \cuam{1/8}\\
2/1 kg                          \cunum{1/2}{kg}\\
5/4 °C                          \cunum{4/5}{C}\\
1 3/2 kg                        \cunum{1_2/3}{kg}
```

9.2.6 spaces

mixed-fraction-space

`mixed-fraction-space = ⟨length⟩`

Sets the length between the fraction and the number in a mixed-fraction, default is `0.1em` (because I said so; if someone has some literature or sources to look up the space, please let me know).

1⅔	`\cuam{1_2/3}\\`
1⅔ kg	`\cunum{1_2/3}{kg}\\`
10⅔ kg	`\cunum{10_2/3}{kg}\\`
	`\cusetup{mixed-fraction-space=1em}`
1 ⅔	`\cuam{1_2/3}\\`
1 ⅔ kg	`\cunum{1_2/3}{kg}\\`
10 ⅔ kg	`\cunum{10_2/3}{kg}\\`
	`\cusetup{mixed-fraction-space=0em}`
1⅔	`\cuam{1_2/3}\\`
1⅔ kg	`\cunum{1_2/3}{kg}\\`
10⅔ kg	`\cunum{10_2/3}{kg}`

cutext-space

`cutext-space = ⟨string⟩`

⟨*string*⟩ is inserted between the numeral part and the unit part when using `\cutext` and `\Cutext`. By default it is set to `\space`. Use this option if you want to e.g. insert an unbreakable space.

1 kilogramme	`\cutext{1}{kg}\\`
10 kilogramme	`\Cutext{10}{kg}\\`
	`\cusetup{cutext-space=~}`
1 kilogramme	`\cutext{1}{kg}\\`
10 kilogramme	`\Cutext{10}{kg}\\`
	`\cusetup{cutext-space={}}`
1kilogramme	`\cutext{1}{kg}\\`
10kilogramme	`\Cutext{10}{kg}\\`
	`\cusetup{cutext-space={qwe}}`
1qwekilogramme	`\cutext{1}{kg}\\`
10qwekilogramme	`\Cutext{10}{kg}\\`

phrase-space

`phrase-space = ⟨string⟩`

⟨*string*⟩ is inserted between the numeral part and the phrase part while using `\cuam`. By default it is set to `\space`. Use this option if you want to e.g. insert an unbreakable space.

```
                                \selectlanguage{ngerman}
1 Dutzend                       \cuam{12}\\
12 Dutzend                      \cuam{144}\\
                                \cusetup{phrase-space=~}
1 Dutzend                       \cuam{12}\\
12 Dutzend                      \cuam{144}\\
                                \cusetup{phrase-space={}}
1Dutzend                        \cuam{12}\\
12Dutzend                       \cuam{144}\\
                                \cusetup{phrase-space={qwe}}
1qweDutzend                     \cuam{12}\\
12qweDutzend                    \cuam{144}\\
```

`value-unit-space` `value-unit-space = ⟨string⟩`

Change the spacing for `\cunum` between the printed amount(s) and the unit. The default value is `\thinspace`.

```
                                \selectlanguage{ngerman}
1 kg                            \cunum{1}{kg}\\
½ kg                            \cunum{1/2}{kg}\\
1–2 kg                          \cunum{1--2}{kg}\\
                                \cusetup{value-unit-space={\hspace{1em}}}
1  kg                           \cunum{1}{kg}\\
½  kg                           \cunum{1/2}{kg}\\
1–2  kg                         \cunum{1--2}{kg}\\
                                \cusetup{value-unit-space={}}
1kg                             \cunum{1}{kg}\\
½kg                             \cunum{1/2}{kg}\\
1–2kg                           \cunum{1--2}{kg}\\
                                \cusetup{value-unit-space={qwe}}
1qwekg                          \cunum{1}{kg}\\
½qwekg                          \cunum{1/2}{kg}\\
1–2qwekg                        \cunum{1--2}{kg}\\
```

9.2.7 label & refs

`recalculate-amount` `recalculate-amount = ⟨true/false⟩`

Set this option to `true` if you want to change your recipes to the given number of people set by `set-number-of-persons`. Note that only those values who have a label are changed.

`set-number-of-persons` `set-number-of-persons = ⟨integer⟩`

With this option you can determine the number of people your recipes are. Note that this option only has an effect on those who have a ⟨*label*⟩ given. It is set to `4` by default.

	`\culabel{anotherrecipe}{2}`
2 persons	`\curef{anotherrecipe}~persons\\`
1 kg	`\cunum<anotherrecipe>{1}{kg}\\`
1	`\cuam<anotherrecipe>{1}\\`
1 kilogramme	`\cutext<anotherrecipe>{1}{kg}\\`
2 persons	`\curef{anotherrecipe}~persons\\`
	`\cusetup{recalculate-amount=true}`
4 persons	`\curef{anotherrecipe}~persons\\`
2 kg	`\cunum<anotherrecipe>{1}{kg}\\`
2	`\cuam<anotherrecipe>{1}\\`
2 kilogramme	`\cutext<anotherrecipe>{1}{kg}\\`
20 kilogramme	`\Cutext[ref=anotherrecipe]{10}{kg}\\`
	`\cusetup{set-number-of-persons=3}`
3 persons	`\curef{anotherrecipe}~persons\\`
1.5 kg	`\cunum<anotherrecipe>{1}{kg}\\`
1.5	`\cuam<anotherrecipe>{1}\\`
1.5 kilogramme	`\cutext<anotherrecipe>{1}{kg}\\`
15 kilogramme	`\Cutext[ref=anotherrecipe]{10}{kg}\\`
	`\cusetup{set-number-of-persons=2}`
2 persons	`\curef{anotherrecipe}~persons\\`
1 kg	`\cunum<anotherrecipe>{1}{kg}\\`
1	`\cuam<anotherrecipe>{1}\\`
1 kilogramme	`\cutext<anotherrecipe>{1}{kg}\\`
10 kilogramme	`\Cutext[ref=anotherrecipe]{10}{kg}\\`
	`\cusetup{set-number-of-persons=1}`
1 person	`\curef{anotherrecipe}~person\\`
0.5 kg	`\cunum<anotherrecipe>{1}{kg}\\`
0.5	`\cuam<anotherrecipe>{1}\\`
0.5 kilogramme	`\cutext<anotherrecipe>{1}{kg}\\`
5 kilogramme	`\Cutext[ref=anotherrecipe]{10}{kg}\\`

`label` `label = ⟨string⟩*⟨integer⟩`

The key-value version of `\culabel`. It defines the label ⟨*string*⟩ which is originally for ⟨*integer*⟩ people. Please note that the `*` is mandatory as it separates the string from the integer. Note that each label is defined globally and must be unique.

	`\cusetup{label=Toast*1}`
1 person	`\curef{Toast}~person\\`
2	`\cuam<Toast>{2}\\`
2 dag	`\cunum<Toast>{2}{dag}\\`
	`\cusetup{recalculate-amount=true}`
4 persons	`\curef{Toast}~persons\\`
8	`\cuam<Toast>{2}\\`
8 dag	`\cunum<Toast>{2}{dag}`

get-label `get-label = ⟨label⟩`

The key-value version of `\curef`. Note that this key doesn't save the value inside a macro but rather prints it directly into the document.

```
                                \culabel{Schinken}{3}
3                               \cusetup{get-label=Schinken}\\
3                               \curef{Schinken}\\
                                \cusetup{recalculate-amount=true}
4                               \cusetup{get-label=Schinken}\\
4                               \curef{Schinken}\\
```

ref `ref = ⟨label⟩`

Instead of using the first optional arguments of the commands in section 2 you may use this option. It requires a valid value and throws an error if ⟨*label*⟩ is not defined.

```
                                \culabel{Kaese}{3}
10 dm                           \cunum<Kaese>[m=dm]{1}{m}\\
10 dm                           \cunum[ref=Kaese,m=dm]{1}{m}\\
                                \cusetup{recalculate-amount=true}
13.33 dm                        \cunum<Kaese>[m=dm]{1}{m}\\
13.33 dm                        \cunum[ref=Kaese,m=dm]{1}{m}
```

9.3 Weird options

check-temperature `check-temperature = ⟨true/false⟩`

Checks if the used temperature is below absolute zero. 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. To add new units see `add-temperature-to-check`.

add-temperature-to-check

```
add-temperature-to-check =
  {
    ⟨unit-key-1⟩ = ⟨minimum-value-1⟩ ,
    ⟨unit-key-2⟩ = ⟨minimum-value-2⟩ ,
    ...
  }
```

This option adds ⟨*unit-key-1*⟩ and so on to the list of units to be checked if `check-temperature` is active. The argument can be a comma-separated list of ⟨*unit-key*⟩ = ⟨*minimum-value*⟩. This sets the allowed minimum value of ⟨*unit-key*⟩ to ⟨*minimum-value*⟩.

For example, this package implements the allowed minimum values for the temperatures `C`, `F`, `K` and `Re` to be checked if `check-temperature` is active using:

```
\cusetup
  {
    add-temperature-to-check =
      {
        K = 0,
        C = -273.15 ,
```

```
      F = -459.67 ,
      Re = -218.52
    }
}
```

If you want to add a new value, for example degree Rømer (which has be defined in another example) you can write:

```
\cusetup
  {
    add-temperature-to-check = { Ro = -135.90375 }
  }
```

convert-to-eV `convert-to-eV = ⟨true/false⟩`

Converts (nearly) every unit in table 1 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 (and as you see this package doesn't support the arrangement of such huge numbers). Also you may want to check the values if you really want to use them, just to be sure (I've checked them several times and hope they are finally correct, but mistakes happen[10]).

```
\cusetup{convert-to-eV=true}
\cunum{1}{kg}\\
\cunum{1}{l}\\
\cunum{1}{J}\\
\cunum{1}{m}\\
\cunum{1}{C}\\
\cunum{1}{s}
```

$5609588650000000000000000000000000\,{}^{eV}/_{c^2}$
$130148929500000000\,{}^{c^3\hbar^3}/_{eV^3}$
6241509126000000000 eV
$5067730.76\,{}^{c\hbar}/_{eV}$
0.02 eV
$1519267461000000\,{}^{\hbar}/_{eV}$

reading

10 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. If you use mail 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 (commands, units) are most welcome. Please also add (if possible) an example of the desired output into the minimal example (and – if by mail – 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 some of the translations of the units given in section 5 or (for better overview) appendix A. I would need

- their singular (and plural) form,
- the gender,
- the printed symbol (if different),

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

As it can happen that not all translations are available, you can send the parts you know.

A Translations

This section contains the list of available translations. Each table shows the available translations for the printed unit, the unit-name (printed if `\cutext` or `\Cutext` is used) and the plural form (if different from the singular form). A second table shows the translations used for phrases (if given).

If a translation is not available a "—" is shown.

A.1 English

⟨*unit-key*⟩	printed unit	unit-name	(plural)	gender
kg	kg	kilogramme		m
dag	dag	decagramme		m
g	g	gramme		m
oz	oz	ounce		m
lb	lb	pound	(pounds)	m
C	°C	degree Celsius	(degrees Celsius)	m
F	°F	degree Fahrenheit	(degrees Fahrenheit)	m
Re	°Ré	degree Réaumur	(degrees Réaumur)	m
K	K	kelvin		m
d	d	day	(days)	m
h	h	hour	(hours)	m
min	min	minute	(minutes)	m
s	s	second	(seconds)	m
m	m	metre	(metres)	m
dm	dm	decimetre	(decimetres)	m
cm	cm	centimetre	(centimetres)	m
mm	mm	millimitre	(millimitres)	m
in	in	inch	(inches)	m
l	ℓ	litre	(litres)	m
dl	dl	decilitre	(decilitres)	m
cl	cl	centilitre	(centilitres)	m
ml	ml	millilitre	(millilitres)	m
cal	cal	calorie	(calories)	m
kcal	kcal	kilocalorie	(kilocalories)	m
J	J	joule	(joules)	m
kJ	kJ	kilojoule	(kilojoules)	m
eV	eV	electron volt		m
pn	pinch	pinch	(pinches)	m
EL	tbsp.	tablespoon	(tablespoons)	m
TL	tsp.	teaspoon	(teaspoons)	m
csp	csp.	coffeespoonful		m
dsp	dsp.	dessertspoonful		m
ssp	ssp.	saltspoonful		m
Msp	Msp.	Messerspitze	(Messerspitzen)	f
decimal-mark	—	.	—	m
one(m)	—	one	—	m
one(f)	—	one	—	m
one(n)	—	one	—	m

A.2 american

⟨*unit-key*⟩	printed unit	unit-name	(plural)	gender
kg	kg	kilogram		m
dag	dag	decagram		m
g	g	gram		m
oz	oz	ounce		m
lb	lb	pound	(pounds)	m
C	°C	degree Celsius	(degrees Celsius)	m
F	°F	degree Fahrenheit	(degrees Fahrenheit)	m
Re	°Ré	degree Réaumur	(degrees Réaumur)	m
K	K	kelvin		m
d	d	day	(days)	m
h	h	hour	(hours)	m
min	min	minute	(minutes)	m
s	s	second	(seconds)	m
m	m	meter	(meters)	m
dm	dm	decimeter	(decimeters)	m
cm	cm	centimeter	(centimeters)	m
mm	mm	millimiter	(millimiters)	m
in	in	inch	(inches)	m
l	ℓ	liter	(liters)	m
dl	dl	deciliter	(deciliters)	m
cl	cl	centiliter	(centiliters)	m
ml	ml	milliliter	(milliliters)	m
cal	cal	calorie	(calories)	m
kcal	kcal	kilocalorie	(kilocalories)	m
J	J	joule	(joules)	m
kJ	kJ	kilojoule	(kilojoules)	m
eV	eV	electron volt		m
pn	pn.	pinch	(pinches)	m
EL	tbsp.	tablespoon	(tablespoons)	m
TL	tsp.	teaspoon	(teaspoons)	m
csp	csp.	coffeespoonful		m
dsp	dsp.	dessertspoonful		m
ssp	ssp.	saltspoonful		m
Msp	Msp.	Messerspitze	(Messerspitzen)	f
decimal-mark	—	.	—	m
one(m)	—	one	—	m
one(f)	—	one	—	m
one(n)	—	one	—	m

A.3 German

⟨*unit-key*⟩	printed unit	unit-name	(plural)	gender
kg	kg	Kilogramm		n
dag	dag	Dekagramm		n
g	g	Gramm		n
oz	oz	Unze		f
lb	lb	Pfund		n
C	°C	Grad Celsius		m
F	°F	Grad Fahrenheit		m
Re	°Ré	Grad Réamur		m
K	K	Kelvin		n
d	d	Tag	(Tage)	m
h	h	Stunde	(Stunden)	f
min	min	Minute	(Minuten)	f
s	s	Sekunde	(Sekunden)	f
m	m	Meter		n
dm	dm	Dezimeter		n
cm	cm	Centimeter		n
mm	mm	Millimeter		n
in	in	Zoll		m
l	l	Liter		m
dl	dl	Deziliter		m
cl	cl	Centiliter		m
ml	ml	Milliliter		m
cal	cal	Kalorie	(Kalorien)	f
kcal	kcal	Kilokalorie	(Kilokalorien)	f
J	J	Joule		m
kJ	kJ	Kilojoule		m
eV	eV	Elektronenvolt		n
pn	Prise	Prise	(Prisen)	f
EL	EL	Esslöffel		m
TL	TL	Teelöffel		m
csp	KL	Mokkalöffel		m
dsp	dsp.	—		m
ssp	ssp.	—		m
Msp	Msp.	Messerspitze	(Messerspitzen)	f
decimal-mark	—	,	—	m
one(m)	—	ein	—	m
one(f)	—	eine	—	m
one(n)	—	ein	—	m

⟨*Phrase-key*⟩	phrase	(plural)	gender
12	Dutzend		n

Some further phrases, just to write them down (they are not implemented, as they are barely used).

⟨*number*⟩	name	Note		(plural)	gender
60	Schock	(5 Dutzend,	$12 * 5$)		n
144	Gros	(12 Dutzend,	$12 * 12$)		n
1728	Großgros	(12 Groß,	$12 * 144$)		n

Note that Großgros has other (probably more common) synonyms.

A.4 French

⟨*unit-key*⟩	printed unit	unit-name	(plural)	gender
kg	kg	kilogramme	(kilogrammes)	m
dag	dag	décagramme	(décagrammes)	m
g	g	gramme		m
oz	oz	once		f
lb	lb	livre	(livres)	f
C	°C	degré Celsius	(degrés Celsius)	m
F	°F	kelvin	(kelvins)	m
Re	°Ré	échelle Réaumur	(degrés Réaumur)	m
K	K	degré Fahrenheit	(degrés Fahrenheit)	m
d	d	jour	(jours)	m
h	h	heure	(heures)	f
min	min	minute	(minutes)	f
s	s	seconde	(secondes)	f
m	m	mètre	(mètres)	m
dm	dm	décimètre	(décimètres)	m
cm	cm	centimètre	(centimètres)	m
mm	mm	millimètre	(millimètres)	m
in	po	pouce	(pouces)	m
l	L	litre	(litres)	m
dl	dL	décilitre	(décilitres)	m
cl	cL	centilitre	(centilitres)	m
ml	mL	millilitre	(millilitres)	m
cal	cal	calorie		m
kcal	kcal	kilocalorie	(kilocalories)	m
J	J	joule	(joules)	m
kJ	kJ	kilojoule	(kilojoules)	m
eV	eV	électron-volt	(électron-volts)	m
pn	pinch	pincée		f
EL	EL	cuillére à soupe		f
TL	TL	cuillére à café		f
csp	csp.	—		m
dsp	dsp.	—		m
ssp	ssp.	—		m
Msp	Msp.	—		m
decimal-mark	—	.	—	m
one(m)	—	un	—	m
one(f)	—	une	—	m
one(n)	—	un	—	m

B Implementation

B.1 Beginning

⟨@@=cooking_units⟩

⟨*package⟩

Na dann, auf gehts!

```
\@ifpackageloaded {xparse}
  { }
  { \RequirePackage {xparse} }
\@ifpackageloaded {expl3}
  { }
  { \RequirePackage {expl3} }
```

: Package

```
\ProvidesExplPackage
  {cooking-units}
  {2017/03/10}
  {1.11}
  {Ein Paket fuer Kocheinheiten}
```

Checking if expl3 is up-to-date, otherwise abort the loading of the package.

```
\@ifpackagelater { expl3 } { 2017/03/07 }
  { }
  {
    \PackageError { cooking-units } { Support~package~expl3~too~old }
      {
        You~need~to~update~your~installation~of~the~bundles~'l3kernel'~and~
        'l3packages'.\MessageBreak
        Loading~cooking-units~will~abort!
      }
    \tex_endinput:D
  }
```

Loading some needed packages.

```
\@ifpackageloaded { translations } { } { \RequirePackage { translations } }
\@ifpackageloaded { xfrac } { } { \RequirePackage { xfrac } }
\@ifpackageloaded { l3keys2e } { } { \RequirePackage { l3keys2e } }
```

Checking if translations is up-to-date, otherwise abort the loading of the package.

```
\@ifpackagelater { translations } { 2017/08/31 }
  { }
  {
    \PackageError { cooking-units } { Support~package~translations~too~old }
      {
        You~need~to~update~your~installation~of~the~package~'translations'.\MessageBreak
        Loading~cooking-units~will~abort!
      }
    \tex_endinput:D
  }
```

Define the only load-time option for this package. If it is set, load package fmtcount.

```
\bool_new:N \g__cooking_units_opt_numeral_bool
\keys_define:nn { cooking-units }
  {
    use-numerals .bool_gset:N = \g__cooking_units_opt_numeral_bool ,
```

```
    use-numerals .default:n = { false },
  }
```

Now process the package options ...

```
\ProcessKeysOptions { cooking-units }
\bool_if:NT \g__cooking_units_opt_numeral_bool
  {
    \@ifpackageloaded { fmtcount } { } { \RequirePackage { fmtcount } }
  }
```

... and redefine the package option such that it cannot be used elsewhere.

```
\keys_define:nn { cooking-units }
  {
    use-numerals .code:n = { \msg_error:nnn { cooking-units } { load-time-option } { fmtcoun
  }
```

B.2 Defining Variables

\tl_replace_all:NVn
\tl_replace_once:NnV
\tl_replace_once:NVn
\tl_if_in:nVTF
\tl_if_in:NVTF
\tl_if_in:NVT
\fp_compare:cNnT
\fp_eval:c
\prop_get:cVc
\int_abs:c

Some variations of commands we will need later.

```
\cs_generate_variant:Nn \tl_replace_all:Nnn { NVn }
\cs_generate_variant:Nn \tl_replace_once:Nnn { NnV, NVn }
\cs_generate_variant:Nn \tl_if_in:nnTF { nVTF }
\cs_generate_variant:Nn \tl_if_in:NnTF { NVTF  }
\cs_generate_variant:Nn \tl_if_in:NnT { NVT }
\cs_generate_variant:Nn \fp_compare:nNnT { cNnT }
\cs_generate_variant:Nn \fp_eval:n { c }
\cs_generate_variant:Nn \prop_get:cVN { cVc }
\cs_generate_variant:Nn \int_abs:n { c }
\cs_generate_variant:Nn \tl_show:n { x , f }
```

(*End definition for* `\tl_replace_all:NVn` *and others. These functions are documented on page* **??**.)

__cooking_units_frac:nn

This command is used to print the fractions and can be changed accordingly.

```
\cs_new_eq:NN \__cooking_units_frac:nn \sfrac
```

(*End definition for* `\__cooking_units_frac:nn`.)

__cooking_units_print_numeral:n
__cooking_units_print_Numeral:n

This command is used to print the fractions and can be changed accordingly.

```
\cs_new:Npn \__cooking_units_print_numeral:n #1 {}
\cs_new:Npn \__cooking_units_print_Numeral:n #1 {}
\bool_if:NT \g__cooking_units_opt_numeral_bool
  {
    \cs_set_eq:NN \__cooking_units_print_numeral:n \numberstringnum
    \cs_set_eq:NN \__cooking_units_print_Numeral:n \Numberstringnum
  }
```

(*End definition for* `\__cooking_units_print_numeral:n` *and* `\__cooking_units_print_Numeral:n`.)

\l__cooking_units_change_unit_prop

Conversions of units are stored within this property list. If someone requests that `kg` should be changed into `g`, `kg` is stored as a key with the value `g`. If someone then uses the unit `kg` the value `g` is restored and the unit is changed accordingly.

```
\prop_new:N \l__cooking_units_change_unit_prop
```

(*End definition for* `\l__cooking_units_change_unit_prop`.)

\l__cooking_units_number_tmpa_tl
\l__cooking_units_number_tmpb_tl
\l__cooking_units_tmpa_tl
\l__cooking_units_tmpb_tl
\l__cooking_units_mixed_fraction_tl
\l__cooking_units_given_unit_tl
\l__cooking_units_option_unit_tl
\l__cooking_units_language_tl
\l__cooking_units_cunum_range_sign_tl
\l__cooking_units_cutext_range_sign_tl
\l__cooking_units_value_unit_space_tl
\l__cooking_units_input_digits_tl
\l__cooking_units_input_decimal_mark_tl
\l__cooking_units_input_value_signs_tl
cooking_units_input_allowed_special_signs_tl
\c__cooking_units_input_str_hash_one_tl
\l__cooking_units_input_range_sign_tl
\l__cooking_units_input_times_persons_sign
\l__cooking_units_cutext_space_tl
\l__cooking_units_cuphrase_space_tl
\l__cooking_units_translation_tmpa_tl
\l__cooking_units_cutext_last_value_tl
\l__cooking_units_phrase_phrase_tl

Quite a lot of tl's.

```
\tl_new:N \l__cooking_units_number_tmpa_tl
\tl_new:N \l__cooking_units_number_tmpb_tl
\tl_new:N \l__cooking_units_tmpa_tl
\tl_new:N \l__cooking_units_tmpb_tl
\tl_new:N \l__cooking_units_mixed_fraction_tl
\tl_new:N \l__cooking_units_given_unit_tl
\tl_new:N \l__cooking_units_option_unit_tl
\tl_new:N \l__cooking_units_language_tl
\tl_new:N \l__cooking_units_cunum_range_sign_tl
\tl_new:N \l__cooking_units_cutext_range_sign_tl
\tl_new:N \l__cooking_units_value_unit_space_tl
\tl_new:N \l__cooking_units_input_digits_tl
\tl_new:N \l__cooking_units_input_decimal_mark_tl
\tl_new:N \l__cooking_units_input_value_signs_tl
\tl_new:N \l__cooking_units_input_allowed_special_signs_tl
\tl_new:N \c__cooking_units_input_str_hash_one_tl
\tl_new:N \l__cooking_units_input_range_sign_tl
\tl_new:N \l__cooking_units_input_times_persons_sign
\tl_new:N \l__cooking_units_cutext_space_tl
\tl_new:N \l__cooking_units_cuphrase_space_tl
\tl_new:N \l__cooking_units_translation_tmpa_tl
\tl_new:N \l__cooking_units_cutext_last_value_tl
\tl_new:N \l__cooking_units_phantom_tl
\tl_new:N \l__cooking_units_phrase_phrase_tl
\tl_new:N \l__cooking_units_unit_key_not_allowed_tl
```

(*End definition for* `\l__cooking_units_number_tmpa_tl` *and others.*)

Setting some token lists to their default value. `str_hash_one_tl` is used for defining single keys. (You will see, I didn't have a better idea)

```
\tl_set:Nn \l__cooking_units_input_digits_tl { 0123456789 }
\tl_set:Nn \l__cooking_units_input_times_persons_sign { * }
\tl_set:Nn \l__cooking_units_input_range_sign_tl { -- }
\tl_set:Nn \l__cooking_units_input_decimal_mark_tl { . , }
\tl_set:Nn \l__cooking_units_input_value_signs_tl { + - }
\tl_set:Nn \l__cooking_units_input_allowed_special_signs_tl { ? }
\tl_set_rescan:Nnn \c__cooking_units_input_str_hash_one_tl
  { \char_set_catcode_letter:N \# } {#1}
\tl_set:Nn \l__cooking_units_cunum_range_sign_tl { -- }
\tl_set:Nn \l__cooking_units_cutext_range_sign_tl { -- }
\tl_set:Nn \l__cooking_units_value_unit_space_tl { \thinspace }
\tl_set:Nn \l__cooking_units_cutext_space_tl { \space }
\tl_set:Nn \l__cooking_units_cuphrase_space_tl { \space }
\tl_set:Nn \l__cooking_units_unit_key_not_allowed_tl { , / }
```

Flat out stolen from siunitx

```
\AtBeginDocument {
  \cs_if_free:cT { T@TS1 }
    {
      \DeclareFontEncoding { TS1 } { } { }
      \DeclareFontSubstitution { TS1 } { cmr } { m } { n }
    }
}
```

```
\DeclareTextSymbolDefault \c__cooking_units_minus_tl { TS1 }
\DeclareTextSymbol \c__cooking_units_minus_tl { TS1 } { 61 }
\AtBeginDocument {
  \@ifpackageloaded { fontspec }
    {
      \@ifpackageloaded { eulervm }
        { }
        {
          \int_const:Nn \c__cooking_units_minus_int { 8722 }
          \tl_set:Nn \c__cooking_units_minus_tl
            { \tex_char:D \c__cooking_units_minus_int }
        }
    }
    { }
}
```

\l__cooking_units_mixed_frac_dim

The dimension between the fraction and the mixed fraction part is stored within this macro. There is no real reason why I have chosen this distance to be 0.1em, I just thought that it looks best. But if someone has some ideas of how large this distance should be I am happy to listen.

```
\dim_new:N \l__cooking_units_mixed_frac_dim
\dim_set:Nn \l__cooking_units_mixed_frac_dim { 0.1 em }
```

(*End definition for* `\l__cooking_units_mixed_frac_dim`.)

oking_units_significant_figures_plus_one_int

Stores the round-precision inside. Not sure if 'significant figures' is the correct therm for this. Also computes the number plus 1, later I will count the tokens after the colon in a number (sorry, can't explain. If you have 123.4567, it gets "4567", 4 tokens, larger then the number plus 1, needs to be rounded).

```
\int_new:N \l__cooking_units_significant_figures_int
\int_new:N \l__cooking_units_significant_figures_plus_one_int
\int_set:Nn \l__cooking_units_significant_figures_int { 2 }
\int_set:Nn \l__cooking_units_significant_figures_plus_one_int { 2 + 1 }
```

(*End definition for* `\l__cooking_units_significant_figures_plus_one_int`.)

\l__cooking_units_print_numerals_below_int

Used if option 'use-numerals' is active. Uses numerals for integers smaller than this number. I learned this number at school.

```
\int_new:N \l__cooking_units_print_numerals_below_int
\int_set:Nn \l__cooking_units_print_numerals_below_int { 13 }
```

(*End definition for* `\l__cooking_units_print_numerals_below_int`.)

\l__cooking_units_number_of_persons_tmpa_int

Each recipe defined by `\culabel` defines a counter to store the number of persons the recipe is for. For calculation the value is retrived and stored inside this temporal counter.

```
\int_new:N \l__cooking_units_number_of_persons_tmpa_int
```

(*End definition for* `\l__cooking_units_number_of_persons_tmpa_int`.)

cooking_units_calc_for_number_of_persons_int

Not only the number of persons are recipe is for is needed for calculation, but also the number of persons you want the recipe to be. This information is stored here.

```
\int_new:N \l__cooking_units_calc_for_number_of_persons_int
\int_set:Nn \l__cooking_units_calc_for_number_of_persons_int { 4 }
```

(*End definition for* `\l__cooking_units_calc_for_number_of_persons_int`.)

```
\int_new:N \l__cooking_units_phrase_number_tl
```

\l__cooking_units_list_of_defined_keys_clist
\g__cooking_units_list_of_defined_units_clist

Sequence of defined units and keys. Units are defined globally as they create new commands, keys do not do that (I think). Could be my mistake.

```
\seq_new:N \l__cooking_units_list_of_defined_keys_seq
\seq_new:N \g__cooking_units_list_of_defined_units_seq
```

(*End definition for* `\l__cooking_units_list_of_defined_keys_clist` *and* `\g__cooking_units_list_of_defined_units_clist`.)

\g__cooking_units_allowed_unit_phrases_tl

sed for the keys 'one(m)', 'one(f)', etc. Those are special keys which cannot be used as units, but are processed by the commands in the language section as such.

```
\clist_new:N \g__cooking_units_allowed_special_keys_clist
```

(*End definition for* `\g__cooking_units_allowed_unit_phrases_tl`.)

\l__cooking_units_phrase_numbers_clist

Inside this list the numbers for which a phrase is defined is stored in. As this is language specific, the list is stored inside a language-sensitive command and retrived when needed.

```
\clist_new:N \l__cooking_units_phrase_numbers_clist
```

(*End definition for* `\l__cooking_units_phrase_numbers_clist`.)

\l__cooking_units_temperatures_to_check_seq

tores units which should be tested if `check-temperature` equals true.

```
\seq_new:N \l__cooking_units_temperatures_to_check_seq
```

(*End definition for* `\l__cooking_units_temperatures_to_check_seq`.)

\l__cooking_units_phrase_prop

Stores the number and he respective phrase. For example if "12" has the phrase "Dutzend", this key-value pair is stored inside.

```
\prop_new:N \l__cooking_units_phrase_prop
```

(*End definition for* `\l__cooking_units_phrase_prop`.)

\l__cooking_units_minus_bool
\l__cooking_units_round_decimal_part_bool
\l__cooking_units_error_bool
\l__cooking_units_eval_fractions_bool
\l__cooking_units_parse_input_bool
\l__cooking_units_round_to_int_bool
\l__cooking_units_special_sign_bool
\l__cooking_units_single_key_bool
\l__cooking_units_check_temperature_bool
\l__cooking_units_convert_to_eV_bool
\l__cooking_units_cutext_uppercase_word_bool
_cooking_units_error_for_unknown_value_bool
\l__cooking_units_using_cutext_bool
\l__cooking_units_cuam_old_bool
\l__cooking_units_calc_for_persons_bool
ooking_units_calc_because_ref_was_given_bool
\l__cooking_units_calc_persons_bool
\l__cooking_units_cutext_to_cunum_bool
\l__cooking_units_cutext_old_bool
\l__cooking_units_cutext_change_unit_bool
\l__cooking_units_round_commercial_bool
\l__cooking_units_use_phrases_bool
\l__cooking_units_check_if_phrase_used_bool

Some booleans we need later.

```
\bool_new:N \l__cooking_units_minus_bool
\bool_new:N \l__cooking_units_round_decimal_part_bool
\bool_new:N \l__cooking_units_error_bool
\bool_new:N \l__cooking_units_eval_fractions_bool
\bool_new:N \l__cooking_units_parse_input_bool
\bool_new:N \l__cooking_units_round_to_int_bool
\bool_new:N \l__cooking_units_special_sign_bool
\bool_new:N \l__cooking_units_single_key_bool
\bool_new:N \l__cooking_units_check_temperature_bool
\bool_new:N \l__cooking_units_convert_to_eV_bool
\bool_new:N \l__cooking_units_cutext_uppercase_word_bool
\bool_new:N \l__cooking_units_error_for_unknown_value_bool
\bool_new:N \l__cooking_units_using_cutext_bool
\bool_new:N \l__cooking_units_cuam_old_bool
\bool_new:N \l__cooking_units_calc_for_persons_bool
\bool_new:N \l__cooking_units_calc_because_ref_was_given_bool
\bool_new:N \l__cooking_units_calc_persons_bool
\bool_new:N \l__cooking_units_cutext_to_cunum_bool
\bool_new:N \l__cooking_units_cutext_old_bool
```

```
\bool_new:N \l__cooking_units_cutext_change_unit_bool
\bool_new:N \l__cooking_units_round_commercial_bool
\bool_new:N \l__cooking_units_use_phrases_bool
\bool_new:N \l__cooking_units_check_if_phrase_used_bool
\bool_new:N \l__cooking_units_local_numeral_bool
```

While rewriting the code I searched for those booleans a lot.

```
\bool_new:N \l__cooking_units_range_in_input_bool
\bool_new:N \l__cooking_units_fraction_in_input_bool
\bool_new:N \l__cooking_units_decimal_in_input_bool
```

(*End definition for* `\l__cooking_units_minus_bool` *and others.*)

Setting some of them to `true` or another boolean respectively.

```
\bool_set_true:N \l__cooking_units_parse_input_bool
\bool_set_true:N \l__cooking_units_cutext_change_unit_bool
\bool_set_eq:NN \l__cooking_units_local_numeral_bool \g__cooking_units_opt_numeral_bool
```

`\q__cooking_units_range` Replacing the sign "--" with `\q_@@_range` for testing.

```
\quark_new:N \q__cooking_units_range
```

(*End definition for* `\q__cooking_units_range`.)

`\q__cooking_units_no_translation` Note the spelling mistake in "available".

```
\quark_new:N \q__cooking_units_no_translation
```

(*End definition for* `\q__cooking_units_no_translation`.)

`\l__cooking_units_tmpa_fp` `\l__cooking_units_tmpa_clist` `\l__cooking_units_tmpa_prop` `\l__cooking_units_tmpb_prop` `\l__cooking_units_tmpa_seq` Some temporal stores which are used thoughout the code.

```
\fp_new:N \l__cooking_units_tmpa_fp
\clist_new:N \l__cooking_units_tmpa_clist
\prop_new:N \l__cooking_units_tmpa_prop
\prop_new:N \l__cooking_units_tmpb_prop
\seq_new:N \l__cooking_units_tmpa_seq
\int_new:N \l__cooking_units_tmpa_int
\int_new:N \l__cooking_units_tmpb_int
```

(*End definition for* `\l__cooking_units_tmpa_fp` *and others.*)

B.3 Keys

Let's define some keys.

```
\keys_define:nn { cooking-units }
  {
```

`eval-fraction` If set to true the fractions are evaluated.

```
    eval-fraction .bool_set:N = \l__cooking_units_eval_fractions_bool ,
    eval-fraction .default:n = { false } ,
```

(*End definition for* `eval-fraction`. *This function is documented on page 21.*)

round-precision Setting the round-precision. Setting those two at once to not calculate it every time.

```
round-precision .code:n =
  {
    \int_set:Nn \l__cooking_units_significant_figures_int {#1}
    \int_set:Nn \l__cooking_units_significant_figures_plus_one_int { #1 + \c_one }
  } ,
round-precision .default:n = { 2 } ,
```

(*End definition for* `round-precision`. *This function is documented on page* 20.)

round-to-int Rounding the results to an integer.

```
round-to-int .bool_set:N = \l__cooking_units_round_to_int_bool ,
round-to-int .default:n = { false } ,
```

(*End definition for* `round-to-int`. *This function is documented on page* 20.)

range-sign Setting the printed range sign and make a difference between cunum and c(C)utext.

```
range-sign .meta:n =
  {
    cunum-range-sign = {#1} ,
    cutext-range-sign = {#1}
  } ,
range-sign .default:n = { -- } ,
cunum-range-sign .tl_set:N = \l__cooking_units_cunum_range_sign_tl ,
cunum-range-sign .default:n = { -- } ,
cutext-range-sign .tl_set:N = \l__cooking_units_cutext_range_sign_tl ,
cutext-range-sign .default:n = { -- } ,
```

(*End definition for* `range-sign`. *This function is documented on page* 20.)

value-unit-space Setting the space between the value and the printed unit.

```
value-unit-space .tl_set:N = \l__cooking_units_value_unit_space_tl ,
value-unit-space .default:n = { \thinspace } ,
```

(*End definition for* `value-unit-space`. *This function is documented on page* 23.)

fraction-command Setting the fraction command

```
fraction-command .code:n = { \cs_set_eq:NN \__cooking_units_frac:nn #1 } ,
fraction-command .default:n = { \sfrac } ,
```

(*End definition for* `fraction-command`. *This function is documented on page* 21.)

fraction-inline Setting the code inline.

```
fraction-inline .code:n = { \cs_set:Npn \__cooking_units_frac:nn ##1##2 {#1} } ,
fraction-inline .default:n = { \sfrac {#1} {#2} } ,
```

(*End definition for* `fraction-inline`. *This function is documented on page* 22.)

mixed-fraction-space Setting the space between the mixed fraction part and the fraction.

```
mixed-fraction-space .dim_set:N = \l__cooking_units_mixed_frac_dim ,
mixed-fraction-space .default:n = { 0.1 em } ,
```

(*End definition for* `mixed-fraction-space`. *This function is documented on page* 22.)

parse-number Parse the numbers? If no the input is printed as is (after some safetyparsing).

```
parse-number .bool_set:N = \l__cooking_units_parse_input_bool ,
parse-number .default:n= { true } ,
```

(*End definition for* `parse-number`. *This function is documented on page* 19.)

add-special-sign Adding a (some) special sign(s) which is (are) allowed in the input.

```
add-special-sign .code:n =
  {
    \str_if_eq:nnTF {#1} { < }
      { \msg_error:nn { cooking-units } { <-not-allowed-as-special-sign } }
      { \tl_put_right:Nn  \l__cooking_units_input_allowed_special_signs_tl {#1} }
  } ,
add-special-sign .default:n = { } ,
```

(*End definition for* `add-special-sign`. *This function is documented on page* 18.)

set-special-sign Doing the same as above but also overrides the old signs.

```
set-special-sign .code:n =
  {
    \str_if_eq:nnTF {#1} { < }
      { \msg_error:nn { cooking-units } { <-not-allowed-as-special-sign } }
      { \tl_set:Nn  \l__cooking_units_input_allowed_special_signs_tl {#1} }
  } ,
```

(*End definition for* `set-special-sign`. *This function is documented on page* 18.)

input-range-sign Don't wanna use `--` as a range sperator in \cunum? Use this option.

```
input-range-sign .tl_set:N = \l__cooking_units_input_range_sign_tl ,
input-range-sign .default:n = { -- } ,
```

(*End definition for* `input-range-sign`. *This function is documented on page* ??.)

check-temperature Weird option. Checking the temperature, if the temperature is below the absolute zero temperature it raises an error.

```
check-temperature .bool_set:N = \l__cooking_units_check_temperature_bool ,
check-temperature .default:n = { true } ,
```

add-temperature-to-check Adds a temperature to check for `check-temperature`. It uses the \keyval_parse:NNn command as this macro is used to parse keys (which is what I need).

```
add-temperature-to-check .code:n =
  {
    \keyval_parse:NNn
      \__cooking_units_temperature_to_check_print_error:n
      \__cooking_units_temperatures_to_check_define:nn
      {#1}
  },
temperature-to-check .value_required:n = { true } ,
```

convert-to-eV Another weird option, converts pretty much any unit defined by this package to electron volt or the respective derivative. As this is a unit transfrmation, it needs to be inside the group.

```
convert-to-eV .bool_set:N = \l__cooking_units_convert_to_eV_bool ,
convert-to-eV .default:n = { true } ,
convert-to-eV .groups:n = { change-unit } ,
```

Use numerals if the intger is below the integer set by this option.

use-numerals-below

```
use-numerals-below .int_set:N = \l__cooking_units_print_numerals_below_int ,
use-numerals-below .default:n = { 13 } ,
```

Sets the message for a special-sign to error, warning or none.

set-unknown-message

```
set-unknown-message .choices:nn =
  { error , warning , none }
  {
    \msg_redirect_name:nnn { cooking-units } { amount-not-known }
      { \l_keys_choice_tl }
  } ,
set-unknown-message .default:n = { warning } ,
```

Sets the message for a special-sign to error, warning or none.

set-unknown-message

```
set-cutext-translation-message .choices:nn =
  { error , warning , none }
  {
    \msg_redirect_name:nnn { cooking-units } { cutext-no-translation-available }
      { \l_keys_choice_tl }
  } ,
set-cutext-translation-message .default:n = { warning } ,
```

Erasing all preset options.

erase-all-options

```
erase-all-options .code:n =
  {
    \seq_map_inline:Nn \g__cooking_units_list_of_defined_units_seq
      {
        \clist_clear:c { l__cooking_units_predefined_option_##1_clist }
      }
  },
```

Choosing between “normal” rounding to even and commercial rounding and sets the boolean accordingly.

round-half

```
round-half .choices:nn =
  { default , commercial }
  {
    \int_case:nn { \l_keys_choice_int }
      {
        { 1 } { \bool_set_false:N \l__cooking_units_round_commercial_bool }
        { 2 } { \bool_set_true:N \l__cooking_units_round_commercial_bool }
      }
  },
round-half .default:n = { default },
```

Setting the number of persons the recipe should be for.

set-number-of-persons

```
set-number-of-persons .int_set:N = \l__cooking_units_calc_for_number_of_persons_int ,
set-number-of-persons .default:n = { 4 } ,
```

Defines a label ... is \culabel as a key.

label

```
label .code:n =
  {
    \__cooking_units_label_and_persons:n {#1}
  } ,
label .value_required:n = { true } ,
```

\curef as a key.

get-label

```
290     get-label .code:n =
291       {
292         \__cooking_units_curef:n {#1}
293       } ,
294     label .value_required:n = { true } ,
```

The <> option for keys.

ref

```
295     ref .code:n =
296       {
297         \__cooking_units_reference_label_and_persons:n {#1}
298       } ,
299     ref .value_required:n = { true } ,
```

Some keys with horrible option names. Reverts the respective command back to its older state (pre v1.10).

cuam-version
cutext-version

```
300     cuam-version .choices:nn =
301       { new , old }
302       {
303         \int_case:nn { \l_keys_choice_int }
304           {
305             { 1 } { \bool_set_false:N \l__cooking_units_cuam_old_bool }
306             { 2 } { \bool_set_true:N \l__cooking_units_cuam_old_bool }
307           }
308       } ,
309     cuam-version .default:n = { new } ,
310     cutext-version .choices:nn =
311       { new , old }
312       {
313         \int_case:nn { \l_keys_choice_int }
314           {
315             { 1 } { \bool_set_false:N \l__cooking_units_cutext_old_bool }
316             { 2 } { \bool_set_true:N \l__cooking_units_cutext_old_bool }
317           }
318       } ,
319     cutext-version .default:n = { new } ,
```

Setting the number of persons your recipes should be for is not enough; it is also needed to tell the package to recalculate the amounts.

recalculate-amount

```
320     recalculate-amount .bool_set:N = \l__cooking_units_calc_for_persons_bool ,
321     recalculate-amount .default:n = { false } ,
```

Don't want any \cutext (or \Cutext) in your document? Use this option!

cutext-to-cunum

```
322     cutext-to-cunum .bool_set:N = \l__cooking_units_cutext_to_cunum_bool ,
323     cutext-to-cunum .default:n = { false } ,
```

The space used in \cutext between the number (or numeral) and unit.

cutext-space

```
324     cutext-space .tl_set:N = \l__cooking_units_cutext_space_tl ,
325     cutext-space .default:n = { \space } ,
```

Same as before, but for phrases

phrase-space

```
326     phrase-space .tl_set:N = \l__cooking_units_cuphrase_space_tl ,
327     phrase-space .default:n = { \space } ,
```

Do not wanna change units in \cutext? Use this option.

cutext-change-unit

```
328     cutext-change-unit .bool_set:N = \l__cooking_units_cutext_change_unit_bool ,
329     cutext-change-unit .default:n = { true } ,
```

Do not wanna use phrases in \cuam? Use this option!

use-phrases

```
330     use-phrases .bool_set:N = \l__cooking_units_use_phrases_bool ,
331     use-phrases .default:n = { true } ,
```

A not very good name, but I couldn't think of a better name.

print-numerals

```
332     print-numerals .bool_set:N = \l__cooking_units_local_numeral_bool ,
333     print-numerals .default:n = { true } ,
```

numeral-function
Numeral-function

```
334     numeral-function .code:n = { \cs_set_eq:NN \__cooking_units_print_numeral:n #1 } ,
335     Numeral-function .code:n = { \cs_set_eq:NN \__cooking_units_print_Numeral:n #1 } ,
```

Ending the definition of keys.

```
336   }
```

(*End definition for* `check-temperature` *and others. These functions are documented on page 26.*)

B.4 Messages

Defining messages.

Messages

I do not allow fractions and ranges in the same input. Maybe I will change this.

```
337 \msg_new:nnnn { cooking-units } { fraction-not-allowed-with-range }
338   {
339     'You' \ cannot \ use \ '/' \ ( and  \ '_' )\ in \ combination \
340     with \ '\l__cooking_units_input_range_sign_tl' \ in \  '#1'.
341   }{
342     You \ cannot \ use \ fractions \ with \ a \ range.
343     \msg_see_documentation_text:n { cooking-units }
344   }
```

Do not allow a _ without a /.

```
345 \msg_new:nnnn { cooking-units } { missing-slash }
346   {
347     You \ cannot \ use \ '_'  \ without \
348     '/' in '#1'.
349   }{
350     You \ cannot \ have \ a \ mixed \ fraction \ ('_') \ without \
351     a \ normal \ fraction \ ('/').
352     \msg_see_documentation_text:n { cooking-units }
353   }
```

Error message if unit is not known to this package.

```
354 \msg_new:nnnn { cooking-units } { unknown-unit }
355   {
356     The \ unit \ '#1' \ is \ not \ defined. \ Use \
357     \newcookingunit \ to \ define \ new \ units.
358   }{
359     Define \ units \ before \ using \ or \ check \ if \ the \
360     unit-key \ is \ written \ correctly.
361     \msg_see_documentation_text:n { cooking-units }
362   }
```

Error if unit is already defined.

```
\msg_new:nnnn { cooking-units } { unit-already-defined }
  { The\ unit \ '#1' \ is \ already \ defined. }
  {
    The \ unit-key \ is \ already \ defined. \ Please \ use \ a \ different \
    key \ for \ a \ new \ unit.
    \msg_see_documentation_text:n {cooking-units}
  }
```

Missing argument in `\cudefinesymbols` (et all).

```
\msg_new:nnnn { cooking-units } { missing-argument }
  { There \ is \ an \ missing \ argument. }
  {
    You \ probably \ have \ forgotten \ a \ curly-brace \ pair.
    \msg_see_documentation_text:n {cooking-units}
  }
```

If fractions are evaluated division by zero is not allowed.

```
\msg_new:nnnn { cooking-units } { Division-by-zero }
  { Division\ by \ zero \ is \ not \ allowed. }
  { See \ a \ math \ book \ of \ your \ choice \ or \ for \ example \ Wikipedia. }
```

Showing the not allowed token in the input. Hope this helps.

```
\msg_new:nnnn { cooking-units } { Token-not-allowed }
  { The \ token \ '#1' \ is \ not \ allowed. }
  {
    The \ command \ accepts \ only \ a \ fixed \ number \ of \ tokens.
    \msg_see_documentation_text:n {cooking-units}
  }
```

A second decimal sign is not allowed (No na net).

```
\msg_new:nnnn { cooking-units } { Second-decimal-sign-not-allowed }
  { A \ second \ decimal \ sign \ is \ not \ allowed. }
  {
    Perhaps \ you \ didn't \ type \ it \ correctly.
    \msg_see_documentation_text:n {cooking-units}
  }
```

Error message for an undefined key.

```
\msg_new:nnnn { cooking-units } { Key-not-defined }
  {
    The \ key \ '#1' \ is \ not \ defined. \ Use \ \cudefinekeys or \
    \cudefinesinglekey to \ define \ keys.
  }
  {
    This \ key \ is \ not \ defined, \ perhaps \ you  \ misspelled \ it.
    \msg_see_documentation_text:n {cooking-units}
  }
```

If the temperature is too low print this error message. Now prints all units for which a zero-point is defined.

```
\msg_new:nnnn { cooking-units } { Temperature-too-low }
  {
    The \ temperature \ '#1' \ is \ too \ low.\ It \ cannot \ be \
    below \ the \ absolute \ zero - point \ of \ '#2'. \ Note \ that \
    the \ temperatures \
```

```
    '\seq_use:Nnnn \l__cooking_units_temperatures_to_check_seq
      { ', ~ ' } { ', ~ ' } { ' ~ and ~ ' }' \
    are \ rounded \ to \ integers.
    \\\\
    You \ can \ disable \ the \ option \ 'check-temperature' \ to \
    disable \ this \ error.
  }
  { See \ for \ example \ Wikipedia. }
```

If for an unit-key the value is wrong the following error message is shown.

```
\msg_new:nnnn { cooking-units } { key-choice-unknown }
  {
    The \ key \ '#1' \ only \ accepts \ only \
    '#3' \ as \ a \ set \ of \ choices \ and \ '#2' \ is \ non \ of \ these.
  }
  {
    The \ key \ accepts \ accepts\ only \ a \ fixed \ set \ of \ choices. \
    You \ can \ add \ new \ choices \ via \ \cuaddkeys, \ \cuaddsinglekeys \
    and \ \cuaddtokeys{}.
    \msg_see_documentation_text:n {cooking-units}
  }
```

An info message for unknown messages.

```
\msg_new:nnnn { cooking-units } { amount-not-known }
  {
    The \ amount \ of \ #1 \ is \ not \ known \ at \ line \
    \msg_line_number: .
  }
  {
    You \ used\  a\  special\  sign\  indicating\  that\  the\  true\  amount\  of\  the\
    specific\  ingredient\  is\  (was) \ not\  known\  to\  you. This\  message\
    reminds\  you\  about\  that\  fact.
    \msg_see_documentation_text:n {cooking-units}
  }
```

If a load time option is not used as a package option, this message is shown.

```
\msg_new:nnnn { cooking-units } { load-time-option }
  {
    The \ option \ '#1' \ is \ only \ available \ as \ a \ load-time-option. \
    Please \ set \ it \ as \ a \ package \ option.
  }
  {
    You \ cannot \ set \ this \ option \ using \ \cusetup \
    but \ only \ as \ a \ package \ option.
    \msg_see_documentation_text:n { cooking-units }
  }
```

Messages for obsolete commands.

```
\msg_new:nnnn { cooking-units } { obsolete-command }
  {
    Command \ #1 is \ obsolete. \ Please \ use \ #2 instead.
  }
  {
    Don't \ use \ this \ old \ command \ ...
    \msg_see_documentation_text:n { cooking-units }
  }
```

Error message if a new temperature to check is defined and no minimal value is given.

```
\msg_new:nnnn { cooking-units } { No-Value-given }
  {
    Please \ input \ a \ number \ to \ check \ for \
    'check-temperature'.
  }
  {
    A \ minimum \ value \ is \ needed \ for \ testing \ if \
    'check-temperature' \ is \ active.
    \msg_see_documentation_text:n { cooking-units }
  }
```

Error message if a zero-point temperature is already defined.

```
\msg_new:nnnn { cooking-units } { Minimum-already-defined }
  {
    A \ minimum \ for \ '#1' \ has \ already \ been \  defined.
  }
  {
    You \ cannot \ redefine \ it.
    \msg_see_documentation_text:n { cooking-units }
  }
```

Using the key version of \culabel one needs to give the number of people the recipe is for after a "*".

```
\msg_new:nnnn { cooking-units } { Number-of-persons-missing }
  {
    Please \ add \ the \ number \ of \ persons \ this \ recipe \ is \ for \ in \ '#1'. \
    Note \ that\ the \ number \ must \ be \ given \ after \ a \ '*'.
  }
  {
    Write \ 'Schweinsbraten*4' \ to \ create \ the \ label \ 'Schweinsbraten' \
    which \ is \ initially \ for \ 4 \ persons.
    \msg_see_documentation_text:n { cooking-units }
  }
```

The number of persons, must be an integer ...

```
\msg_new:nnnn { cooking-units } { Number-of-persons-is-not-an-integer }
  {
    The \ number \ of \ persons \ the \ recipe \ is \ for \ must \ be \ an \
    integer. \ '#1' \ is \ not \ allowed.
  }
  {
    The \ number \ '#1' \ is \ not \ allowed.
    \msg_see_documentation_text:n { cooking-units }
  }
```

Each label defined by \culabel needs to be new.

```
\msg_new:nnnn { cooking-units } { label-already-defined }
  {
    The \ label \ '#1' \ has \ already \ been \ defined.
  }
  {
    Each \ label \ must \ be \ unique.
    \msg_see_documentation_text:n { cooking-units }
  }
```

Message if a label is not defined.

```
\msg_new:nnnn { cooking-units } { label-not-defined }
  {
    The \ label \ is \ not \ defined. \ Please \ note \ that \ a \ label
    \ has \ to \ defined \ before \ it \ is \ referenced.
  }
  {
    Define \ the \ label \ before \ using \ it.
    \msg_see_documentation_text:n { cooking-units }
  }
```

If an unit is already defined and redefined by an \declarecookingunit.

```
\msg_new:nnnn { cooking-units } { redefine-unit }
  {
    The \ unit \ '#1' \ is \ redefined \ by \ \declarecookingunit at \ \msg_line_context: .
  }
  {
    \msg_see_documentation_text:n { cooking-units }
  }
```

A "phrase" must be an integer.

```
\msg_new:nnnn { cooking-units } { phrase-unit-not-an-integer }
  {
    A \ phrase \ must \ be \ an \ integer,  \ '#1' \ is \ not \ allowed.
  }
  {
    You \ can \ only \ use \ integers.
    \msg_see_documentation_text:n { cooking-units }
  }
```

A translation for \cutext or \Cutext is not available. For this case – instead of printing the keyname (see translations) – we fall back to \cunum.

```
\msg_new:nnnn { cooking-units } { cutext-no-translation-available } %% ToDo
  {
    For \ the \ unit \ '#1' \ there \ exists \ no \ translation \ to \ be \ used \
    for \ \cutext and \ \Cutext{}. \
    You \ can \ define \ new \ translations \ for \ a \ given \ language \ using \
    \cudefinename{}.
  }
  {
    \cunum \ is \ used \ instead.
    \msg_see_documentation_text:n { cooking-units }
  }
\msg_new:nnnn { cooking-units } { Translation-not-available } %% ToDo
  {
    The \ translation \ for \ #1 \ does \ not \ exist.
    Please \ define \ it \ using \ \cudefinename{}.
  }
  {
    And \ you \ may \ send \ me \ the \ translation \ as \ it \ is \
    not \ available \ yet.
    \msg_see_documentation_text:n { cooking-units }
  }
```

```
\msg_new:nnnn { cooking-units } { <-not-allowed-as-special-sign }
  {
    Currently \ (and \ probably \ forever) \ the \ sign \ '<' \
    is \ not \ allowed \ to \ be \ used \ as \ a \ special \ sign.
  }
  {
    I \ apologize \ for \ the \ inconvenience.
    \msg_see_documentation_text:n { cooking-units }
  }
\msg_new:nnnn { cooking-units } { unknown-gender }
  {
    '#1' \ is \ not \ allowed \ to \ be \ used \ as \ a \ gender-specification.
    Only \ 'm', \ 'f' \ or \ 'n' \ are \ allowed
  }{
    Please \ remove \ spaces \ if \ there \ are \ some.
    \msg_see_documentation_text:n { cooking-units }
  }
```

(*End definition for* `Messages`. *This function is documented on page* **??**.)

C Helper Macros

The name says it all.

`\__cooking_units_error_if_unit_not_defined:n`
`\__cooking_units_error_if_unit_not_defined:V`

A little helper macro. Checks if the unit is defined, if not raise an error.

```
\cs_new:Npn \__cooking_units_error_if_unit_not_defined:n #1
  {
    \seq_if_in:NnF \g__cooking_units_list_of_defined_units_seq {#1}
      { \msg_error:nnn { cooking-units } { unknown-unit } {#1} }
  }
\cs_generate_variant:Nn \__cooking_units_error_if_unit_not_defined:n { V }
```

(*End definition for* `\__cooking_units_error_if_unit_not_defined:n`.)

`\__cooking_units_if_integer:n`
`\__cooking_units_if_parse_and_integer:n`

Checking if the input consists only of numbers. Furthermore tests if the input should be parsed at all.

```
\prg_new_protected_conditional:Npnn \__cooking_units_if_integer:n #1 { TF , F , T }
  {
    \tl_map_inline:nn {#1}
      {
        \tl_if_in:NnF \l__cooking_units_input_digits_tl {##1}
          {
            \tl_map_break:n { \use_iii:nnn }
          }
      }
    \use_i:nn \prg_return_true: \prg_return_false:
  }
\cs_generate_variant:Nn \__cooking_units_if_integer:nTF { V }
\cs_generate_variant:Nn \__cooking_units_if_integer:nT { V }
\cs_generate_variant:Nn \__cooking_units_if_integer:nF { V }
\prg_new_protected_conditional:Npnn \__cooking_units_if_parse_and_integer:n #1 { TF }
  {
    \bool_if:NTF \l__cooking_units_parse_input_bool
```

```
      {
        \__cooking_units_if_integer:nTF {#1}
          { \prg_return_true: }
          { \prg_return_false: }
      }
      { \prg_return_false: }
  }
\cs_generate_variant:Nn \__cooking_units_if_parse_and_integer:nTF { V }
```

(*End definition for* `\__cooking_units_if_integer:n` *and* `\__cooking_units_if_parse_and_integer:n`.)

`\__cooking_units_fp_if_equal_one:nTF`
`\__cooking_units_int_if_equal_one:nTF`

```
\prg_new_conditional:Npnn \__cooking_units_fp_if_equal_one:n #1 { TF }
  {
    \fp_compare:nNnTF {#1} = { \c_one_fp }
      { \prg_return_true: }
      { \prg_return_false: }
  }
\prg_new_conditional:Npnn \__cooking_units_int_if_equal_one:n #1 { TF }
  {
    \int_compare:nNnTF {#1} = { \c_one }
      { \prg_return_true: }
      { \prg_return_false: }
  }
```

(*End definition for* `\__cooking_units_fp_if_equal_one:nTF` *and* `\__cooking_units_int_if_equal_one:nTF`.)

`ooking_units_check_if_correct_gender_input:n`

```
\cs_new:Npn \__cooking_units_check_if_correct_gender_input:n #1
  {
    \str_case:nnF {#1}
      {
        { m } {}
        { f } {}
        { n } {}
      } { \msg_error:nnn { cooking-units } { unknown-gender } {#1} }
  }
```

(*End definition for* `\__cooking_units_check_if_correct_gender_input:n`.)

`\__cooking_units_convert_to_eV:` Still work in progress. Will probably forever be. Changes all predefined units into natural units (except for those who cannot be changed, like Msp. for example).

```
\seq_new:N \g__cooking_units_natural_units_seq
\prop_new:N \g__cooking_units_convert_to_eV_remember_prop
\seq_set_split:Nnn \g__cooking_units_natural_units_seq { , }
  {
    eV ,
    eVc-2 ,
    hbareV-1 ,
    chbareV-1 ,
    (chbareV-1)3 ,
  }
\cs_new:Npn \__cooking_units_keys_set:nnn #1#2#3
  {
```

```
    \keys_set:nn {#1} { #2 = #3 }
  }
\cs_generate_variant:Nn \__cooking_units_keys_set:nnn { nVV , nVn }
\cs_new:Npn \__cooking_units_convert_to_eV:
  {
    \prop_if_exist:cT { l__cooking_units_cukeys_ \l__cooking_units_given_unit_tl _prop }
      {
        \prop_get:NVNTF \g__cooking_units_convert_to_eV_remember_prop
          \l__cooking_units_given_unit_tl \l__cooking_units_tmpa_tl
          {
            \__cooking_units_keys_set:nVV { cooking-units } \l__cooking_units_given_unit_tl
          }{
            \prop_get:cnNT { l__cooking_units_cukeys_ \l__cooking_units_given_unit_tl _prop
              {
                \seq_map_inline:Nn \g__cooking_units_natural_units_seq
                  {
                    \seq_if_in:NnT \l__cooking_units_tmpa_seq {##1}
                      {
                        \prop_gput:NVn \g__cooking_units_convert_to_eV_remember_prop
                          \l__cooking_units_given_unit_tl {##1}
                        \__cooking_units_keys_set:nVn { cooking-units } \l__cooking_units_gi
                        \seq_map_break:
                      }
                  }
              }
          }
      }
  }
```

(*End definition for* `\__cooking_units_convert_to_eV:`.)

`\__cooking_units_tl_set_fp_and_eval:Nn` I am using this construction often enough, so ... yeah.

```
\cs_new:Npn \__cooking_units_tl_set_fp_and_eval:Nn #1#2
  { \tl_set:Nx #1 { \fp_eval:n {#2} } }
\cs_generate_variant:Nn \__cooking_units_tl_set_fp_and_eval:Nn { Nc }
```

(*End definition for* `\__cooking_units_tl_set_fp_and_eval:Nn`.)

`ing_units_temperature_to_check_print_error:n`
`ooking_units_temperatures_to_check_define:nn` Commands used in the option `temperature-to-check`. If no value is given (first command) it raises an error. If a value is given the unit is added to a check-list and a new constant is defined.

```
\cs_new:Npn \__cooking_units_temperature_to_check_print_error:n #1
  {
    \msg_error:nn { cooking-units } { No-Value-given }
  }
\cs_new:Npn \__cooking_units_temperatures_to_check_define:nn #1#2
  {
    \__cooking_units_error_if_unit_not_defined:n {#1}
    \fp_if_exist:cTF { c__cooking_units_ #1 _min_fp }
      { \msg_warning:nnn { cooking-units } { Minimum-already-defined } {#1} }
      {
        \seq_put_right:Nn \l__cooking_units_temperatures_to_check_seq {#1}
        \fp_const:cn { c__cooking_units_ #1 _min_fp } {#2}
      }
  }
```

(*End definition for* `\__cooking_units_temperature_to_check_print_error:n` *and* `\__cooking_units_temperatures_to_check_define:nn`.)

`\__cooking_units_rounding_function:n` I think this definition is somewhat stupid, but I don't have a better idea.

```
\cs_new:Npn \__cooking_units_rounding_function:n #1
  {
    #1
      \bool_if:NTF \l__cooking_units_round_commercial_bool
        {
          ,
          \bool_if:NTF \l__cooking_units_round_to_int_bool
            { \c_zero_fp }
            { \l__cooking_units_significant_figures_int }
          , #1
        }
        { \bool_if:NF \l__cooking_units_round_to_int_bool { , \l__cooking_units_significant_
  }
```

(*End definition for* `\__cooking_units_rounding_function:n`.)

`\__cooking_units_label_and_persons:n`

```
\cs_new:Npn \__cooking_units_label_and_persons:n #1
  {
    \tl_if_in:nnTF {#1} { * }
      {
        \__cooking_units_label_and_persons_times_persons:ww #1 \q_stop
      }{
        \msg_error:nnn { cooking-units} { Number-of-persons-missing } {#1}
      }
  }
```

(*End definition for* `\__cooking_units_label_and_persons:n`.)

```
\cs_new:Npn \__cooking_units_label_and_persons_times_persons:ww #1 * #2 \q_stop
  {
    \__cooking_units_culabel:nn {#1} {#2}
  }
```

`\__cooking_units_culabel:nn`
`\__cooking_units_curef:n`

```
\cs_new:Npn \__cooking_units_culabel:nn #1#2
  {
    \int_if_exist:cTF { l__cooking_units_number_of_persons_ #1 _int }
      {
        \msg_error:nnn { cooking-units } { label-already-defined } {#1}
      }{
        \__cooking_units_if_integer:nTF {#2}
          {
            \int_new:c { l__cooking_units_number_of_persons_ #1 _int }
            \int_gset:cn { l__cooking_units_number_of_persons_ #1 _int } {#2}
          }
          { \msg_error:nnn { cooking-units } { Number-of-persons-is-not-an-integer } {#2} }
      }
  }
```

(*End definition for* `\__cooking_units_culabel:nn` *and* `\__cooking_units_curef:n`.)

_cooking_units_reference_label_and_persons:n

```
\cs_new:Npn \__cooking_units_reference_label_and_persons:n #1
  {
    \int_if_exist:cTF { l__cooking_units_number_of_persons_ #1 _int }
      {
        \bool_set_true:N \l__cooking_units_calc_because_ref_was_given_bool
        \int_set_eq:Nc \l__cooking_units_number_of_persons_tmpa_int
          { l__cooking_units_number_of_persons_ #1 _int }
      }
      { \msg_error:nnn { cooking-units } { label-not-defined } {#1} }
  }
```

(*End definition for* `\__cooking_units_reference_label_and_persons:n`.)

`\__cooking_units_curef:n`

```
\cs_new:Npn \__cooking_units_curef:n #1
  {
    \int_if_exist:cTF { l__cooking_units_number_of_persons_ #1 _int }
      {
        \bool_if:NTF \l__cooking_units_calc_for_persons_bool
          { \int_use:N \l__cooking_units_calc_for_number_of_persons_int }
          { \int_use:c { l__cooking_units_number_of_persons_ #1 _int } }
      }{ \msg_error:nnn { cooking-units } { label-not-defined } {#1} }
  }
```

(*End definition for* `\__cooking_units_curef:n`.)

`\culabel`
`\curef`

```
\NewDocumentCommand \culabel { m m } { \__cooking_units_culabel:nn {#1} {#2} }
\NewExpandableDocumentCommand \curef { m } { \__cooking_units_curef:n {#1} }
```

(*End definition for* `\culabel` *and* `\curef`. *These functions are documented on page* **??**.)

oking_units_tl_if_in_remove_and_set_bool:NnN

Can probably optimize this as the sign is always at the first place (this is done after cheking the input, so the asumption is safe), but yeah ...

```
\cs_new:Npn \__cooking_units_tl_if_in_remove_and_set_bool:NnN #1#2#3
  {
    \tl_if_in:NnTF #1 {#2}
      {
        \bool_set_true:N #3
        \tl_remove_once:Nn #1 {#2}
      }
      { \bool_set_false:N #3 }
  }
```

(*End definition for* `\__cooking_units_tl_if_in_remove_and_set_bool:NnN`.)

g_units_tl_if_in_remove_and_reverse_bool:NnN

```
\cs_new:Npn \__cooking_units_tl_if_in_remove_and_reverse_bool:NnN #1#2#3
  {
    \tl_if_in:NnT #1 {#2}
      {
        \bool_if:NTF #3
          { \bool_set_false:N #3 }
          { \bool_set_true:N #3 }
```

```
        \tl_remove_once:Nn #1 {#2}
      }
  }
```

(*End definition for* `\__cooking_units_tl_if_in_remove_and_reverse_bool:NnN`.)

C.1 Language Macros

```
\tl_const:Nn \c__cooking_units_postfix_unit_tl { ( cu-unit ) }
\tl_const:Nn \c__cooking_units_postfix_unit_pl_tl { ( cu-unit-pl ) }
\tl_const:Nn \c__cooking_units_postfix_unitname_tl { ( cu-unitname ) }
\tl_const:Nn \c__cooking_units_postfix_unitname_pl_tl { ( cu-unitname-pl ) }
\tl_const:Nn \c__cooking_units_postfix_gender_tl { ( cu-unitgender ) }
\tl_const:Nn \c__cooking_units_postfix_phrase_tl { ( cu-unitphrase ) }
```

`\__cooking_units_deftranslation:nn`
`\__cooking_units_deftranslation:xxn`
`\__cooking_units_deftranslation:Vnv`

```
\cs_new:Npn \__cooking_units_deftranslation_base:nnn #1#2#3
  {
    \declaretranslationfallback { #1 #2 } {#3}
  }
\cs_new:Npn \__cooking_units_deftranslation_to:nnnn #1#2#3#4
  {
    \declaretranslation {#1} { #2 #3 } {#4}
  }
\cs_generate_variant:Nn \__cooking_units_deftranslation_base:nnn { xx , xxV }
\cs_generate_variant:Nn \__cooking_units_deftranslation_to:nnnn { Vxxv, VxxV , Vxxn, Vxxx,Vx
```

(*End definition for* `\__cooking_units_deftranslation:nn` *and* `\__cooking_units_deftranslation:xxn`.)

`\__cooking_units_newtranslation:xxn`
`\__cooking_units_newtranslation:nxxn`

```
\cs_new:Npn \__cooking_units_newtranslation_base:nnn #1#2#3
  {
    \definetranslationfallback { #1 #2 } {#3}
  }
\cs_new:Npn \__cooking_units_newtranslation_to:nnnn #1#2#3#4
  {
    \definetranslation {#1} { #2 #3 } {#4}
  }
\cs_generate_variant:Nn \__cooking_units_newtranslation_base:nnn { nVn }
\cs_generate_variant:Nn \__cooking_units_newtranslation_to:nnnn { nnVn }
```

(*End definition for* `\__cooking_units_newtranslation:xxn` *and* `\__cooking_units_newtranslation:nxxn`.)

`\__cooking_units_translate:nn`
`\__cooking_units_translate:xx`
`\__cooking_units_translate_let:Nxx`
`\__cooking_units_translate_let:Nxx`

```
\cs_new:Npn \__cooking_units_translate:nn #1#2
  {
    \GetTranslation { #1 #2 }
  }
\cs_new:Npn \__cooking_units_translate_let:Nnn #1#2#3
  {
    \SaveTranslation {#1} { #2 #3 }
  }
\cs_new:Npn \__cooking_units_translate_let:nNnn #1#2#3#4
  {
    \SaveTranslationFor {#2} {#1} { #3 #4 }
```

```
  }
\cs_generate_variant:Nn \__cooking_units_translate:nn { xx  }
\cs_generate_variant:Nn \__cooking_units_translate_let:Nnn { Nxx }
\cs_generate_variant:Nn \__cooking_units_translate_let:nNnn { VNxx, nNxx }
```

(*End definition for* `\__cooking_units_translate:nn` *and* `\__cooking_units_translate_let:Nxx`.)

`\__cooking_units_unitname_get:NnTF`
`\__cooking_units_unitname_get:NxTF`

```
\prg_new_conditional:Npnn \__cooking_units_unitname_get:Nn #1#2 { F }
  {
    \__cooking_units_translate_let:Nxx #1 {#2} \c__cooking_units_postfix_unitname_tl
    \tl_if_eq:NNTF #1 \q__cooking_units_no_translation
      { \prg_return_false: }
      { \prg_return_true: }
  }
\cs_generate_variant:Nn \__cooking_units_unitname_get:NnF { Nx }
```

(*End definition for* `\__cooking_units_unitname_get:NnTF`.)

`its_translate_one_to_and_check_existance:Nx`

```
\cs_new:Npn \__cooking_units_translate_one_to_and_check_existance:Nx #1#2
  {
    \__cooking_units_translate_let:Nxx #1 {#2} \c__cooking_units_postfix_gender_tl
    \__cooking_units_unitname_get:NxF #1 { one (#1) }
      {
        \__cooking_units_translate_let:Nxx #1 {#2} \c__cooking_units_postfix_gender_tl
        \msg_error:nnn { cooking-units } { Translation-not-available } { one (#1) }
      }
  }
```

(*End definition for* `\__cooking_units_translate_one_to_and_check_existance:Nx`.)

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

Defining the translation for the decimal-mark. Note that those 'phrases' are stored inside `\g_@@_allowed_special_keys_clist`. Furthermore some translations are defined.

```
\clist_gset:Nn \g__cooking_units_allowed_special_keys_clist
  {
    decimal-mark ,
    one (m) ,
    one (f) ,
    one (n)
  }
```

(*End definition for* `decimal-mark` *and others. These functions are documented on page* 12.)

```
\__cooking_units_newtranslation_base:nVn { decimal-mark } \c__cooking_units_postfix_unitname
\__cooking_units_newtranslation_base:nVn { decimal-mark } \c__cooking_units_postfix_unitname
\__cooking_units_newtranslation_to:nnVn { German } { decimal-mark } \c__cooking_units_postfi
```

Note that the plural versions just exist for completing the set.

```
\__cooking_units_newtranslation_base:nVn { one (m) } \c__cooking_units_postfix_unitname_tl {
\__cooking_units_newtranslation_base:nVn { one (f) } \c__cooking_units_postfix_unitname_tl {
\__cooking_units_newtranslation_base:nVn { one (n) } \c__cooking_units_postfix_unitname_tl {
\__cooking_units_newtranslation_base:nVn { one (m) } \c__cooking_units_postfix_unitname_pl_t
\__cooking_units_newtranslation_base:nVn { one (f) } \c__cooking_units_postfix_unitname_pl_t
\__cooking_units_newtranslation_base:nVn { one (n) } \c__cooking_units_postfix_unitname_pl_t
```

```
\__cooking_units_newtranslation_to:nnVn { English } { one (m) } \c__cooking_units_postfix_un
\__cooking_units_newtranslation_to:nnVn { English } { one (f) } \c__cooking_units_postfix_un
\__cooking_units_newtranslation_to:nnVn { English } { one (n) } \c__cooking_units_postfix_un

\__cooking_units_newtranslation_to:nnVn { German } { one (m) } \c__cooking_units_postfix_uni
\__cooking_units_newtranslation_to:nnVn { German } { one (f) } \c__cooking_units_postfix_uni
\__cooking_units_newtranslation_to:nnVn { German } { one (n) } \c__cooking_units_postfix_uni
```

C.2 Parsing and checking numbers

This section contains macros helping an checking the input.

__cooking_units_parse_input:N
__cooking_units_parse_input:V

Splits the input into fractions or ranges (`q_@@_range`). Depending on the input on of the four macros are used.

```
\cs_new:Npn \__cooking_units_parse_input:N #1
  {
    \bool_if:NTF \l__cooking_units_range_in_input_bool
      { \__cooking_units_parse_range_in_input:ww #1 \q_stop }
      {
        \tl_if_in:nnTF {#1} { / }
          {
            \bool_lazy_or:nnTF
              { \l__cooking_units_calc_persons_bool }{ \l__cooking_units_eval_fractions_bool
              { \bool_set_false:N \l__cooking_units_fraction_in_input_bool }
              { \bool_set_true:N \l__cooking_units_fraction_in_input_bool }
            \tl_if_in:nnTF {#1} { _ }
              { \__cooking_units_parse_mixed_fraction_in_input:www #1 \q_stop }
              { \__cooking_units_parse_fraction_in_input:ww #1 \q_stop }
          }
          { \__cooking_units_parse_number_in_input:n {#1} }
      }
  }
\cs_generate_variant:Nn \__cooking_units_parse_input:N { V }
```

(*End definition for* `\__cooking_units_parse_input:N` *and* `\__cooking_units_parse_input:V`.)

__cooking_units_parse_number_in_input:n

If no fractions and ranges are used this macro activated. It just checks the input, stores the checked (and a bit changed) input into a macro. This macro is given to another function to calculate the input and print it.

```
\cs_new:Npn \__cooking_units_parse_number_in_input:n #1
  {
    \__cooking_units_parse_input_and_safe_in:nN {#1} \l__cooking_units_number_tmpa_tl
    \__cooking_units_process_and_print_number_in_input:N \l__cooking_units_number_tmpa_tl
  }
\cs_new:Npn \__cooking_units_process_and_print_number_in_input:N #1
  {
    \__cooking_units_pre_process_input:NN #1 \q_no_value
    \__cooking_units_calculate_and_store_in:N #1
    \__cooking_units_post_process_input:NN #1 \q_no_value
    \__cooking_units_print_input:N #1
  }
```

(*End definition for* `\__cooking_units_parse_number_in_input:n`.)

__cooking_units_parse_range_in_input:ww

Is used if a `\q_@@_range` is found inside the input. Seperates the input, checks it and prints it (after calculating it). Furthermore a boolean is set to true, is used to check for errors (ergo if fractions are used).

```
\cs_new:Npn \__cooking_units_parse_range_in_input:ww #1 \q__cooking_units_range #2 \q_stop
  {
    \__cooking_units_parse_input_and_safe_in:nN {#1} \l__cooking_units_number_tmpa_tl
    \__cooking_units_parse_input_and_safe_in:nN {#2} \l__cooking_units_number_tmpb_tl
    \__cooking_units_pre_process_input:NN \l__cooking_units_number_tmpa_tl \l__cooking_units
    \__cooking_units_calculate_and_store_in:N \l__cooking_units_number_tmpa_tl
    \__cooking_units_calculate_and_store_in:N \l__cooking_units_number_tmpb_tl
    \__cooking_units_post_process_input:NN \l__cooking_units_number_tmpa_tl \l__cooking_unit
    \__cooking_units_print_input:N \l__cooking_units_number_tmpa_tl
    \bool_if:NTF \l__cooking_units_using_cutext_bool
      { \tl_use:N \l__cooking_units_cutext_range_sign_tl }
      { \tl_use:N \l__cooking_units_cunum_range_sign_tl }
    \__cooking_units_print_input:N \l__cooking_units_number_tmpb_tl
  }
```

(*End definition for* `\__cooking_units_parse_range_in_input:ww`.)

__cooking_units_parse_fraction_in_input:ww

If a `/` (but no `_`) is found inside the input. Well ... does the same as the functions before. If fractions should be evaluated the input is ... well, evaluated and printed. Otherwise the input is given to another function which prints the fractions. Note that the empty argument in `\@@_formatiere_fractions:nnn` indicates a "normal" fraction.

```
\cs_new:Npn \__cooking_units_parse_fraction_in_input:ww #1/#2 \q_stop
  {
    \__cooking_units_parse_input_and_safe_in:nN {#1} \l__cooking_units_number_tmpa_tl
    \__cooking_units_parse_input_and_safe_in:nN {#2} \l__cooking_units_number_tmpb_tl
%    \bool_if:NTF \l__cooking_units_fraction_in_input_bool
    \bool_lazy_or:nnTF
      { \l__cooking_units_fraction_in_input_bool }
      { \l__cooking_units_special_sign_bool }
      {
        \__cooking_units_tl_if_in_remove_and_set_bool:NnN \l__cooking_units_number_tmpa_tl {
        \__cooking_units_tl_if_in_remove_and_reverse_bool:NnN \l__cooking_units_number_tmpb_
        \__cooking_units_formatiere_fractions:nnn
          { }
          { \l__cooking_units_number_tmpa_tl }
          { \l__cooking_units_number_tmpb_tl }
      }{
        \__cooking_units_tl_set_fp_and_eval:Nn \l__cooking_units_number_tmpa_tl
          { \l__cooking_units_number_tmpa_tl / \l__cooking_units_number_tmpb_tl }
        \__cooking_units_process_and_print_number_in_input:N \l__cooking_units_number_tmpa_t
      }
  }
```

(*End definition for* `\__cooking_units_parse_fraction_in_input:ww`.)

king_units_parse_mixed_fraction_in_input:www

The same procedure as last function? The same procedure as every function!

If it should be evaluated it is important to check if the mixed-fraction part is positive or negative. That's why the minuses are removed *before* checking if there is a fraction (in opposite to the command before).

$$1\,2/3 = 1 + 2/3 \tag{1}$$
$$=1\,2/3 = -1 - 2/3 \tag{2}$$

```
\cs_new:Npn \__cooking_units_parse_mixed_fraction_in_input:www #1_#2/#3 \q_stop
  {
    \__cooking_units_parse_input_and_safe_in:nN {#1} \l__cooking_units_mixed_fraction_tl
    \__cooking_units_parse_input_and_safe_in:nN {#2} \l__cooking_units_number_tmpa_tl
    \__cooking_units_parse_input_and_safe_in:nN {#3} \l__cooking_units_number_tmpb_tl
    \__cooking_units_tl_if_in_remove_and_set_bool:NnN \l__cooking_units_mixed_fraction_tl {
    \__cooking_units_tl_if_in_remove_and_reverse_bool:NnN \l__cooking_units_number_tmpa_tl {
    \__cooking_units_tl_if_in_remove_and_reverse_bool:NnN \l__cooking_units_number_tmpb_tl {
    \bool_lazy_or:nnTF
      { \l__cooking_units_fraction_in_input_bool }
      { \l__cooking_units_special_sign_bool }
      {
        \__cooking_units_formatiere_fractions:nnn
          { \l__cooking_units_mixed_fraction_tl }
          { \l__cooking_units_number_tmpa_tl }
          { \l__cooking_units_number_tmpb_tl }
      }{
        \__cooking_units_tl_set_fp_and_eval:Nn \l__cooking_units_number_tmpa_tl
          {
            \bool_if:NTF \l__cooking_units_minus_bool
              { - \l__cooking_units_mixed_fraction_tl - }
              { \l__cooking_units_mixed_fraction_tl + }
            \l__cooking_units_number_tmpa_tl / \l__cooking_units_number_tmpb_tl
          }
        \__cooking_units_process_and_print_number_in_input:N \l__cooking_units_number_tmpa_t
      }
  }
```

(*End definition for* `\__cooking_units_parse_mixed_fraction_in_input:www`.)

`\__cooking_units_parse_input_and_safe_in:nN` Used to check the input. `\l_@@_tmpa_tl` is cleared at the beginning. At first it checks if the first token is a sign or not. The parsed input is stored into `\l_@@_tmpa_tl`.

```
\cs_new:Npn \__cooking_units_parse_input_and_safe_in:nN #1 #2
  {
    \tl_clear:N \l__cooking_units_tmpa_tl
    \bool_set_false:N \l__cooking_units_decimal_in_input_bool
    \__cooking_units_parse_vorzeichen_and_rest:Nw #1 \q_stop
    \bool_if:NTF \l__cooking_units_error_bool
      { \tl_set:Nn #2 {#1} }
      { \tl_set_eq:NN #2 \l__cooking_units_tmpa_tl }
  }
```

(*End definition for* `\__cooking_units_parse_input_and_safe_in:nN`.)

`__cooking_units_parse_vorzeichen_and_rest:Nw` This function seperates the input into two parts with the first part being the first token. This token is checked whetever or not it is a sign. If it is a sign it is put into `\l_@@_tmpa_tl`, if not it is checked normally by `\@@_parse_input_for_safety_aux:N`. #2 (the other tokens) is then given to `\tl_map_function:nN`. This also works if the input only conists of one token with #2 being empty (hopefully).

```
\cs_new:Npn \__cooking_units_parse_vorzeichen_and_rest:Nw #1#2 \q_stop
  {
    \tl_if_in:NnTF \l__cooking_units_input_value_signs_tl {#1}
      { \tl_put_right:Nn \l__cooking_units_tmpa_tl {#1} }
      {
        \tl_map_function:nN {#1} \__cooking_units_parse_input_for_safety_aux:N
      }
    \bool_if:NF \l__cooking_units_error_bool
      { \tl_map_function:nN {#2} \__cooking_units_parse_input_for_safety_aux:N }
  }
```

(*End definition for* `\__cooking_units_parse_vorzeichen_and_rest:Nw`.)

`__cooking_units_parse_input_for_safety_aux:N`

Parses the input. It also is more or less copied from `siunitx`. Checks if the input consists only of numbers, one decimal-sign and the allowed special input. If an allowed token is found the boolean `\l_@@_special_sign_bool` is set to true.

```
\cs_new:Npn \__cooking_units_parse_input_for_safety_aux:N #1
  {
    \tl_if_in:NnTF \l__cooking_units_input_digits_tl {#1}
      { \tl_put_right:Nn \l__cooking_units_tmpa_tl {#1} }
      {
        \tl_if_in:NnTF \l__cooking_units_input_decimal_mark_tl {#1}
          {
            \bool_if:NT \l__cooking_units_decimal_in_input_bool
              {
                \msg_error:nn { cooking-units }
                  { Second-decimal-sign-not-allowed }
              }
            \bool_set_true:N \l__cooking_units_decimal_in_input_bool
            \tl_put_right:Nn \l__cooking_units_tmpa_tl { . }
          }{
            \tl_if_in:NnTF \l__cooking_units_input_allowed_special_signs_tl {#1}
              {
                \bool_set_true:N \l__cooking_units_special_sign_bool
                \tl_put_right:Nn \l__cooking_units_tmpa_tl {#1}
              }{
                \bool_if:NTF \l__cooking_units_range_in_input_bool
                  {
                    \tl_if_in:nnTF { / _ } {#1}
                      { \msg_error:nnn { cooking-units } { fraction-not-allowed-with-range }
                      { \msg_error:nnn { cooking-units } { Token-not-allowed } {#1} }
                  }{
                    \tl_if_in:nnTF { _ } {#1}
                      { \msg_error:nnn { cooking-units } { missing-slash } {#1} }
                      { \msg_error:nnn { cooking-units } { Token-not-allowed } {#1} }
                   }
                \bool_set_true:N \l__cooking_units_error_bool
                \tl_map_break:
              }
          }
      }
  }
```

(*End definition for* `\__cooking_units_parse_input_for_safety_aux:N`.)

C.3 Formatiere & Calculiere

__cooking_units_calculate_and_store_in:N

After parsing the input, it is given to this function. If a not allowed token is found (and \l_@@_error_bool is set to true) it just prints the input. Otherwise it checks if a allowed sign was found. If so it is just stored inside \l_@@_tmpa_tl, otherwise it is calulcated and stored in \l_@@_tmpa_tl. The input #1 can be either a token or a number.

```
\cs_new:Npn \__cooking_units_calculate_and_store_in:N #1
  {
    \bool_if:NF \l__cooking_units_error_bool
      {
        \bool_if:NTF \l__cooking_units_special_sign_bool
          {
            \tl_set_eq:NN \l__cooking_units_tmpa_tl #1
            \msg_warning:nnx { cooking-units } { amount-not-known } \l__cooking_units_tmpa_t
          }{
            \bool_lazy_and:nnTF
              { \l__cooking_units_using_cutext_bool } { ! \l__cooking_units_cutext_change_un
              { \tl_set_eq:NN \l__cooking_units_tmpa_tl #1 }
              { \__cooking_units_calculate_input_and_store_in:nN {#1} \l__cooking_units_tmpa
            \bool_if:NT \l__cooking_units_calc_persons_bool
              { \__cooking_units_calc_for_number_of_persons_and_store_in:NN \l__cooking_unit
            \__cooking_units_round_calculated_input:NV \l__cooking_units_tmpa_tl \l__cooking
            \__cooking_units_check_temperature_limit:N \l__cooking_units_tmpa_tl
          }
        \tl_set_eq:NN #1 \l__cooking_units_tmpa_tl
      }
  }
```

(*End definition for* __cooking_units_calculate_and_store_in:N.)

ooking_units_calculate_input_and_store_in:nN

Well ... this function calculates the input. First checks if the wanted conversion contains a \l_@@_tmpa_fp (ergo a #1 in the key definition). If true the conversion token (which already has \l_@@_tmpa_fp in its input) is executed, else the input number is multiplicated with the conversion token.

```
\cs_new:Npn \__cooking_units_calculate_input_and_store_in:nN #1#2
  {
    \fp_set:Nn \l__cooking_units_tmpa_fp {#1}
    \tl_if_in:cnTF { l__cooking_units_tmpa_ \l__cooking_units_given_unit_tl  _ tl } { \l__co
      {
        \__cooking_units_tl_set_fp_and_eval:Nc #2 { l__cooking_units_tmpa_ \l__cooking_units
      }{
        \__cooking_units_tl_set_fp_and_eval:Nn #2
          { \l__cooking_units_tmpa_fp * \tl_use:c { l__cooking_units_tmpa_ \l__cooking_units
      }
  }
```

(*End definition for* __cooking_units_calculate_input_and_store_in:nN.)

```
\cs_new:Npn \__cooking_units_calc_for_number_of_persons_and_store_in:NN #1#2
  {
    \int_compare:nNnF
      { \l__cooking_units_calc_for_number_of_persons_int }
      =
      { \l__cooking_units_number_of_persons_tmpa_int }
      {
```

```
        \__cooking_units_tl_set_fp_and_eval:Nn #1
          {
            \l__cooking_units_calc_for_number_of_persons_int /
            \l__cooking_units_number_of_persons_tmpa_int *
            #2
          }
      }
  }
```

__cooking_units_check_number_for_rounding:n
__cooking_units_check_number_after_dot_aux:w

Getting the number after the decimal point. If it doesn't exist the boolean is set to false, otherwise it checks how many tokens are within the number after the decimal point. If the number of tokens is greater than significant_figures_plus_one the bool is set to true (which tells the package later that the number should be rounded). This should be safe as the input was already checked.

```
\cs_new:Npn \__cooking_units_check_number_for_rounding:n #1
  {
    \bool_set_false:N \l__cooking_units_round_decimal_part_bool
    \__cooking_units_check_number_after_dot_aux:w #1 . \q_recursion_tail .
    \q_recursion_stop
  }
\cs_new:Npn \__cooking_units_check_number_after_dot_aux:w #1. #2 .
  {
    \quark_if_recursion_tail_stop:n {#2}
    \bool_if:NTF \l__cooking_units_round_to_int_bool
      { \int_zero:N \l_tmpa_int }
      { \int_set_eq:NN \l_tmpa_int \l__cooking_units_significant_figures_plus_one_int }
    \int_compare:nNnF
      { \tl_count:n {#2} } < { \l_tmpa_int }
      { \bool_set_true:N \l__cooking_units_round_decimal_part_bool }
    \use_none_delimit_by_q_recursion_stop:w
  }
```

(*End definition for* `\__cooking_units_check_number_for_rounding:n` *and* `\__cooking_units_check_number_after_dot_aux:w`.)

__cooking_units_round_calculated_input:NN
__cooking_units_round_calculated_input:NV

After calculating the numbers are rounded (if needed, safed inside `\l_tmpa_bool`). We test at first if the input needs to be rounded by comparing the number of tokens after the decimal with the number of significant figures plus one. Afterwards the input is expanded and stored in #1. If rounding needs to be done it happens now.

```
\cs_new:Npn \__cooking_units_round_calculated_input:NN #1#2
  {
    \__cooking_units_check_number_for_rounding:n {#2}
    \tl_set:Nx #1
      {
        \bool_if:NTF \l__cooking_units_round_decimal_part_bool
          { \fp_eval:n { round ( \__cooking_units_rounding_function:n {#2} ) } }
          {#2}
      }
  }
\cs_generate_variant:Nn \__cooking_units_round_calculated_input:NN { NV }
```

(*End definition for* `\__cooking_units_round_calculated_input:NN` *and* `\__cooking_units_round_calculated_input:NV`.)

__cooking_units_print_input:N
__cooking_units_print_correct_unit:
__cooking_units_post_process_input:NN
__cooking_units_pre_process_input:NN

Wrapper macro for printing the (not)calculated output. Note that if no calculation happens in \cutext (and \Cutext) \l__cooking_units_option_unit_tl is set to \l__cooking_units_given_unit_tl (the unit given in the second argument of \cutext or \Cutext) by default.

```
\cs_new_protected:Npn \__cooking_units_print_input:N #1 { }
\cs_new_protected:Npn \__cooking_units_print_correct_unit: { }
\cs_new_protected:Npn \__cooking_units_do_not_process_input:NN #1#2 { }
\cs_new_eq:NN \__cooking_units_pre_process_input:NN \__cooking_units_do_not_process_input:NN
\cs_new_eq:NN \__cooking_units_post_process_input:NN \__cooking_units_do_not_process_input:N
```

(*End definition for* `\__cooking_units_print_input:N` *and others.*)

oking_units_set_process_and_print_for_cunum:
king_units_set_process_and_print_for_cutext:
ooking_units_set_process_and_print_for_cuam:

It is handy to set the "print" and "process" command all in one go. It's also less to use messy.

```
\cs_new_protected:Npn \__cooking_units_set_process_and_print_for_cunum:
  {
    \cs_set_eq:NN \__cooking_units_print_input:N \__cooking_units_print_numerical_input:N
    \cs_set_eq:NN \__cooking_units_print_correct_unit: \__cooking_units_cunum_print_correct_
    \cs_set_eq:NN \__cooking_units_pre_process_input:NN \__cooking_units_do_not_process_inpu
    \cs_set_eq:NN \__cooking_units_post_process_input:NN \__cooking_units_do_not_process_inp
  }
\cs_new_protected:Npn \__cooking_units_set_process_and_print_for_cutext:
  {
    \cs_set_eq:NN \__cooking_units_print_input:N \__cooking_units_cutext_print_input:N
    \cs_set_eq:NN \__cooking_units_print_correct_unit: \__cooking_units_cutext_print_correct
    \cs_set_eq:NN \__cooking_units_pre_process_input:NN \__cooking_units_cutext_pre_process_
    \cs_set_eq:NN \__cooking_units_post_process_input:NN \__cooking_units_cutext_post_proces
  }
\cs_new_protected:Npn \__cooking_units_set_process_and_print_for_cuam:
  {
    \cs_set_eq:NN \__cooking_units_print_input:N \__cooking_units_cuam_print_numerical_input
    \cs_set_eq:NN \__cooking_units_print_correct_unit: \__cooking_units_cuam_print_correct_u
    \cs_set_eq:NN \__cooking_units_pre_process_input:NN \__cooking_units_do_not_process_inpu
    \cs_set_eq:NN \__cooking_units_post_process_input:NN \__cooking_units_cuam_post_process_
  }
```

(*End definition for* `\__cooking_units_set_process_and_print_for_cunum:`, `\__cooking_units_set_process_and_print_for_cutext:`, *and* `\__cooking_units_set_process_and_print_for_cuam:`.)

__cooking_units_print_numerical_input:N

Prints the nummerical output (if it is not a fraction).

Changed \l_@@_tmpa_tl by \l_@@_translation_tmpa_tl, it's better that way.

```
\cs_new_protected:Npn \__cooking_units_print_numerical_input:N #1
  {
    \tl_if_in:NnT #1 { . }
      {
        \__cooking_units_translate_let:Nxx \l__cooking_units_translation_tmpa_tl
          { decimal-mark } \c__cooking_units_postfix_unitname_tl
        \tl_replace_once:Nnn #1 { . } { \l__cooking_units_translation_tmpa_tl }
      }
    \tl_if_in:NnT #1 { - }
      { \tl_replace_once:NnV #1 { - } \c__cooking_units_minus_tl }
    #1
  }
```

(*End definition for* `\__cooking_units_print_numerical_input:N`.)

`\__cooking_units_formatiere_fractions:nnn` The name of this function is the name of the game: It prints fractions. Furthermore it sets the boolean `fraction_in_input_bool` to true so that the correct unit is printed. Instead of `\kern` I used the `\hbox_to_wd:nn` command (I don't have a better idea).

Since v.1.10(alpha) it also prints the minus sign (hopefully correct).

```
\cs_new:Npn \__cooking_units_formatiere_fractions:nnn #1#2#3
  {
    \bool_if:NT \l__cooking_units_minus_bool { \c__cooking_units_minus_tl }
    \tl_if_empty:nF {#1}
      {
        #1
        \hbox_to_wd:nn { \l__cooking_units_mixed_frac_dim } { }
      }
    \__cooking_units_frac:nn {#2} {#3}
  }
```

(*End definition for* `\__cooking_units_formatiere_fractions:nnn`.)

`\__cooking_units_check_temperature_limit:n` If `check_temperature_bool` is set to true it now checks if the value is below the absolute temperature. `\clist_if_in:nVT` is needed due to `Re`, a `\tl_if_in:nVT` would also check for `R` and `e` instead of only `Re`.

```
\cs_new:Npn \__cooking_units_check_temperature_limit:N #1
  {
    \bool_if:NT \l__cooking_units_check_temperature_bool
      {
        \seq_if_in:NVT \l__cooking_units_temperatures_to_check_seq \l__cooking_units_option_
          {
            \fp_compare:cNnT
              { c__cooking_units_ \l__cooking_units_option_unit_tl _min_fp } > {#1}
              {
                \msg_error:nnxx { cooking-units }
                  { Temperature-too-low }
                  { #1 \space \l__cooking_units_option_unit_tl }
                  {
                    \fp_use:c { c__cooking_units_ \l__cooking_units_option_unit_tl _min_fp }
                    \space \l__cooking_units_option_unit_tl
                  }
              }
          }
      }
  }
```

(*End definition for* `\__cooking_units_check_temperature_limit:n`.)

```
\cs_new_nopar:Npn \__cooking_units_grab_arrows_for_safety_do_afterwards:nN #1#2
  {
    \str_if_eq:nnTF {#2} { > }
      {
        \tl_put_right:Nx \l__cooking_units_tmpa_tl { \tl_to_str:N > }
        \exp_last_unbraced:NV #1 \l__cooking_units_tmpa_tl
      }{
        \tl_put_right:Nn \l__cooking_units_tmpa_tl {#2}
        \__cooking_units_grab_arrows_for_safety_do_afterwards:nN {#1}
      }
```

```
  }
\cs_new_nopar:Npn \__cooking_units_if_arrow_grab_until_close_do:nnTF #1#2#3#4
  {
    \str_if_eq:nnTF {#1} { < }
      {
        \tl_if_in:NoT \l__cooking_units_input_allowed_special_signs_tl { < }
          { \msg_error:nn {cooking-units} { <-not-allowed-as-special-sign } }
        \tl_clear:N \l__cooking_units_tmpa_tl
        \tl_put_right:Nx \l__cooking_units_tmpa_tl { \tl_to_str:N < }
        \__cooking_units_grab_arrows_for_safety_do_afterwards:nN
          {#3} #2
      }
      {#4}
  }
```

C.4 \cunum

\cunum The main command of this package.

```
\NewDocumentCommand \cunum { d<> O{} m O{} m }
  {
    \__cooking_units_if_arrow_grab_until_close_do:nnTF {#3} {#5}
      { \cunum }
      {
        \group_begin:
        \__cooking_units_cunum:nnnnn {#1} {#2} {#3} {#4} {#5}
        \group_end:
      }
  }
```

(*End definition for* \cunum. *This function is documented on page* ??.)

__cooking_units_cunum

```
\cs_new:Npn \__cooking_units_cunum:nnnnn #1#2#3#4#5
  {
    \__cooking_units_cunum_initialise:nnnnn {#1} {#2} {#3} {#4} {#5}
    \__cooking_units_cunum_parse_numerical_input:n {#3}
    \__cooking_units_print_correct_unit:
  }
```

(*End definition for* __cooking_units_cunum.)

__cooking_units_cunum_initialise:nnnn Lots of things to do in this function. It checks if the unit is defined or not, sets the keys for converting the unit and sets the optional unit (the unit our input is converted to) accordingly, sets the options again (to get the option-specific options) and ... yeah.

First parses the change of unit ('set groups'), afterwards set the predefined options for the new unit. Afterwards set the 'normal' options given by the optional argument.

```
\cs_new_protected:Npn \__cooking_units_cunum_initialise:nnnnn #1#2#3#4#5
  {
    \__cooking_units_set_process_and_print_for_cunum:
    \tl_set:Nn \l__cooking_units_phantom_tl {#4}
    \__cooking_units_initialise_default:nnn {#1} {#2} {#5}
    \__cooking_units_initialise_unit_change:nnn {#1} {#2} {#5}
    \__cooking_units_initialise_after_unit_change:nn {#1} {#2}
  }
```

(*End definition for* `\__cooking_units_cunum_initialise:nnnn`.)

`\__cooking_units_initialise_default:n` Function shared by all initialization functions:

```
\cs_new_protected:Npn \__cooking_units_initialise_default:nnn #1#2#3
  {
    \tl_set:Nn \l__cooking_units_given_unit_tl {#3}
    \__cooking_units_error_if_unit_not_defined:V \l__cooking_units_given_unit_tl
    \bool_set_false:N \l__cooking_units_special_sign_bool
    \bool_set_false:N \l__cooking_units_error_bool
    \bool_set_false:N \l__cooking_units_range_in_input_bool
    \bool_set_false:N \l__cooking_units_fraction_in_input_bool
  }
```

Some units have options added to them. To get those options it is first needed to know which unit will be used at all. Therefore units will be changed first and afterwards the other options are processed.

```
\cs_new_protected:Npn \__cooking_units_initialise_unit_change:nnn #1#2#3
  {
    \tl_if_empty:nF {#2}
      {
        \keys_set_groups:nnn { cooking-units } { change-unit } {#2}
      }
    \bool_if:NT \l__cooking_units_convert_to_eV_bool { \__cooking_units_convert_to_eV: }
    \prop_get:NVNF \l__cooking_units_change_unit_prop \l__cooking_units_given_unit_tl \l__co
      {
        \tl_set_eq:NN \l__cooking_units_option_unit_tl \l__cooking_units_given_unit_tl
      }
  }
\cs_new_protected:Npn \__cooking_units_initialise_after_unit_change:nn #1#2
  {
    \IfNoValueF {#1}
      { \__cooking_units_reference_label_and_persons:n {#1} }
    \clist_if_empty:cF { l__cooking_units_predefined_option_ \l__cooking_units_option_unit_t
      {
        \keys_set_filter:nnv
          { cooking-units }
          { change-unit }
          { l__cooking_units_predefined_option_ \l__cooking_units_option_unit_tl _clist }
      }
    \tl_if_empty:nF {#2}
      { \keys_set_filter:nnn { cooking-units } { change-unit } {#2} }
    \bool_lazy_and:nnTF
      { \l__cooking_units_calc_because_ref_was_given_bool } { \l__cooking_units_calc_for_per
      { \bool_set_true:N \l__cooking_units_calc_persons_bool }
      { \bool_set_false:N \l__cooking_units_calc_persons_bool }
  }
```

(*End definition for* `\__cooking_units_initialise_default:n`.)

```
\cs_new:Npn \__cooking_units_cunum_parse_numerical_input:n #1
  {
    \bool_if:NTF \l__cooking_units_parse_input_bool
      { \__cooking_units_parse_and_evaluate_input:n {#1} }
      { \__cooking_units_do_not_parse:n {#1} }
  }
```

__cooking_units_do_not_parse:n

If the input shoudn't be parsed this function is used to print the input without error messages (mostly concerning the _). Spaces are still ignored.

```
\cs_new:Npn \__cooking_units_do_not_parse:n #1
  {
    \tl_set_rescan:Nnn \l__cooking_units_number_tmpa_tl
      {
        \char_set_catcode_letter:N \_ %
        \char_set_catcode_ignore:N \ %
      } {#1}
    \l__cooking_units_number_tmpa_tl
  }
```

(*End definition for* `\__cooking_units_do_not_parse:n`.)

__cooking_units_parse_and_evaluate_input:n

Rescans the input to get rid of spaces and to make _ and ? inactive (french with babel makes ? active and changes the definition of it). `--` is replaced by `\q_@@_range` and the input is parsed (if not empty). Afterwards the units are printed.

```
\cs_new:Npn \__cooking_units_parse_and_evaluate_input:n #1
  {
    \tl_set_rescan:Nnn \l__cooking_units_tmpa_tl
      {
        \char_set_catcode_letter:N \_ %
        \char_set_catcode_ignore:N \ %
        \char_set_catcode_other:N ? %
      } {#1}
    \tl_if_empty:NF \l__cooking_units_tmpa_tl
      {
        \tl_if_in:NVT \l__cooking_units_tmpa_tl \l__cooking_units_input_range_sign_tl
          {
            \tl_replace_once:NVn \l__cooking_units_tmpa_tl \l__cooking_units_input_range_sig
            \bool_set_true:N \l__cooking_units_range_in_input_bool
          }
        \__cooking_units_parse_input:V \l__cooking_units_tmpa_tl
      }
  }
```

(*End definition for* `\__cooking_units_parse_and_evaluate_input:n`.)

__cooking_units_cunum_print_correct_unit:

The invisible space is added by a `\phantom`, afterwards `value_unit_space_tl` is used (which is set to `\thinspace` by default) and if either special signs or fractions are parsed the input-unit is printed else the converted unit is.

```
\cs_new:Npn \__cooking_units_cunum_print_correct_unit:
  {
    \tl_if_empty:NF \l__cooking_units_phantom_tl { \phantom { \l__cooking_units_phantom_tl }
    \tl_use:N \l__cooking_units_value_unit_space_tl
    \bool_lazy_any:nTF
      {
        { \l__cooking_units_fraction_in_input_bool }
        { \l__cooking_units_special_sign_bool }
        { ! \l__cooking_units_parse_input_bool }
      }
      { \__cooking_units_translate:xx \l__cooking_units_given_unit_tl \c__cooking_units_post
      { \__cooking_units_translate:xx \l__cooking_units_option_unit_tl \c__cooking_units_pos
  }
```

(*End definition for* `\__cooking_units_cunum_print_correct_unit:`.)

D cutext & Cutext

A quite primitive implentation of `cutext-to-cunum`, but sufficient for now.

```
\NewDocumentCommand \cutext { d<> O{} m m }
  {
    \__cooking_units_if_arrow_grab_until_close_do:nnTF {#3} {#4}
      { \cutext }
      {
        \group_begin:
        \__cooking_units_cutext_initialise:nnn {#1} {#2} {#4}
        \bool_set_false:N \l__cooking_units_cutext_uppercase_word_bool
        \__cooking_units_cutext_do:nnnn {#1} {#2} {#3} {#4}
        \group_end:
      }
  }
\NewDocumentCommand \Cutext { d<> O{} m m }
  {
    \__cooking_units_if_arrow_grab_until_close_do:nnTF {#3} {#4}
      { \Cutext }
      {
        \group_begin:
        \__cooking_units_cutext_initialise:nnn {#1} {#2} {#4}
        \bool_set_true:N \l__cooking_units_cutext_uppercase_word_bool
        \__cooking_units_cutext_do:nnnn {#1} {#2} {#3} {#4}
        \group_end:
      }
  }
\cs_new_protected:Npn \__cooking_units_cutext_initialise:nnn #1#2#3
  {
    \bool_set_true:N \l__cooking_units_using_cutext_bool
    \__cooking_units_set_process_and_print_for_cutext:
    \__cooking_units_initialise_default:nnn {#1} {#2} {#3}
    \bool_if:NTF \l__cooking_units_cutext_change_unit_bool
      { \__cooking_units_initialise_unit_change:nnn {#1} {#2} {#3} }
      { \tl_set_eq:NN \l__cooking_units_option_unit_tl \l__cooking_units_given_unit_tl }
    \__cooking_units_initialise_after_unit_change:nn {#1} {#2}
  }
```

Besser benennen.

```
\cs_new:Npn \__cooking_units_cutext_do:nnnn #1#2#3#4
  {
    \bool_if:NTF \l__cooking_units_cutext_to_cunum_bool
      { \cunum <#1> [#2] {#3} {#4} }
      {
        \__cooking_units_cutext:nnnn {#1} {#2} {#3} {#4}
      }
  }
% #1: label, #2: Options, #3: Values, #4: unit
\cs_new:Npn \__cooking_units_cutext:nnnn #1#2#3#4
  {
```

```
    \bool_if:NTF \l__cooking_units_parse_input_bool
      {
        \bool_if:NTF \l__cooking_units_cutext_old_bool
          {
            \__cooking_units_old_cutext_default:nnn {#2} {#3} {#4}
            \tl_set_eq:NN \l__cooking_units_option_unit_tl \l__cooking_units_given_unit_tl
            \__cooking_units_cutext_print_correct_unitname:
          }{
            \__cooking_units_parse_and_evaluate_input:n {#3}
            \__cooking_units_print_correct_unit:
          }
      }{
        \__cooking_units_do_not_parse:n {#3}
        \tl_set_eq:NN \l__cooking_units_option_unit_tl \l__cooking_units_given_unit_tl
        \__cooking_units_cutext_print_correct_unitname:
      }
  }
```

_and_parse_and_smaller_then_print_numerals:N

```
\prg_new_conditional:Npnn \__cooking_units_cutext_if_numeral_is_int_and_parse_and_smaller_th
  {
    \bool_lazy_and:nnTF
      { \g__cooking_units_opt_numeral_bool }
      { \l__cooking_units_local_numeral_bool }
      {
        \__cooking_units_if_parse_and_integer:VTF #1
          {
            \int_compare:nNnTF {#1} < { \l__cooking_units_print_numerals_below_int }
              { \prg_return_true: }
              { \prg_return_false: }
          }{ \prg_return_false: }
      }{ \prg_return_false: }
  }
```

(*End definition for* `\__cooking_units_cutext_if_numeral_is_int_and_parse_and_smaller_then_print_numerals:N`.)

__cooking_units_cutext_print_input:Nn
__cooking_units_cutext_print_input:NV

```
\cs_new_protected:Npn \__cooking_units_cutext_print_input:Nn #1#2
  {
    \__cooking_units_cutext_if_numeral_is_int_and_parse_and_smaller_then_print_numerals:NTF
      {
        \__cooking_units_int_if_equal_one:nTF {#1}
          {
            \__cooking_units_translate_one_to_and_check_existance:Nx \l__cooking_units_trans
            \bool_if:NTF \l__cooking_units_cutext_uppercase_word_bool
              {
                \exp_args:Nx \tl_upper_case:n { \tl_head:V \l__cooking_units_translation_tmp
                \tl_tail:V \l__cooking_units_translation_tmpa_tl
              }
              { \l__cooking_units_translation_tmpa_tl }
          }{
            \bool_if:NTF \l__cooking_units_cutext_uppercase_word_bool
              { \__cooking_units_print_Numeral:n {#1} }
```

```
            { \__cooking_units_print_numeral:n {#1} }
        }
    }
    { \__cooking_units_print_numerical_input:N #1 }
  }
\cs_generate_variant:Nn \__cooking_units_cutext_print_input:Nn { NV }
\cs_new_protected:Npn \__cooking_units_cutext_print_input:N #1
  { \__cooking_units_cutext_print_input:NV #1 \l__cooking_units_option_unit_tl }
```

(*End definition for* `\__cooking_units_cutext_print_input:Nn`.)

```
\cs_new:Npn \__cooking_units_cutext_print_correct_unitname:
  {
    \l__cooking_units_cutext_space_tl
    \bool_lazy_any:nTF
      {
        { \l__cooking_units_fraction_in_input_bool }
        { \l__cooking_units_special_sign_bool }
        { ! \l__cooking_units_parse_input_bool }
      }
      { \__cooking_units_translate:xx \l__cooking_units_given_unit_tl \c__cooking_units_post
      {
        \__cooking_units_fp_if_equal_one:nTF { \l__cooking_units_cutext_last_value_tl }
          { \__cooking_units_translate:xx \l__cooking_units_option_unit_tl \c__cooking_units
          { \__cooking_units_translate:xx \l__cooking_units_option_unit_tl \c__cooking_units
      }
  }
\cs_new:Npn \__cooking_units_cutext_pre_process_input:NN #1#2
  {
    \__cooking_units_cutext_check_unitname_consequences:NN #1#2
  }
\cs_new:Npn \__cooking_units_cutext_post_process_input:NN #1#2
  {
    \bool_if:NTF \l__cooking_units_range_in_input_bool
      {
        \tl_set_eq:NN \l__cooking_units_cutext_last_value_tl #2
        \bool_if:NT \g__cooking_units_opt_numeral_bool
          {
            \bool_lazy_and:nnF
              { \fp_compare_p:nNn {#1} < { \l__cooking_units_print_numerals_below_int } }
              { \fp_compare_p:nNn {#2} < { \l__cooking_units_print_numerals_below_int } }
              { \bool_set_false:N \l__cooking_units_local_numeral_bool }
          }
      }
      { \tl_set_eq:NN \l__cooking_units_cutext_last_value_tl #1 }
  }
```

It doesn't matter if I check if the singular or the plural translation exists, as the plural one exists if the singular exists and vice versa.

```
\cs_new:Npn \__cooking_units_cutext_check_unitname_consequences:NN #1#2
  {
    \bool_lazy_any:nTF
      {
        { \l__cooking_units_fraction_in_input_bool }
        { \l__cooking_units_special_sign_bool }
```

```
        { ! \l__cooking_units_parse_input_bool }
      }
      { \tl_set_eq:NN \l__cooking_units_tmpb_tl \l__cooking_units_given_unit_tl }
      { \tl_set_eq:NN \l__cooking_units_tmpb_tl \l__cooking_units_option_unit_tl }
    \__cooking_units_unitname_get:NxF \l__cooking_units_tmpa_tl \l__cooking_units_tmpb_tl
      {
        \msg_warning:nnx
          { cooking-units }
          { cutext-no-translation-available }
          \l__cooking_units_tmpb_tl
        \bool_set_false:N \l__cooking_units_using_cutext_bool
        \__cooking_units_set_process_and_print_for_cunum:
      }
  }
```

__cooking_units_old_cutext_default:nnn

```
\cs_new:Npn \__cooking_units_old_cutext_default:nnn #1#2#3
  {
    \bool_if:NTF \l__cooking_units_parse_input_bool
      {
        \tl_set:Nn \l__cooking_units_cutext_last_value_tl {#2}
        \tl_if_in:NVTF \l__cooking_units_cutext_last_value_tl \l__cooking_units_input_range_
          {
            \tl_replace_once:NVn \l__cooking_units_cutext_last_value_tl
              \l__cooking_units_input_range_sign_tl { \q__cooking_units_range }
            \__cooking_units_old_cutext_parse_range:Vn \l__cooking_units_cutext_last_value_t
          }{
            \__cooking_units_cutext_print_input:Nn \l__cooking_units_cutext_last_value_tl {#
          }
      }
      {#2}
  }
```

(*End definition for* `\__cooking_units_old_cutext_default:nnn`.)

```
\cs_new:Npn \__cooking_units_old_cutext_parse_range:Nn #1 #2
  {
    \__cooking_units_old_cutext_parse_range_aux:nww {#2} #1 \q_stop
  }
\cs_generate_variant:Nn \__cooking_units_old_cutext_parse_range:Nn { V }
\cs_new:Npn \__cooking_units_old_cutext_parse_range_aux:nww #1 #2 \q__cooking_units_range #3
  {
    \tl_set:Nn \l__cooking_units_tmpa_tl {#2}
    \tl_set:Nn \l__cooking_units_cutext_last_value_tl {#3}
    \__cooking_units_cutext_print_input:Nn \l__cooking_units_tmpa_tl {#1}
    \tl_use:N \l__cooking_units_cutext_range_sign_tl
    \__cooking_units_cutext_print_input:Nn \l__cooking_units_cutext_last_value_tl {#1}
  }
```

E cuam

```
\tl_const:Nn \c__cooking_units_cuam_marker_tl { __cooking_units_cunum }
\tl_new:c { l__cooking_units_tmpa_ \c__cooking_units_cuam_marker_tl  _ tl }
\tl_set:cn { l__cooking_units_tmpa_ \c__cooking_units_cuam_marker_tl  _ tl } { \c_one_fp }
```

```
\clist_new:c { l__cooking_units_predefined_option_ \c__cooking_units_cuam_marker_tl _clist }
```

Replaces and extends .

\cuam

```
\NewDocumentCommand \cuam { d<> O{} m }
  {
    \__cooking_units_if_arrow_grab_until_close_do:nnTF {#3} { }
      { \cuam }
      {
        \group_begin:
        \__cooking_units_cuam_initialise:nn {#1} {#2}
        \__cooking_units_cuam:n {#3}
        \group_end:
      }
  }
```

(*End definition for* \cuam. *This function is documented on page* ??.)

```
\cs_new:Npn \__cooking_units_cuam:n #1
  {
    \bool_if:NTF \l__cooking_units_parse_input_bool
      {
        \bool_if:NTF \l__cooking_units_cuam_old_bool
          { \__cooking_units_cuam_old:n {#1} }
          {
            \__cooking_units_parse_and_evaluate_input:n {#1}
            \__cooking_units_print_correct_unit:
          }
      }
      { \__cooking_units_do_not_parse:n {#1} }
  }
\cs_new_protected:Npn \__cooking_units_cuam_initialise:nn #1#2
  {
    \__cooking_units_set_process_and_print_for_cuam:
    \tl_set_eq:NN \l__cooking_units_given_unit_tl \c__cooking_units_cuam_marker_tl
    \tl_set_eq:NN \l__cooking_units_option_unit_tl \c__cooking_units_cuam_marker_tl
    \__cooking_units_initialise_after_unit_change:nn {#1} {#2}
  }
```

I should redo this command ... later

```
\cs_new_protected:Npn \__cooking_units_cuam_post_process_input:NN #1#2
  {
    \__cooking_units_if_integer:VTF #1
      { \bool_set_true:N \l__cooking_units_tmpa_bool }
      { \bool_set_false:N \l__cooking_units_tmpa_bool }
    \bool_lazy_and:nnT
      { \l__cooking_units_range_in_input_bool } { \l__cooking_units_tmpa_bool }
      {
        \__cooking_units_if_integer:VTF #2
          { \bool_set_true:N \l__cooking_units_tmpa_bool }
          { \bool_set_false:N \l__cooking_units_tmpa_bool }
      }
    \bool_lazy_and:nnT
      { \l__cooking_units_use_phrases_bool } { \l__cooking_units_tmpa_bool }
      {
```

```
        \__cooking_units_if_phrase_list_exists:NT \l__cooking_units_phrase_prop
          {
            \__cooking_units_translate_let:Nxx \l__cooking_units_phrase_numbers_clist
              { phrase-list-list } \c__cooking_units_postfix_phrase_tl
            \__cooking_units_cuam_process_input_aux:NNN #1 \l__cooking_units_tmpa_int \l__co
            \bool_if:NT \l__cooking_units_check_if_phrase_used_bool
              {
                \bool_if:NTF \l__cooking_units_range_in_input_bool
                  {
                    \__cooking_units_cuam_process_input_aux:NNN #2 \l__cooking_units_tmpb_in
                    \bool_lazy_and:nnTF
                      { \l__cooking_units_check_if_phrase_used_bool }
                      { \tl_if_eq_p:NN \l__cooking_units_phrase_number_tl \l__cooking_units_
                      { \bool_set_true:N \l__cooking_units_check_if_phrase_used_bool }
                      { \bool_set_false:N \l__cooking_units_check_if_phrase_used_bool }
                    \bool_if:NT \l__cooking_units_check_if_phrase_used_bool
                      {
                        \tl_set:NV #1 \l__cooking_units_tmpa_int
                        \tl_set:NV #2 \l__cooking_units_tmpb_int
                        \__cooking_units_cuam_get_phrase_name:NVN
                          \l__cooking_units_phrase_phrase_tl \l__cooking_units_phrase_number
                      }
                  }{
                      \tl_set:NV #1 \l__cooking_units_tmpa_int
                      \__cooking_units_cuam_get_phrase_name:NVN
                        \l__cooking_units_phrase_phrase_tl \l__cooking_units_phrase_number_t
                  }
              }
          }
      }
    \bool_lazy_and:nnT
      { \g__cooking_units_opt_numeral_bool } { \l__cooking_units_local_numeral_bool }
      {
        \bool_set_eq:NN \l__cooking_units_local_numeral_bool \l__cooking_units_tmpa_bool
        \bool_if:NT \l__cooking_units_local_numeral_bool
          {
            \bool_if:NTF \l__cooking_units_range_in_input_bool
               {
                 \bool_lazy_and:nnF
                   { \int_compare_p:nNn {#1} < { \l__cooking_units_print_numerals_below_int
                   { \int_compare_p:nNn {#2} < { \l__cooking_units_print_numerals_below_int
                   { \bool_set_false:N \l__cooking_units_local_numeral_bool }
               }{
                 \int_compare:nNnF {#1} < { \l__cooking_units_print_numerals_below_int }
                   { \bool_set_false:N \l__cooking_units_local_numeral_bool }
               }
          }
      }
  }
\prg_new_conditional:Npnn \__cooking_units_cuam_check_if_larger:nn #1#2 { F }
  {
    \int_compare:nNnTF {#1} > {#2}
      { \prg_return_true: }
      { \prg_return_false: }
```

```
  }
\cs_new:Npn \__cooking_units_cuam_process_input_aux:NNN #1#2#3
  {
    \bool_set_false:N \l__cooking_units_check_if_phrase_used_bool
    \clist_map_inline:Nn \l__cooking_units_phrase_numbers_clist
      {
        \__cooking_units_cuam_check_if_larger:nnF { \int_abs:n {##1} } {#1}
          {
            \int_compare:nNnTF {##1} < { \c_zero }
              { \int_set_eq:NN \l_tmpa_int \c_one }
              { \int_set:Nn \l_tmpa_int { \int_div_truncate:nn {#1} {##1} } }
            \int_compare:nNnT { \int_abs:n {##1} * \l_tmpa_int } = {#1}
              {
                \int_set_eq:NN #2 \l_tmpa_int
                \tl_set:Nn #3 {##1}
                \bool_set_true:N \l__cooking_units_check_if_phrase_used_bool
                \clist_map_break:
              }
          }
      }
  }
\cs_new:Npn \__cooking_units_cuam_get_phrase_name:NnN #1#2#3
  {
    \__cooking_units_int_if_equal_one:nTF {#3}
      { \prop_get:NnN \l__cooking_units_phrase_prop {#2} #1 }
      { \prop_get:NnN \l__cooking_units_phrase_prop { #2-pl } #1 }
  }
\cs_generate_variant:Nn \__cooking_units_cuam_get_phrase_name:NnN { NVN }
\cs_new_protected:Npn \__cooking_units_cuam_print_numerical_input:N #1
  {
    \bool_lazy_all:nTF
      {
        { \l__cooking_units_check_if_phrase_used_bool }
        { \g__cooking_units_opt_numeral_bool }
        { \l__cooking_units_local_numeral_bool }
      }{
        \__cooking_units_int_if_equal_one:nTF {#1}
          {
            \__cooking_units_translate_one_to_and_check_existance:Nx \l__cooking_units_trans
              { \l__cooking_units_phrase_number_tl -phrase-gender }
            \l__cooking_units_translation_tmpa_tl
          }
          { \exp_args:NV \__cooking_units_print_numeral:n #1 }
      }{ \__cooking_units_print_numerical_input:N #1 }
  }
\cs_new:Npn \__cooking_units_cuam_print_correct_unitphrase:
  {
    \bool_if:NT \l__cooking_units_check_if_phrase_used_bool
      {
        \l__cooking_units_cuphrase_space_tl
        \l__cooking_units_phrase_phrase_tl
      }
  }
```

__cooking_units_cuam_old:n

```
\cs_new:Npn \__cooking_units_cuam_old:n #1
  {
    \tl_set_rescan:Nnn \l__cooking_units_tmpa_tl
      {
        \char_set_catcode_letter:N \_ %
        \char_set_catcode_ignore:N\ %
      } {#1}
    \__cooking_units_cuam_old_parse:V \l__cooking_units_tmpa_tl
  }
```

(*End definition for* __cooking_units_cuam_old:n.)

```
\cs_new_protected:Npn \__cooking_units_cuam_old_parse:n #1
  {
    \tl_if_in:nVTF {#1} \l__cooking_units_input_range_sign_tl
      {
        \tl_set:Nn \l__cooking_units_tmpa_tl {#1}
        \tl_replace_once:NVn \l__cooking_units_tmpa_tl \l__cooking_units_input_range_sign_tl
        \__cooking_units_cuam_old_parse_range:V \l__cooking_units_tmpa_tl
      }{
        \tl_if_in:nnTF {#1} { / }
          {
            \tl_if_in:nnTF {#1} { _ }
              { \__cooking_units_cuam_old_parse_mixed_frac:www #1 \q_stop }
              { \__cooking_units_cuam_old_parse_frac:ww #1 \q_stop }
          }{
            \tl_if_in:nnTF {#1} { _ }
              { \msg_error:nnn { cooking-units } { missing-slash } {#1} }
              { \__cooking_units_cuam_old_parse_scale:n {#1} }
          }
      }
  }
\cs_generate_variant:Nn \__cooking_units_cuam_old_parse:n { V }
\cs_new:Npn \__cooking_units_cuam_old_parse_range:n #1
  {
    \__cooking_units_cuam_old_parse_range_aux:ww #1 \q_nil
  }
\cs_generate_variant:Nn \__cooking_units_cuam_old_parse_range:n { V }
\cs_new:Npn \__cooking_units_cuam_old_parse_range_aux:ww #1 \q__cooking_units_range #2 \q_ni
  {
    #1 \l__cooking_units_cunum_range_sign_tl #2
  }
\cs_new:Npn \__cooking_units_cuam_old_parse_scale:n #1 {#1}
\cs_new:Npn \__cooking_units_cuam_old_parse_frac:ww #1/#2 \q_stop
  { \__cooking_units_frac:nn {#1} {#2} }
\cs_new:Npn \__cooking_units_cuam_old_parse_mixed_frac:www #1_#2/#3 \q_stop
  {
    #1
    \hbox_to_wd:nn { \l__cooking_units_mixed_frac_dim } { }
    \__cooking_units_frac:nn {#2} {#3}
  }
```

F cufrac

Obsolete.

```
\NewDocumentCommand \cufrac { O{} m }
  {
    \msg_error:nnnn { cooking-units } { obsolete-command } { \cufrac } { \cuam }
    \group_begin:
    \tl_if_empty:nF {#1}
      { \keys_set:nn { cooking-units } {#1} }
    \__cooking_units_cufrac:n {#2}
    \group_end:
  }
\cs_new:Npn \__cooking_units_cufrac:n #1
  {
    \tl_set_rescan:Nnn \l__cooking_units_tmpa_tl
      {
        \char_set_catcode_letter:N \_ %
        \char_set_catcode_ignore:N\ %
      } {#1}
    \__cooking_units_cufrac_parse:V \l__cooking_units_tmpa_tl
  }
\cs_new:Npn \__cooking_units_cufrac_parse:n #1
  {
    \tl_if_in:nnTF {#1} { / }
      {
        \tl_if_in:nnTF {#1} { _ }
          { \__cooking_units_cufrac_parse_mixed_frac:www #1 \q_stop }
          { \__cooking_units_cufrac_parse_frac:ww #1 \q_stop }
      }{
        \tl_if_in:nnTF {#1} { _ }
          { \msg_error:nnn { cooking-units } { missing-slash } {#1} }
          { \__cooking_units_cufrac_parse_scale:n {#1} }
      }
  }
\cs_generate_variant:Nn \__cooking_units_cufrac_parse:n { V }
\cs_new:Npn \__cooking_units_cufrac_parse_scale:n #1 {#1}
\cs_new:Npn \__cooking_units_cufrac_parse_frac:ww #1/#2 \q_stop
  { \__cooking_units_frac:nn {#1} {#2} }
\cs_new:Npn \__cooking_units_cufrac_parse_mixed_frac:www #1_#2/#3 \q_stop
  {
    #1
    \hbox_to_wd:nn { \l__cooking_units_mixed_frac_dim } { }
    \__cooking_units_frac:nn {#2} {#3}
  }
```

G cukeys

G.1 Define Keys

\cudefinekeys Defining keys.

```
\NewDocumentCommand \cudefinekeys { m m }
  {
    \bool_set_false:N \l__cooking_units_single_key_bool
```

```
    \__cooking_units_cukeys_define_keys_and_single_keys:nn {#1} {#2}
  }
```

(*End definition for* `\cudefinekeys`. *This function is documented on page* 9.)

`\cudefinesinglekey` Again, but with the boolean set to true.

```
\NewDocumentCommand \cudefinesinglekey { m m }
  {
    \bool_set_true:N \l__cooking_units_single_key_bool
    \__cooking_units_cukeys_define_keys_and_single_keys:nn {#1} {#2}
  }
```

(*End definition for* `\cudefinesinglekey`. *This function is documented on page* 9.)

`_units_cukeys_define_keys_and_single_keys:nn` First checks whetever the unit is defined or not, then clears some macros which are needed later. Further procession depends on the boolean. If more than one key is created, the value `\l_@@_tmpa_fp` is set to one. This is important for adding keys where `\l_@@_tmpa_fp` is changed accordingly.

```
\cs_new:Npn \__cooking_units_cukeys_define_keys_and_single_keys:nn #1#2
  {
    \__cooking_units_error_if_unit_not_defined:n {#1}
    \tl_if_blank:nF {#2}
      {
        \seq_clear:N \l__cooking_units_tmpa_seq
        \prop_clear:N \l__cooking_units_tmpa_prop
        \bool_if:NTF \l__cooking_units_single_key_bool
          {
            \__cooking_units_cukeys_parse_and_create_single_key:nn {#1} {#2}
          }{
            \fp_set_eq:NN \l__cooking_units_tmpa_fp \c_one_fp
            \tl_set:Nn \l__cooking_units_given_unit_tl {#1}
            \__cooking_units_cukeys_parse_and_create_keys:nn {#1}
              {
                {#1} { \c_one_fp } #2
              }
          }
      }
  }
```

(*End definition for* `\__cooking_units_cukeys_define_keys_and_single_keys:nn`.)

`ooking_units_cukeys_parse_and_create_keys:nn` A simple parsing function using quarks (which are pretty handy). At first parses the input, then creates the property lists for each key (containing all the values) and at last defines the keys.

```
\cs_new:Npn \__cooking_units_cukeys_parse_and_create_keys:nn #1#2
  {
    \__cooking_units_cukeys_parse_input:nn #2
    \q_recursion_tail \q_recursion_tail \q_recursion_stop
    \__cooking_units_cukeys_create_key_prop:n {#1}
    \clist_set_from_seq:NN \l__cooking_units_tmpa_clist \l__cooking_units_tmpa_seq
    \__cooking_units_cukeys_define_keys:V \l__cooking_units_tmpa_clist
  }
```

(*End definition for* `\__cooking_units_cukeys_parse_and_create_keys:nn`.)

__cooking_units_cukeys_parse_input:nn

Yeah ... `\l_@@_tmpa_clist` stores all the used unit-keys, while the property list saves the relation to each other. This cycle is repeated until an recursion tail is found. If you define a new key, `\l_@@_tmpa_fp` is set to 1 and changed later if a new key is added.

```
\cs_new:Npn \__cooking_units_cukeys_parse_input:nn #1#2
  {
    \quark_if_recursion_tail_stop:n {#1}
    \quark_if_recursion_tail_stop_do:nn {#2}
      { \msg_error:nn { cooking-units } { missing-argument } }
    \__cooking_units_error_if_unit_not_defined:n {#1}
    \seq_put_right:Nn \l__cooking_units_tmpa_seq {#1}
    \prop_put:Nnx \l__cooking_units_tmpa_prop {#1} { \fp_eval:n { (#2) / \l__cooking_units_t
    \__cooking_units_cukeys_parse_input:nn
  }
```

(*End definition for* `\__cooking_units_cukeys_parse_input:nn`.)

__cooking_units_cukeys_create_key_prop:n

All linked unit-keys are stored within `\l_@@_tmpa_clist` and are mapped one after another. At first a property list is created (or cleared), this property list stores the units linked to this unit, saves the created property list which contains the numerical relation of each unit. `Erstes Ding` (firsth thing) stores the first unit used for defining the keys, it is needed later for adding keys.

For each unit the other units are added in the cleared or newly created property list as keys with their value being the correct numerical relation. For example:

$$1\,\mathrm{kg} = 1\,\mathrm{kg}$$
$$1\,\mathrm{kg} = 100\,\mathrm{dag}$$
$$1\,\mathrm{kg} = 1000\,\mathrm{g}$$
$$1\,\mathrm{kg} = 35.273\,99\,\mathrm{oz}$$

Therefore the property list for `kg` contains the keys and values: `kg=1` `dag=100`, `g=1000` and `oz= 35.27399`.

For the next unit (`dag` in this case) a new property list is created, relation above stored inside etc. The condition now is that `dag=1`, therefore every number is divided the the number `100`:

$$1\,\mathrm{dag} = 0.01\,\mathrm{kg}$$
$$1\,\mathrm{dag} = 1\,\mathrm{dag}$$
$$1\,\mathrm{dag} = 10\,\mathrm{g}$$
$$1\,\mathrm{dag} = 0.352\,739\,9\,\mathrm{oz}$$

Same for `g`:

$$1\,\mathrm{g} = 0.001\,\mathrm{kg}$$
$$1\,\mathrm{g} = 0.1\,\mathrm{dag}$$
$$1\,\mathrm{g} = 1\,\mathrm{g}$$
$$1\,\mathrm{g} = 0.035\,273\,99\,\mathrm{oz}$$

and oz

$$1\,\mathrm{oz} = 0.028\,349\,5\,\mathrm{kg}$$
$$1\,\mathrm{oz} = 2.834\,95\,\mathrm{dag}$$
$$1\,\mathrm{oz} = 28.3495\,\mathrm{g}$$
$$1\,\mathrm{oz} = 1\,\mathrm{oz}$$

```
\cs_new:Npn \__cooking_units_cukeys_create_key_prop:n #1
  {
    \prop_clear:N \l__cooking_units_tmpb_prop
    \prop_put:NnV \l__cooking_units_tmpb_prop { Liste } \l__cooking_units_tmpa_seq
    \prop_put:NnV \l__cooking_units_tmpb_prop { prop } \l__cooking_units_tmpa_prop
    \prop_put:NnV \l__cooking_units_tmpb_prop { Erstes Ding } \l__cooking_units_given_unit_t
    \seq_map_inline:Nn \l__cooking_units_tmpa_seq
      {
        \prop_set_eq:cN { l__cooking_units_cukeys_ ##1 _prop } \l__cooking_units_tmpb_prop
        \tl_set_eq:cN   { l__cooking_units_tmpa_ ##1 _tl } \c_one_fp
        \seq_map_inline:Nn \l__cooking_units_tmpa_seq
          {
            \prop_put:cnx { l__cooking_units_cukeys_ ##1 _prop }
              {####1}
              {
                \fp_eval:n
                  {
                    ( \prop_item:Nn \l__cooking_units_tmpa_prop {####1} ) /
                    ( \prop_item:Nn \l__cooking_units_tmpa_prop {##1} )
                  }
              }
          }
      }
  }
```

(*End definition for* `\__cooking_units_cukeys_create_key_prop:n`.)

`\__cooking_units_cukeys_define_keys:N`
`\__cooking_units_cukeys_define_keys:V`

Defining the keys: It maps through the list of unit-keys and creates a unit key respectively.

```
\cs_new:Npn \__cooking_units_cukeys_define_keys:N #1
  {
   \seq_map_inline:Nn \l__cooking_units_tmpa_seq
     {
       \seq_if_in:NnF \l__cooking_units_list_of_defined_keys_seq {##1}
         { \seq_put_right:Nn \l__cooking_units_list_of_defined_keys_seq {##1} }
       \keys_define:nn { cooking-units }
         {
           ##1 .choices:Vn =
             \l__cooking_units_tmpa_clist
             {
               \__cooking_units_cukeys_define_keys_and_single_key_aux:n {##1}
             } ,
           ##1 / unknown .code:n=
             {
               \seq_set_split:Nnn \l_tmpa_seq { , } {#1}
               \msg_error:nnxxx
```

```
                { cooking-units }
                { key-choice-unknown }
                {##1}
                {####1}
                { \seq_use:Nnnn \l_tmpa_seq { ',~ ' } { ',~ ' } { ' ~ and ~ ' } }
            } ,
          ##1 .default:n = {##1} ,
          ##1 .groups:n = { change-unit }
        }
    }
  }
\cs_generate_variant:Nn \__cooking_units_cukeys_define_keys:N { V }
```

(*End definition for* `\__cooking_units_cukeys_define_keys:N` *and* `\__cooking_units_cukeys_define_keys:V`.)

its_cukeys_define_keys_and_single_key_aux:n

```
\cs_new:Npn \__cooking_units_cukeys_define_keys_and_single_key_aux:n #1
  {
    \prop_get:cVc
      { l__cooking_units_cukeys_#1_prop }
      \l_keys_choice_tl
      { l__cooking_units_tmpa_ #1_tl }
    \prop_put:NnV \l__cooking_units_change_unit_prop {#1} \l_keys_choice_tl
  }
```

(*End definition for* `\__cooking_units_cukeys_define_keys_and_single_key_aux:n`.)

```
\cs_new:Npn \__cooking_units_cukeys_parse_and_create_single_key:nn #1#2
  {
    \tl_set_rescan:Nnn \l__cooking_units_tmpa_tl
      {
        \char_set_catcode_letter:N \# %
        \char_set_catcode_ignore:N\ %
      } { {#1} { \c_one_fp } #2 }
    \__cooking_units_cusinglekeys_parse_input:V \l__cooking_units_tmpa_tl
    \__cooking_units_cusinglekeys_create_key_prop:n {#1}
    \clist_set_from_seq:NN \l__cooking_units_tmpa_clist \l__cooking_units_tmpa_seq
    \__cooking_units_cukeys_define_singlekey:nV {#1} \l__cooking_units_tmpa_clist
  }
\cs_new:Npn \__cooking_units_cusinglekeys_parse_input:n #1
  {
    \__cooking_units_cusinglekeys_parse_input_aux:nn #1
    \q_recursion_tail \q_recursion_tail \q_recursion_stop
  }
\cs_generate_variant:Nn \__cooking_units_cusinglekeys_parse_input:n { V }
%
\cs_new:Npn \__cooking_units_cusinglekeys_parse_input_aux:nn #1#2
  {
    \quark_if_recursion_tail_stop:n {#1}
    \quark_if_recursion_tail_stop_do:nn {#2}
      { \msg_error:nn { cooking-units } { missing-argument } }
    \__cooking_units_error_if_unit_not_defined:n {#1}
    \seq_put_right:Nn \l__cooking_units_tmpa_seq {#1}
    \tl_set:Nn \l__cooking_units_tmpa_tl {#2}
```

```
    \tl_replace_all:NVn \l__cooking_units_tmpa_tl \c__cooking_units_input_str_hash_one_tl {
    \prop_put:NnV \l__cooking_units_tmpa_prop {#1} \l__cooking_units_tmpa_tl
    \__cooking_units_cusinglekeys_parse_input_aux:nn
  }
\cs_new:Npn \__cooking_units_cusinglekeys_create_key_prop:n #1
  {
    \tl_set_eq:cN { l__cooking_units_tmpa_ #1 _tl } \c_one_fp
    \prop_set_eq:cN { l__cooking_units_cukeys_ #1 _prop } \l__cooking_units_tmpa_prop
    \prop_put:cnn { l__cooking_units_cukeys_ #1 _prop } { Erstes Ding } {#1}
    \prop_put:cnV { l__cooking_units_cukeys_ #1 _prop } { Liste } \l__cooking_units_tmpa_seq
    \prop_put:cnV { l__cooking_units_cukeys_ #1 _prop } { prop } \l__cooking_units_tmpa_prop
  }
\cs_new:Npn \__cooking_units_cukeys_define_singlekey:nN #1#2
  {
    \seq_if_in:NnF \l__cooking_units_list_of_defined_keys_seq {#1}
      {  \seq_put_right:Nn \l__cooking_units_list_of_defined_keys_seq {#1} }
     \keys_define:nn { cooking-units }
        {
          #1 .choices:Vn =
            \l__cooking_units_tmpa_clist
              {
                \__cooking_units_cukeys_define_keys_and_single_key_aux:n {#1}
              } ,
          #1 / unknown .code:n=
            {
              \seq_set_split:Nnn \l_tmpa_seq { , } {#2}
              \msg_error:nnxxx
                { cooking-units }
                { key-choice-unknown }
                {#1}
                {##1}
                { \seq_use:Nnnn \l_tmpa_seq { ',' } { ',' } { ' ~ and ~ ' } }
            } ,
          #1 .default:n = {#1} ,
          #1 .groups:n = { change-unit } ,
        }
  }
\cs_generate_variant:Nn \__cooking_units_cukeys_define_singlekey:nN { nV }
```

H Adding Keys

Question to me: Why do we need `\l_tmpa(b)_tl`? Maybe due to `\cuaddtokeys`?

```
\NewDocumentCommand \cuaddkeys { m m }
  {
    \bool_set_false:N \l__cooking_units_single_key_bool
    \tl_set:Nn \l__cooking_units_tmpa_tl {#1}
    \tl_set:Nn \l__cooking_units_tmpb_tl {#2}
    \__cooking_units_cukeys_add_keys_or_single_keys:VV \l__cooking_units_tmpa_tl \l__cooking
  }
\NewDocumentCommand \cuaddsinglekeys { m m }
  {
    \bool_set_true:N \l__cooking_units_single_key_bool
```

```
    \tl_set:Nn \l__cooking_units_tmpa_tl {#1}
    \tl_set:Nn \l__cooking_units_tmpb_tl {#2}
    \__cooking_units_cukeys_add_keys_or_single_keys:VV \l__cooking_units_tmpa_tl \l__cooking
  }
\NewDocumentCommand \cuaddtokeys { m m m }
  {
    \bool_set_false:N \l__cooking_units_single_key_bool
    \tl_set:Nn \l__cooking_units_tmpa_tl {#1}
    \tl_set:Nn \l__cooking_units_tmpb_tl { {#2} { \fp_eval:n { \c_one_fp / (#3) } } }
    \__cooking_units_cukeys_add_keys_or_single_keys:VV \l__cooking_units_tmpa_tl \l__cooking
  }
\cs_new:Npn \__cooking_units_cukeys_add_keys_or_single_keys:nn #1#2
  {
    \__cooking_units_error_if_unit_not_defined:n {#1}
    \seq_if_in:NnF \l__cooking_units_list_of_defined_keys_seq {#1}
      { \msg_error:nnn { cooking-units } { Key-not-defined } {#1} }
    \tl_if_blank:nF {#2}
      {
        \__cooking_units_cukeys_add_keys_and_single_key_aux:n {#1}
        \bool_if:NTF \l__cooking_units_single_key_bool
          { \__cooking_units_cukeys_parse_and_create_single_key:nn {#1} {#2} }
          { \__cooking_units_cukeys_parse_and_create_keys:nn {#1} {#2} }
      }
  }
\cs_generate_variant:Nn \__cooking_units_cukeys_add_keys_or_single_keys:nn { VV }
\cs_new:Npn \__cooking_units_cukeys_add_keys_and_single_key_aux:n #1
  {
    \prop_get:cnN { l__cooking_units_cukeys_ #1 _prop } { Liste } \l__cooking_units_tmpa_seq
    \prop_get:cnN { l__cooking_units_cukeys_ #1 _prop } { prop } \l__cooking_units_tmpa_prop
    \prop_get:cnN { l__cooking_units_cukeys_ #1 _prop } { Erstes Ding } \l__cooking_units_gi
    \prop_get:cVN { l__cooking_units_cukeys_ #1 _prop } \l__cooking_units_given_unit_tl \l__
  }
```

I Creating New Units

\declarecookingunit
\newcookingunit
\providecookingunit

```
\NewDocumentCommand \declarecookingunit { o m }
  {
    \seq_if_in:NnTF \g__cooking_units_list_of_defined_units_seq {#2}
      { \msg_info:nnn { cooking-units } { redefine-unit } {#2} }
      { \__cooking_units_new_cooking_unit:nn {#1} {#2} }
    \__cooking_units_set_cooking_unit:nn {#1} {#2}
  }
\NewDocumentCommand \newcookingunit { o m }
  {
    \__cooking_units_new_cooking_unit:nn {#1} {#2}
    \__cooking_units_set_cooking_unit:nn {#1} {#2}
  }
\NewDocumentCommand \providecookingunit { o m }
  {
    \seq_if_in:NnF \g__cooking_units_list_of_defined_units_seq {#2}
      {
```

```
        \__cooking_units_new_cooking_unit:nn {#1} {#2}
        \__cooking_units_set_cooking_unit:nn {#1} {#2}
      }
  }
```

(*End definition for* `\declarecookingunit`, `\newcookingunit`, *and* `\providecookingunit`. *These functions are documented on page 8.*)

```
\cs_new:Npn \__cooking_units_new_cooking_unit:nn #1#2
  {
    \seq_if_in:NnTF \g__cooking_units_list_of_defined_units_seq {#2}
      { \msg_error:nnn { cooking-units } { unit-already-defined } {#2} }
      {
        \seq_put_right:Nn \g__cooking_units_list_of_defined_units_seq {#2}
        \tl_new:c { l__cooking_units_tmpa_  #2 _tl }
        \tl_set_eq:cN { l__cooking_units_tmpa_  #2 _tl } \c_one_fp
        \clist_new:c { l__cooking_units_predefined_option_#2_clist }
        \keys_define:nn { cooking-units }
          {
            set-option-for-#2 .clist_set:c = { l__cooking_units_predefined_option_#2_clist }
            add-option-for-#2 .code:n =
              { \clist_put_right:cn { l__cooking_units_predefined_option_#2_clist } {##1} },
          }
        \prop_new:c { l__cooking_units_cukeys_ #2 _prop }
        \tl_new:c { l__cooking_units_default_unit_ #2 _tl }
      }
  }
\cs_new:Npn \__cooking_units_set_cooking_unit:nn #1#2
  {
    \IfNoValueTF {#1}
      {
        \tl_set:cn { l__cooking_units_default_unit_ #2 _tl } {#2}
        \__cooking_units_deftranslation_base:xxn {#2} \c__cooking_units_postfix_unit_tl {#2}
      }{
        \tl_set:cn { l__cooking_units_default_unit_ #2 _tl } {#1}
        \__cooking_units_deftranslation_base:xxn {#2} \c__cooking_units_postfix_unit_tl {#1}
      }
    \__cooking_units_deftranslation_base:xxn {#2} \c__cooking_units_postfix_unitname_tl { \q
    \__cooking_units_deftranslation_base:xxn {#2} \c__cooking_units_postfix_unitname_pl_tl {
    \__cooking_units_deftranslation_base:xxn {#2} \c__cooking_units_postfix_gender_tl { m }
  }
```

J Names

```
\tl_new:N \l__cooking_units_sanitise_tl
\cs_new_protected:Npn \__cooking_units_sanitise_aux:w #1 \q_mark
  { \tl_set:Nn \l__cooking_units_sanitise_tl {#1} }
\group_begin:
  \char_set_catcode_active:n { `\< }
  \char_set_catcode_active:n { `\> }
  \cs_new:Npn \__cooking_units_sanitize_open_arrow:
    {
      \exp_after:wN \__cooking_units_sanitize_open_arrow_auxi:w \l__cooking_units_sanitise_t
        \q_mark < \q_nil <
```

```
      \exp_after:wN \__cooking_units_sanitise_aux:w \l__cooking_units_sanitise_tl
    }
  \cs_new_protected:Npn \__cooking_units_sanitize_open_arrow_auxi:w #1 <
    {
      \tl_set:Nn \l__cooking_units_sanitise_tl {#1}
      \__cooking_units_sanitize_open_arrow_auxii:w
    }
  \cs_new_protected:Npn \__cooking_units_sanitize_open_arrow_auxii:w #1 <
    {
      \quark_if_nil:nF {#1}
        {
          \tl_set:Nx \l__cooking_units_sanitise_tl
            {
              \exp_not:V \l__cooking_units_sanitise_tl
              \token_to_str:N <
              \exp_not:n {#1}
            }
          \exp_after:wN \__cooking_units_sanitize_open_arrow_auxii:w
        }
    }
  \cs_new:Npn \__cooking_units_sanitize_close_arrow:
    {
      \exp_after:wN \__cooking_units_sanitize_close_arrow_auxi:w \l__cooking_units_sanitise_
        \q_mark > \q_nil >
      \exp_after:wN \__cooking_units_sanitise_aux:w \l__cooking_units_sanitise_tl
    }
  \cs_new_protected:Npn \__cooking_units_sanitize_close_arrow_auxi:w #1 >
    {
      \tl_set:Nn \l__cooking_units_sanitise_tl {#1}
      \__cooking_units_sanitize_close_arrow_auxii:w
    }
  \cs_new_protected:Npn \__cooking_units_sanitize_close_arrow_auxii:w #1 >
    {
      \quark_if_nil:nF {#1}
        {
          \tl_set:Nx \l__cooking_units_sanitise_tl
            {
              \exp_not:V \l__cooking_units_sanitise_tl
              \token_to_str:N >
              \exp_not:n {#1}
            }
          \exp_after:wN \__cooking_units_sanitize_close_arrow_auxii:w
        }
    }
\group_end:
\cs_new_protected:Npn \__cooking_units_sanitize_arrows:n #1
  {
    \tl_set:Nn \l__cooking_units_sanitise_tl {#1}
    \__cooking_units_sanitize_open_arrow:
    \__cooking_units_sanitize_close_arrow:
  }
\NewDocumentCommand \cudefinename { m m }
  {
    \tl_set:Nn \l__cooking_units_language_tl {#1}
```

```
    \__cooking_units_sanitize_arrows:n {#2}
    \exp_last_unbraced:NV
    \__cooking_units_cuname_parse_input:n \l__cooking_units_sanitise_tl
    \q_recursion_tail \q_recursion_tail \q_recursion_stop
  }
\cs_new:Npn \__cooking_units_cuname_parse_input:n #1
  {
    \peek_meaning_ignore_spaces:NTF [
      {
        \__cooking_units_cuname_parse_unit_symbol:nw {#1}
      }{
        \clist_if_in:NnTF \g__cooking_units_allowed_special_keys_clist {#1}
          { \__cooking_units_cuname_parse_input_aux:nn {#1} }
          { \__cooking_units_cuname_parse_unit_symbol:nw {#1} [ \q_no_value ] }
      }
  }
\cs_new:Npn \__cooking_units_cuname_parse_unit_symbol:nw #1 [#2]
  {
    \quark_if_recursion_tail_stop:n {#1}
    \quark_if_recursion_tail_stop_do:nn {#2}
      { \msg_error:nn { cooking-units } { missing-argument } }
    \__cooking_units_error_if_unit_not_defined:n {#1}
    \quark_if_no_value:nTF {#2}
      {
        \__cooking_units_deftranslation_to:Vxxv
          \l__cooking_units_language_tl {#1}
          \c__cooking_units_postfix_unit_tl
          { l__cooking_units_default_unit_ #1 _tl }
      }{
        \__cooking_units_deftranslation_to:Vxxn
          \l__cooking_units_language_tl {#1}
          \c__cooking_units_postfix_unit_tl {#2}
      }
    \__cooking_units_cuname_parse_input_aux:nn {#1}
  }
\cs_new:Npn \__cooking_units_cuname_parse_input_aux:nn #1#2
  {
    \quark_if_recursion_tail_stop:n {#1}
    \quark_if_recursion_tail_stop_do:nn {#2}
      { \msg_error:nn { cooking-units } { missing-argument } }
    \clist_if_in:NnF \g__cooking_units_allowed_special_keys_clist {#1}
      { \__cooking_units_error_if_unit_not_defined:n {#1} }
    \__cooking_units_deftranslation_to:Vxxn
      \l__cooking_units_language_tl {#1}
      \c__cooking_units_postfix_unitname_tl {#2}
    \peek_meaning_ignore_spaces:NTF [
      { \__cooking_units_cuname_parse_bracket:nw {#1} }
      { \__cooking_units_cuname_parse_bracket:nw {#1} [#2] }
  }
\cs_new:Npn \__cooking_units_cuname_parse_bracket:nw #1 [#2]
  {
    \clist_if_in:NnF \g__cooking_units_allowed_special_keys_clist {#1}
      {
```

```
        \__cooking_units_deftranslation_to:Vxxn \l__cooking_units_language_tl {#1}
          \c__cooking_units_postfix_unitname_pl_tl {#2}
      }
    \peek_meaning_ignore_spaces:NTF <
      { \__cooking_units_cuname_parse_gender:nw {#1} }
      { \__cooking_units_cuname_parse_gender:nw {#1} <m> }
  }
\cs_new:Npn \__cooking_units_cuname_parse_gender:nw #1 <#2>
  {
    \__cooking_units_check_if_correct_gender_input:n {#2}
    \__cooking_units_deftranslation_to:Vxxn
      \l__cooking_units_language_tl {#1}
      \c__cooking_units_postfix_gender_tl {#2}
    \__cooking_units_cuname_parse_input:n
  }
```

J.1 cudefinesymbol

```
\NewDocumentCommand \cudefinesymbol { m m }
  {
    \tl_set:Nn \l__cooking_units_language_tl {#1}
    \__cooking_units_cuprint_define_printed_unit:nn #2
    \q_recursion_tail \q_recursion_tail \q_recursion_stop
  }
\cs_new:Npn \__cooking_units_cuprint_define_printed_unit:nn #1#2
  {
    \quark_if_recursion_tail_stop:n {#1}
    \quark_if_recursion_tail_stop_do:nn {#2}
      { \msg_error:nn { cooking-units } { missing-argument } }
    \clist_if_in:NnTF \g__cooking_units_allowed_special_keys_clist {#1}
      {
        \__cooking_units_deftranslation_to:Vxxn \l__cooking_units_language_tl {#1}
          \c__cooking_units_postfix_unitname_tl {#2}
      }{
        \__cooking_units_error_if_unit_not_defined:n {#1}
        \__cooking_units_deftranslation_to:Vxxn
          \l__cooking_units_language_tl {#1}
          \c__cooking_units_postfix_unit_tl {#2}
      }
    \__cooking_units_cuprint_define_printed_unit:nn
  }
```

J.2 Phrases

```
\__cooking_units_newtranslation_base:nVn { phrase-list } \c__cooking_units_postfix_phrase_tl
\prg_new_conditional:Npnn \__cooking_units_phrase_list_get_for:NN #1#2 { TF , T , F }
  {
    \__cooking_units_translate_let:VNxx #2 #1 { phrase-list } \c__cooking_units_postfix_phra
    \tl_if_eq:NNTF #1 \q__cooking_units_no_translation
      { \prg_return_false: }
      { \prg_return_true: }
  }
\prg_new_conditional:Npnn \__cooking_units_if_phrase_list_exists:N #1 { TF , T , F }
  {
```

```
    \__cooking_units_translate_let:Nxx \l_tmpa_tl { phrase-list } \c__cooking_units_postfix_
    \tl_if_eq:NNTF \l_tmpa_tl \q__cooking_units_no_translation
      { \prg_return_false: }
      { \prg_return_true: }
  }
\NewDocumentCommand \cudefinephrase { m m }
  {
    \__cooking_units_cuphrase:nn {#1} {#2}
  }
\cs_new:Npn \__cooking_units_cuphrase:nn #1#2
  {
    \tl_set:Nn \l__cooking_units_language_tl {#1}
      \__cooking_units_phrase_list_get_for:NNTF \l__cooking_units_phrase_prop \l__cooking_un
        {
          \__cooking_units_translate_let:VNxx \l__cooking_units_language_tl \l__cooking_unit
            { phrase-list-list } \c__cooking_units_postfix_phrase_tl
        }{
          \prop_clear:N \l__cooking_units_phrase_prop
          \clist_clear:N \l__cooking_units_phrase_numbers_clist
        }
      \__cooking_units_sanitize_arrows:n {#2}
      \exp_last_unbraced:NV
      \__cooking_units_cuphrase_parse:n \l__cooking_units_sanitise_tl
        \q_recursion_tail \q_recursion_tail \q_recursion_stop
    \clist_sort:Nn \l__cooking_units_phrase_numbers_clist
      {
        \int_compare:nNnTF { \int_abs:n {##1} } < { \int_abs:n {##2} }
          { \sort_return_swapped: }
          {
            \int_compare:nNnTF { \int_abs:n {##1} } = { \int_abs:n {##2} }
              {
                \int_compare:nNnTF {##1} < {##2}
                  { \sort_return_same: }
                  { \sort_return_swapped: }
              }{ \sort_return_same: }
          }
      }
    \__cooking_units_deftranslation_to:VxxV
      \l__cooking_units_language_tl { phrase-list }
      \c__cooking_units_postfix_phrase_tl
      \l__cooking_units_phrase_prop
    \__cooking_units_deftranslation_to:VxxV
      \l__cooking_units_language_tl { phrase-list-list }
      \c__cooking_units_postfix_phrase_tl \l__cooking_units_phrase_numbers_clist
  }
\cs_new:Npn \__cooking_units_cuphrase_parse:n #1
  {
    \quark_if_recursion_tail_stop:n {#1}
    \__cooking_units_if_integer:nF {#1}
      { \msg_error:nnn { cooking-units} { phrase-unit-not-an-integer } {#1} }
    \peek_meaning_remove_ignore_spaces:NTF *
      {
```

```
        \int_set:Nn \l__cooking_units_tmpa_int {-#1}
        \__cooking_units_cuphrase_parse_normal:Vn \l__cooking_units_tmpa_int
      }{
        \int_set:Nn \l__cooking_units_tmpa_int {#1}
        \__cooking_units_cuphrase_parse_normal:Vn \l__cooking_units_tmpa_int
      }
  }
\cs_new:Npn \__cooking_units_cuphrase_parse_normal:nn #1#2
  {
    \quark_if_recursion_tail_stop_do:nn {#2}
      { \msg_error:nn { cooking-units } { missing-argument } }
    \prop_put:Nnn \l__cooking_units_phrase_prop {#1} {#2}
    \clist_if_in:NnF \l__cooking_units_phrase_numbers_clist {#1}
      { \clist_put_right:Nn \l__cooking_units_phrase_numbers_clist {#1} }
    \peek_meaning_ignore_spaces:NTF [
      {
        \__cooking_units_chuphrase_parse_plural:nw {#1}
      }{
        \__cooking_units_chuphrase_parse_plural:nw {#1} [#2]
      }
  }
\cs_generate_variant:Nn \__cooking_units_cuphrase_parse_normal:nn { V }
\cs_new:Npn \__cooking_units_chuphrase_parse_plural:nw #1 [#2]
  {
    \prop_put:Nnn \l__cooking_units_phrase_prop { #1-pl } {#2}
    \peek_meaning_ignore_spaces:NTF <
      {
        \__cooking_units_chuphrase_parse_gender:nw {#1}
      }{
        \__cooking_units_chuphrase_parse_gender:nw {#1} < m >
      }
  }
\cs_new:Npn \__cooking_units_chuphrase_parse_gender:nw #1 <#2>
  {
    \__cooking_units_check_if_correct_gender_input:n {#2}
    \__cooking_units_deftranslation_to:Vxxn
      \l__cooking_units_language_tl { #1-phrase-gender }
      \c__cooking_units_postfix_gender_tl {#2}
    \__cooking_units_cuphrase_parse:n
  }
```

J.3 cusetup

```
\NewDocumentCommand \cusetup { m }
  {
    \keys_set:nn { cooking-units } {#1}
  }
```

J.4 Definitions et all

```
\newcookingunit { kg }
\newcookingunit { dag }
\newcookingunit { g }
\newcookingunit { oz }
```

```
\newcookingunit { lb }
\newcookingunit [ \ensuremath{ \__cooking_units_frac:nn { eV } { c^2 } } ] { eVc-2 }

\newcookingunit { K }
\newcookingunit [ \ensuremath{ {} ^ { \circ } } \kern-\scriptspace C ]  { C }
\newcookingunit [ \ensuremath{ {} ^ { \circ } } \kern-\scriptspace F ] { F }
\newcookingunit [ \ensuremath{ {} ^ { \circ } } \kern-\scriptspace R\'{e} ] { Re }

\newcookingunit { d }
\newcookingunit { h }
\newcookingunit { min }
\newcookingunit { s }
\newcookingunit [ \ensuremath{ \__cooking_units_frac:nn { \hbar } { eV } } ] { hbareV-1 }

\newcookingunit { m }
\newcookingunit { cm }
\newcookingunit { dm }
\newcookingunit { mm }
\newcookingunit { in }
\newcookingunit [ \ensuremath{ \__cooking_units_frac:nn { c\hbar } { eV } } ] { chbareV-1 }

\newcookingunit { l }
\newcookingunit { dl }
\newcookingunit { cl }
\newcookingunit { ml }
\newcookingunit [ \ensuremath { \__cooking_units_frac:nn { c^3 \hbar^3 } { eV^3 } } ] { (chb

\newcookingunit { cal }
\newcookingunit { kcal }
\newcookingunit { J }
\newcookingunit { kJ }
\newcookingunit { eV }

\newcookingunit [ pinch ] { pn }
\newcookingunit { EL }
\newcookingunit { TL }
\newcookingunit [ ssp. ] { ssp } %% saltspoonful
\newcookingunit [ csp. ] { csp } %% coffeespoonful
\newcookingunit [ dsp. ] { dsp }
\newcookingunit [ Msp. ] { Msp }

\DeclareLanguageAlias { AmericanEnglish } { American }

\cudefinename { German }
  {
    { kg } { Kilogramm } < n >
    { dag } { Dekagramm } < n >
    { g } { Gramm } < n >
    { oz } { Unze } < f >
    { lb } { Pfund } < n >

    { d } { Tag } [ Tage ]
    { h } { Stunde } [ Stunden ] < f >
    { min } { Minute } [ Minuten ] < f >
    { s } { Sekunde } [ Sekunden ] < f >

    { C } { Grad \space Celsius }
    { K } { Kelvin } < n >
    { F } { Grad \space Fahrenheit }
    { Re } { Grad \space R\'{e}amur }
```

```
    { m } { Meter } < n >
    { dm } { Dezimeter } < n >
    { cm } { Centimeter } < n >
    { mm } { Millimeter } < n >
    { in } { Zoll }

    { l } [ l ] { Liter }
    { dl } { Deziliter }
    { cl } { Centiliter }
    { ml } { Milliliter }

    { cal } { Kalorie } [ Kalorien ] < f >
    { kcal } { Kilokalorie } [ Kilokalorien ] < f >
    { J } { Joule }
    { kJ } { Kilojoule }
    { eV } { Elektronenvolt } < n >

    { Msp } [ Msp. ] { Messerspitze } [ Messerspitzen ] < f >
    { pn } [ Prise ] { Prise } [ Prisen ] < f >
    { EL } [ EL ] { Essl{\"o}ffel }
    { TL } [ TL ] { Teel{\"o}ffel }
    { csp } [ KL ] { Mokkal{\"o}ffel }

    { decimal-mark } { , }
    { one (m) } { ein }
    { one (f) } { eine }
    { one (n) } { ein }
  }
\cudefinename { English }
  {
    { kg } { kilogramme }
    { dag } { decagramme }
    { g } { gramme }
    { oz } { ounce }
    { lb } { pound } [ pounds ]

    { d } { day } [ days ]
    { h } { hour } [ hours ]
    { min } { minute } [ minutes ]
    { s } { second } [ seconds ]

    { C } { degree \space Celsius }  [ degrees \space Celsius ]
    { F } { degree \space Fahrenheit } [ degrees \space Fahrenheit ]
    { K } { kelvin }
    { Re } { degree \space R\'{e}aumur } [ degrees \space R\'{e}aumur ]

    { m } { metre } [ metres ]
    { dm } { decimetre } [ decimetres ]
    { cm } { centimetre } [ centimetres ]
    { mm } { millimitre } [ millimitres ]
    { in } { inch } [ inches ]

    { l } [ \ensuremath { \ell } ] { litre } [ litres ]
    { dl } { decilitre } [ decilitres ]
    { cl } { centilitre } [ centilitres ]
    { ml } { millilitre } [ millilitres ]

    { cal } { calorie } [ calories ]
    { kcal } { kilocalorie } [ kilocalories ]
```

```
    { J } { joule } [ joules ]
    { kJ } { kilojoule } [ kilojoules ]
    { eV } { electron \space volt }
%    { Msp } [ pinch ] { pinch } [ pinches ]
    { Msp } { Messerspitze } [ Messerspitzen ] <f>
    { pn } [ pinch ] { pinch } [ pinches ]
    { EL } [ tbsp. ] { tablespoon } [ tablespoons ]
    { TL } [ tsp. ] { teaspoon } [ teaspoons ]
    { dsp } { dessertspoonful }
    { csp } { coffeespoonful }
    { ssp } { saltspoonful }
    { decimal-mark } { . }
    { one (m) } { one }
    { one (f) } { one }
    { one (n) } { one }
  }
\cudefinename { AmericanEnglish }
  {
    { kg } { kilogram }
    { dag } { decagram }
    { g } { gram }
    { oz } { ounce }
    { m } { meter } [ meters ]
    { dm } { decimeter } [ decimeters ]
    { cm } { centimeter } [ centimeters ]
    { mm } { millimiter } [ millimiters ]
    { in } { inch } [ inches ]
    { l } [ \ensuremath { \ell } ] { liter } [ liters ]
    { dl } { deciliter } [ deciliters ]
    { cl } { centiliter } [ centiliters ]
    { ml } { milliliter } [ milliliters ]
    { Msp } { Messerspitze } [ Messerspitzen ] <f>
    { pn } [ pn. ] { pinch } [ pinches ]
  }
\cudefinename { French }
  {
    { kg } { kilogramme } [ kilogrammes ]
    { dag } { d\'{e}cagramme } [ d\'{e}cagrammes]
    { g } { gramme } [ gramme ]
    { oz } { once } < f >
    { lb } { livre } [ livres ] < f >
    { d } { jour } [ jours ]
    { h } { heure } [ heures ] < f >
    { min } { minute } [ minutes ] < f >
    { s } { seconde } [ secondes ] < f >
    { C } { degr\'{e} \space Celsius }  [ degr\'{e}s \space Celsius ]
    { K } { degr\'{e} \space Fahrenheit } [ degr\'{e}s \space Fahrenheit ]
    { F } { kelvin } [ kelvins ]
    { Re } { \'{e}chelle \space R\'{e}aumur } [ degr\'{e}s \space R\'{e}aumur ]
    { m } { m\‘{e}tre } [ m\‘{e}tres ]
```

```
    { dm } { d\'{e}cim\'{e}tre } [ d\'{e}cim\'{e}tres ]
    { cm } { centim\'{e}tre } [ centim\'{e}tres ]
    { mm } { millim\'{e}tre } [ millim\'{e}tres ]
    { in } [ po ] { pouce } [ pouces ]
    { l } [ L ] { litre } [ litres ]
    { dl } [ dL ] { d\'{e}cilitre } [ d\'{e}cilitres ]
    { cl } [ cL ] { centilitre } [ centilitres ]
    { ml } [ mL ] { millilitre } [ millilitres ]
    { cal } { calorie } [ calorie ]
    { kcal } { kilocalorie } [ kilocalories ]
    { J } { joule } [ joules ]
    { kJ } { kilojoule } [ kilojoules ]
    { eV } { \'{e}lectron-volt } [ \'{e}lectron-volts ]
    { pn } { pinc\'{e}e } < f >
    { EL } { cuill\'{e}re  \space \'{a} \space  soupe } < f >
    { TL } { cuill\'{e}re \space \'{a} \space caf\'{e} } < f >
    { decimal-mark } { . }
    { one (m) } { un }
    { one (f) } { une }
    { one (n) } { un }
  }
\cudefinekeys { kg }
  {
    { dag }{ 100 }
    { g }  { 1000 }
    { oz } { 35.27399 }
    { lb } { 2.204 622 6 } %% 2.204 622 6
    { eVc-2 } { 560958865.0e+27 } %% 560958865.0 +- 3.5 e+27
  }
%\cuaddtokeys { kg } { eVc-2 } { 1.78266173e-16 }
\cudefinekeys { d }
  {
    { h }  { 24 }
    { min }{ 1440 }
    { s }  { 86400 }
    { hbareV-1 } { 151926746.1e+7 * 86400 } %% 151926746.1 +- 2.1 e+7
  }
%\cuaddtokeys { s } { hbareV-1 } { 6.582119514e-16 }
\cudefinekeys { m }
  {
    { dm } { 10 }
    { cm } { 100 }
    { mm } { 1000 }
    { in } { 39.370079 }
    { chbareV-1 } { 5067730.759 } %% 5067730.759 +- 0.070
  }
%\cuaddtokeys { m } { chbareV-1 } { 1.97326972e-7 }
\cudefinekeys { l }
  {
    { dl } { 10 }
    { cl } { 100 }
```

```
    { ml }{ 1000 }
    { (chbareV-1)3 } { 130148929.5e+12 * 1e-3 } %% (130148929.5 +- 5.4 e+12)*1e-3
  }
\cudefinekeys { J }
  {
    { kJ }{ 1e-3 }
    { cal }{ 0.2388459 }
    { kcal }{ 0.2388459e-3 }
    { eV }{ 624150912.6e+10 } %% 624150912.6 +- 3.9 e+10
  }
%\cuaddtokeys { J } { eV } { 1.6021766208e-19 }
\fp_const:Nn \c__cooking_units_kb_eV_fp {  8.617 330 3 e-5  }
\cudefinesinglekey { C }
  {
    { K } { #1 + 273.15 }
    { F } { #1 * 1.8 + 32 }
    { Re } { #1 * 0.8 }
    { eV } { ( #1 + 273.15 ) * \c__cooking_units_kb_eV_fp }
  }
\cudefinesinglekey { F }
  {
    { C } { ( #1 - 32 ) *  5/9 }
    { K } { ( #1 + 459.67 ) *  5/9 }
    { Re } { ( #1 - 32 ) * 4/9 }
    { eV } {( #1 + 459.67 ) *  5/9 * \c__cooking_units_kb_eV_fp }
  }
\cudefinesinglekey { K }
  {
    { C } { #1 - 273.15 }
    { F } { #1 * 1.8 - 459.67 }
    { Re } { ( #1 - 273.15 ) * 0.8 }
    { eV } { #1 * \c__cooking_units_kb_eV_fp }
  }
\cudefinesinglekey { Re }
  {
    { K } { #1 * 1.25 + 273.15 }
    { C } { #1  * 1.25 }
    { F } { #1 * 2.25 + 32 }
    { eV } { ( #1 * 1.25 + 273.15 ) * \c__cooking_units_kb_eV_fp }
  }
\cudefinephrase { German }
  {
%    { 6 } * { halbes \  Dutzend } < n >
%    { 6 }  { efkjwefjkl \  Dutzend } < n >
    { 12 } { Dutzend } < n >
%    { 60 } { Schock } < n >
%    { 1728 } { Gro{\ss}gros } < n >
%    { 144 } { Gros } < n >
  }
```

J.5 Finish

```
\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 } ,
    add-temperature-to-check =
      {
        K = 0,
        C = -273.15 ,
        F = -459.67 ,
        Re = -218.52
      } ,
  }
```

⟨/package⟩