siunitx – A comprehensive (SI) units package*

Joseph Wright†

Released 2026-03-26

Contents

*This file describes v3.5.1, last revised 2026-03-26.

†E-mail: joseph@texdev.net

1 Formatting complex values

This submodule is concerned with formatting complex numbers. It augments the standard functions `\siunitx_number_format:nN` and `\siunitx_quantity:nn` by allowing parsing of numbers with a complex part. There are no additional assumptions concerning LaTeX 2_ε commands in the submodule beyond those in the core number and unit submodules.

`\siunitx_complex_number:n` `\siunitx_complex_number:e`

`\siunitx_complex_number:n {⟨number⟩}`

Parses the ⟨*number*⟩ and splits into real and complex parts, which are then formatted as described for `\siunitx_number_format:nN`. The results are combined and printed using the standard functions in the module. If the setting `complex-mode` is set to ⟨*polar*⟩, the input is parsed, converted to polar form and then passed to `\siunitx_complex_number:nn`. This parsing requires that the complex root is given as `i` at the *end* of the value.

`\siunitx_complex_number:nn`

`\siunitx_complex_number:nn {⟨magnitude⟩} {⟨angle⟩}`

Parses the ⟨*magnitude*⟩ and ⟨*angle*⟩ and then formats each as described for `\siunitx_number_format:nN`. The two are separated by the angle symbol, which is treated as a numerical part. If `complex-angle-unit` is set to `degrees` then the unit symbol is added: this is printed as a unit in the usual way. If the setting `complex-mode` is set to ⟨*cartesian*⟩, the input is parsed, converted to Cartesian form and then passed to `\siunitx_complex_number:n`.

`\siunitx_complex_quantity:nn` `\siunitx_complex_quantity:en` `\siunitx_complex_quantity:nnn`

`\siunitx_complex_quantity:nn {⟨number⟩} {⟨units⟩}`
`\siunitx_complex_quantity:nnn {⟨magnitude⟩} {⟨angle⟩} {⟨units⟩}`

These functions treat their numerical argument(s) as described for the corresponding `number` functions. They then typeset the entire numerical part and the unit as described for `\siunitx_quantity:nn`.

`complex-angle-unit`

`complex-angle-unit = degrees|radians`

Sets how the unit for polar complex numbers is treated. This setting is used to determine how polar *input* is interpreted, how conversion to polar form works and how output in polar form is typeset. The standard setting is `degrees`.

`complex-mode`

`complex-mode = cartesian|input|polar`

Selects how complex values are formatted: a choice from the options `cartesian`, `input` and `polar`. The option `cartesian` means that complex values will always be typeset in cartesian ($x + y$i) format, whilst `polar` means that complex are typeset as a magnitude and angle. Finally, `input` setting means that the input format (*i.e.* difference between `\siunitx_complex_number:n` and `\siunitx_complex_number:nn`) is maintained. The standard setting is `input`.

`complex-phase-command`

`complex-phase-command = ⟨cmd⟩`

Sets the command used during output of the phase of a complex number in polar form. The standard setting is `\angle`.

`complex-root-position` `complex-root-position = after-number|before-number`

Choice which determines where the complex root symbol is printed relative to the numbers. The standard setting is `after-number`.

`complex-symbol-degree` `complex-symbol-degree =` ⟨*symbol*⟩

Sets the symbol used for polar degrees.

`input-complex-root` `input-complex-root =` ⟨*tokens*⟩

The token(s) considered as complexes roots for number parsing. The standard setting is `ij`.

`output-complex-root` `output-complex-root =` ⟨*tokens*⟩

The token(s) used to show the complex root in output. The standard setting is `\mathrm{i}`.

`print-complex-unity` `print-complex-unity = true|false`

Switch to determine if the number 1 is printed for a complex part which is exactly unity.

2 Compound numbers and quantities

`\siunitx_compound_number:n` `\siunitx_compound_number:n {`⟨*entries*⟩`}`

Prints a set of numbers in the ⟨*entries*⟩, each of which should be given as a ⟨*balanced text*⟩. Unlike `\siunitx_number_list:nn`, this function may semantically take any form.

`\siunitx_compound_quantity:nn` `\siunitx_compound_quantity:nn {`⟨*entries*⟩`} {`⟨*unit*⟩`}`

Prints a set of quantities in the ⟨*entries*⟩, each of which should be given as a ⟨*balanced text*⟩. Unlike `\siunitx_quantity_list:nn`, this function may semantically take any form.

`\siunitx_number_list:n` `\siunitx_number_list:nn {`⟨*entries*⟩`}`

Prints the list of numbers in the ⟨*entries*⟩, each of which should be given as a ⟨*balanced text*⟩.

`\siunitx_quantity_list:nn` `\siunitx_quantity_list:nn {`⟨*entries*⟩`} {`⟨*unit*⟩`}`

Prints the list of quantities in the ⟨*entries*⟩, each of which should be given as a ⟨*balanced text*⟩.

`\siunitx_number_product:n` `\siunitx_number_product:n {`⟨*entries*⟩`}`

Prints the series of numbers in the ⟨*entries*⟩, each of which should be given as a ⟨*balanced text*⟩.

`\siunitx_quantity_product:nn` `\siunitx_number_product:n {⟨entries⟩} {⟨unit⟩}`

Prints the series of quantities in the ⟨*entries*⟩, each of which should be given as a ⟨*balanced text*⟩.

`\siunitx_number_range:nn` `\siunitx_number_range:nn {⟨start⟩} {⟨end⟩}`

Prints the range of numbers from the ⟨*start*⟩ to the ⟨*end*⟩.

`\siunitx_quantity_range:nnn` `\siunitx_number_range:nn {⟨start⟩} {⟨end⟩} {⟨unit⟩}`

Prints the range of quantities from the ⟨*start*⟩ to the ⟨*end*⟩.

`\l_siunitx_list_separator_pair_tl`
`\l_siunitx_list_separator_tl`
`\l_siunitx_list_separator_final_tl`

Separators for lists of numbers and quantities.

`\l_siunitx_range_phrase_tl` Phrase (or similar) used between limits of a range.

`compound-boundary-mode` `compound-boundary-mode = number|text`

Choice which determines whether the material at the start and end of a compound quantity are typeset a `number` or as `text`; the latter is the standard setting.

`compound-close-boundary` `compound-close-boundary = ⟨tokens⟩`
`compound-open-boundary` `compound-open-boundary = ⟨tokens⟩`

Literals which are inserted at the opening and closing boundary of a compound quantity; they are not used when the number of items is one. The standard settings set these empty.

`compound-close-bracket` `compound-close-bracket = ⟨token⟩`
`compound-open-bracket` `compound-open-bracket = ⟨token⟩`

Literals containing the tokens inserted at the start and end of a compound value when `compounds-units` is set to `bracket`. The standard settings are `(` and `)`.

`compound-exponents` `compound-exponents = combine|combine-bracket|individual`

`compound-final-separator` `compound-final-separator = ⟨text⟩`

`compound-independent-prefix` `compound-independent-prefix = true|false`

Switch which determines whether unit prefixes are calculated independently when units are repeated. The standard setting is `false`.

`compound-pair-separator` `compound-pair-separator = ⟨text⟩`

compound-separator `compound-separator = ⟨text⟩`

compound-separator-mode `compound-separator-mode = number|text`

Choice which determines whether the separators between components of compound quantity are typeset a `number` or as `text`; the latter is the standard setting.

compound-units `compound-units = bracket|repeat|single`

list-close-bracket `list-close-bracket = ⟨token⟩`
list-open-bracket `list-open-bracket = ⟨token⟩`

Literals containing the tokens inserted at the start and end of a list when `list-units` is set to `bracket`. The standard settings are `(` and `)`.

list-exponents `list-exponents = combine|combine-bracket|individual`

list-final-separator `list-final-separator = ⟨text⟩`

list-independent-prefix `list-independent-prefix = true|false`

Switch which determines whether unit prefixes are calculated independently when units are repeated. The standard setting is `false`.

list-pair-separator `list-pair-separator = ⟨text⟩`

list-separator `list-separator = ⟨text⟩`

list-units `list-units = bracket|repeat|single`

product-close-bracket `product-close-bracket = ⟨token⟩`
product-open-bracket `product-open-bracket = ⟨token⟩`

Literals containing the tokens inserted at the start and end of a product when `product-units` is set to `bracket`. The standard settings are `(` and `)`.

product-exponents `product-exponents = combine|combine-bracket|individual`

product-independent-prefix `product-independent-prefix = true|false`

Switch which determines whether unit prefixes are calculated independently when units are repeated. The standard setting is `false`.

product-mode `product-mode = phrase|choice`

product-phrase `product-phrase = ⟨text⟩`

product-symbol `product-symbol = ⟨symbol⟩`

range-exponents `range-exponents = combine|combine-bracket|individual`

range-close-bracket `range-close-bracket = ⟨token⟩`
range-open-bracket `range-open-bracket = ⟨token⟩`

Literals containing the tokens inserted at the start and end of a range when `range-units` is set to `bracket`. The standard settings are `(` and `)`.

range-independent-prefix `range-independent-prefix = true|false`

Switch which determines whether unit prefixes are calculated independently when units are repeated. The standard setting is `false`.

range-open-phrase `range-open-phrase = ⟨text⟩`

Literal containing the material to be inserted at the start of a range. The standard setting is empty.

range-phrase `range-phrase = ⟨text⟩`

Literal containing the material to be inserted between the start and end of a range. The standard setting contains the word `to` inside the `\text` command, along with appropriate spacing commands to allow this material to work in both math and text typesetting modes.

range-units `range-units = bracket|repeat|single`

3 Localization

This submodule is concerned with localization of siunitx output based on the locale. If the translations package is available, this is loaded here and used to provide various fixed strings for output.

locale `locale = ⟨locale⟩`

Selects the ⟨*locale*⟩ used to apply standard settings for other keys, principally `exponent-product`, `inter-unit-product` and `output-decimal-marker`.

4 Parsing and formatting numbers

This submodule is dedicated to parsing and formatting numbers. A small number of LaTeX 2_ε math mode commands are assumed to be available as part of the formatted output. The sign commands `\mp`, `\pm`, `\ll`, `\le`, `\gg` and `\ge` are used to replace two-character input; `\pm` is also required for the output of uncertainties, and `\sim` for approximate values. The standard settings require `\times`. For the display of colored negative numbers, the command `\color` is assumed to be available. Where the latter may apply, numbers should be printed inside a group: note that TeX grouping is not added *within* formatted numbers as they may need to be decomposed into parts (see `\siunitx_number_output:NN`). Such a color will be the *first* part of the result, meaning that a test for an initial `\color` and following brace group may be used to detect/remove/adjust this part.

`\siunitx_number_parse:nN` `\siunitx_number_parse:VN`

`\siunitx_number_parse:nN {⟨number⟩} ⟨tl var⟩`

Parses the *number* and stores the resulting internal representation in the ⟨*tl var*⟩. The parsing is influenced by the various key–value settings for numerical input. The ⟨*number*⟩ should comprise a single real value, possibly with comparator, uncertainty and exponent parts. If the number is invalid, or if number parsing is disabled, the result will be an entirely empty ⟨*tl var*⟩.

The structure of a valid number is:

`{`⟨*comparator*⟩`}{`⟨*sign*⟩`}{`⟨*integer*⟩`}{`⟨*decimal*⟩`} {`⟨*uncertainty*⟩`}`
`{`⟨*exponent sign*⟩`}{`⟨*exponent*⟩`}`

where the two sign parts must be single tokens if present, and all other components must be given in braces. The number will have at least one digit for either the ⟨*integer*⟩ and ⟨*exponent*⟩ parts. The ⟨*uncertainty*⟩ part should either be blank or contain an ⟨*identifier*⟩ (as a brace group), followed by one or more data entries. Valid uncertainty ⟨*identifiers*⟩ currently are

- `S` A single symmetrical uncertainty (*e.g.* a statistical standard uncertainty). The data item here is a single value representing the uncertainty in the least-significant digits.
- `A` A single unsymmetrical uncertainty. The data item here contains two brace groups, each using the same least-significant digit approach as the `S` type. The positive component is given first and the negative second, and neither has a sign.
- A combination of `S` and `A` entries, with one data item per entry. These are then iterated over to be output in order.

If a decimal marker should be explicitly recorded as present for a value with no decimal digits, the ⟨*decimal*⟩ part should contain `\empty`.

\siunitx_number_process:NN \siunitx_number_process:N ⟨*tl var1*⟩ ⟨*tl var2*⟩
\siunitx_number_process:cc

Applies a set of number processing operations to the ⟨*internal number*⟩ stored in the ⟨*tl var1*⟩, *viz.* in order

1. Dropping uncertainty
2. Converting to scientific mode (or similar)
3. Rounding
4. Dropping zero decimal part
5. Forcing a minimum number of digits

with the result stored in ⟨*tl var2*⟩.

\siunitx_number_output:N		⋆	\siunitx_number_output:N ⟨*number*⟩
\siunitx_number_output:c	\siunitx_number_output:n	⋆	\siunitx_number_output:NN ⟨*number*⟩ ⟨*marker*⟩
\siunitx_number_output:NN		⋆	
\siunitx_number_output:cN	\siunitx_number_output:nN	⋆	

Formats the ⟨*number*⟩ (in the siunitx internal format), producing the result in a form suitable for typesetting in math mode. The details for the formatting are controlled by a number of key–value options. Note that *formatting* does not apply any manipulation (processing) to the number. This function is usable in an e- or x-type expansion, and further uncontrolled expansion is prevented by appropriate use of \exp_not:n internally.

In the NN version, the ⟨*marker*⟩ token is inserted at each possible alignment position in the output, *viz.*

- Between the comparator and the integer (*before* any sign for the integer)
- Between the sign and the first digit of the integer
- Both sides of the decimal marker
- Both sides of the separated uncertainty sign (*i.e.* after the decimal part and before any integer uncertainty part)
- Both sides of the decimal marker for a separated uncertainty
- Both sides of the multiplication symbol for the exponent part.

The n and nN version take a token list, which should be in the internal siunitx format.

\siunitx_number_format:nN \siunitx_number_format:nN {⟨*number*⟩} ⟨*tl var*⟩

Carries out a combination of \siunitx_number_parse:nN, \siunitx_number_process:NN and \siunitx_number_output:N using x-type expansion to place the result in the ⟨*tl var*⟩. If \l_siunitx_number_parse_bool if false, the input is simply stored inside the ⟨*tl var*⟩ inside \ensuremath.

\siunitx_number_adjust_exponent:Nn ⋆ \siunitx_number_adjust_exponent:Nn ⟨*number*⟩ {⟨*fp expr*⟩}
\siunitx_number_adjust_exponent:nn ⋆
\siunitx_number_adjust_exponent:Vn ⋆

Adjusts the exponent of the ⟨*number*⟩ (in internal format) by the ⟨*fp expr*⟩ and leaves the result in the input stream.

\siunitx_number_normalize_symbols:N \siunitx_number_normalize_symbols:N ⟨*tl var*⟩

Replaces all multi-token signs and comparators in the ⟨*tl var*⟩ with their single-token equivalents. Replaces any active hyphen tokens with non-active versions.

\siunitx_if_number_p:n ⋆ \siunitx_if_number_token:NTF {⟨*tokens*⟩}
\siunitx_if_number:nTF ⋆ {⟨*true code*⟩} {⟨*false code*⟩}

Determines if the ⟨*tokens*⟩ form a valid number which can be fully parsed by siunitx.

\siunitx_if_number_token:NTF \siunitx_if_number_token:NTF {⟨*token*⟩}
{⟨*true code*⟩} {⟨*false code*⟩}

Determines if the ⟨*token*⟩ is valid in a number based on those tokens currently set up for detection in a number.

\l_siunitx_bracket_ambiguous_bool

A switch to control whether ambiguous numbers are bracketed: this can also be covered in quantity formatting by a setting there.

\l_siunitx_number_parse_bool

A switch to control whether any parsing is attempted for numbers.

\l_siunitx_number_comparator_tl
\l_siunitx_number_exponent_tl
\l_siunitx_number_sign_tl

The list of possible input comparators, exponent markers and signs.

\l_siunitx_number_input_decimal_tl
\l_siunitx_number_output_decimal_tl

The list of possible input decimal marker(s), and the output marker.

allow-uncertainty-breaks allow-uncertainty-breaks = true|false

Specifies whether breaks are permitted for a (separated) uncertainties. The standard setting is `true`.

bracket-ambiguous-numbers bracket-ambiguous-numbers = true|false

bracket-negative-numbers bracket-negative-numbers = true|false

`drop-exponent` `drop-exponent = true|false`

`drop-uncertainty` `drop-uncertainty = true|false`

`drop-zero-decimal` `drop-zero-decimal = true|false`

`evaluate-expression` `evaluate-expression = true|false`

`exponent-base` `exponent-base =` ⟨*base*⟩

`exponent-mode` `exponent-mode = engineering|fixed|input|scientific|threshold`

Choice which determines whether numbers are converted to exponent form. The option `engineering` forces exponent form with an exponent which is the smallest power of three which gives a mantissa with an integer part. The option `fixed` uses a fixed exponent (set in `fixed-exponent`). The option `input` leaves the input unchanged (which will therefore produce an exponent only if the input contained one). The choice `scientific` gives an exponent with the mantissa m in the range $1 \leq m < 10$. Finally, the option `threshold` will apply `scientific` if the exponent of input is outside of the range stored in `exponent-thresholds`. The standard setting is `input`.

`exponent-product` `exponent-product =` ⟨*symbol*⟩

`expression` `expression =` ⟨*expression*⟩

`final-digit-group-min-size` `final-digit-group-min-size =` ⟨*integer*⟩

Minimum number of digits which must be present in a group of digits; if this is not met, the digits of the final group will be combined with those from the immediately-preceding one.

`fixed-exponent` `fixed-exponent =` ⟨*exponent*⟩

`digit-group-size` `digit-group-number =` ⟨*integer*⟩
`digit-group-first-size`
`digit-group-other-size`

Sets the size of the block (the number of digits) used when grouping digits. The option `digit-group-first-size` applies to the first grouping, *i.e.* immediately next to the decimal marker, while `digit-group-other-size` applies to all other groups. Both can be set using `digit-group-size`. The standard setting for both options is 3.

`group-digits` `group-digits = all|decimal|integer|none`

Choice to specify whether digits in a number are grouped. The option `none` entirely disables this, while `all` means that both the integer and decimal parts are grouped. The settings `integer` and `decimal` activate grouping for the relevant part only. The standard setting is `all`.

group-minimum-digits `group-minimum-digits = ⟨value⟩`

The number of digits that must be present in a numerical part (integer or decimal) before digit grouping is attempted. The standard setting is 4.

group-separator `group-separator = ⟨symbol⟩`

Sets the symbol inserted between groups of digits. The standard setting is a thin space (`\,`).

input-close-uncertainty `input-close-uncertainty = ⟨tokens⟩`

input-comparators `input-comparators = ⟨tokens⟩`

input-close-uncertainty `input-close-uncertainty = ⟨tokens⟩`

input-decimal-markers `input-decimal-markers = ⟨tokens⟩`

input-digits `input-digits = ⟨tokens⟩`

input-exponent-markers `input-exponent-markers = ⟨tokens⟩`

input-open-uncertainty `input-open-uncertainty = ⟨tokens⟩`

input-signs `input-signs = ⟨tokens⟩`

input-uncertainty-signs `input-uncertainty-signs = ⟨tokens⟩`

input-uncertainty-divider `input-uncertainty-divider = ⟨tokens⟩`

minimum-decimal-digits `minimum-decimal-digits = ⟨min⟩`

minimum-integer-digits `minimum-integer-digits = ⟨min⟩`

negative-color `negative-color = ⟨color⟩`

output-close-uncertainty `output-close-uncertainty = ⟨symbol⟩`

`output-decimal-marker` `output-decimal-marker = ⟨symbol⟩`

`output-open-uncertainty` `output-open-uncertainty = ⟨symbol⟩`

`parse-numbers` `parse-numbers = true|false`

`print-implicit-plus` `print-implicit-plus = true|false`
`print-mantissa-implicit-plus` `print-mantissa-implicit-plus = true|false`
`print-exponent-implicit-plus` `print-exponent-implicit-plus = true|false`

Controls whether the plus sign implicit in a positive number is printed; this can be controlled at the level of the mantissa or exponent, or can be activated for both.

`print-unity-mantissa` `print-unity-mantissa = true|false`

`print-zero-exponent` `print-zero-exponent = true|false`

`print-zero-integer` `print-zero-integer = true|false`

`retain-explicit-plus` `retain-explicit-plus = true|false`

Switch which determines if an explicit `+` is retained as a sign when parsing. The standard setting is `false`.

`retain-explicit-decimal-marker` `retain-explicit-decimal-marker = true|false`

Switch which determines if an explicit decimal marker is retained when parsing a number where there is no decimal part to a number (*i.e.* whether to differentiate `10` and `10.`). The standard setting is `false`.

`retain-negative-zero` `retain-negative-zero = true|false`

Switch which determines if a negative sign is retained where the value of a parsed number is exactly zero. The standard setting is `false`.

`retain-zero-uncertainty` `retain-zero-uncertainty = true|false`

Switch which determines if an entirely zero uncertainty part is retained on parsing, or whether this is normalised to remove the uncertainty. The standard setting is `false`.

`round-direction` `round-direction = down|nearest|up`

Choice which determines how values are rounded. The setting `up` means that the value is always rounded away from zero, whereas the setting `down` means that the value will be rounded toward zero. The setting `nearest` means that the value will be rounded to the nearest (either up or down), taking account of the setting of `round-half`. The standard setting is `nearest`.

round-half `round-half = even|up`

Choice which determines how values of exactly half are rounded. The setting `up` means that the value is always rounded away from zero, whereas the setting `even` means that the value will be rounded to the closes even number. The standard setting is `up`.

round-minimum `round-minimum =` ⟨*min*⟩

Literal which sets a minimum value below which rounded values will be replaced by this value and a `>` or `<`, as appropriate for the sign of the value. The standard setting is empty, *i.e.* there is no minimum.

round-mode `round-mode = figures|none|places|uncertainty`

Choice which specifies the rounding approach used for numbers. The choice `figures` means that values are rounding to the number of significant figures specified by `round-precision`. The setting `places` rounds to `round-precision` interpreted as a number of decimal places: this may be negative (rounding to an integer). The setting `none` disables rounding. The setting `uncertainty` first rounds the uncertainty to the number of significant figures specified by `round-precision`, then rounds the main value such that its accuracy is correctly specified by this updated uncertainty. The standard setting is `none`.

round-pad `round-pad = true|false`

Switch which specifies if values should be padded to the required number length when rounding to a number of decimal places. The standard setting is `true`.

round-precision `round-precision =` ⟨*precision*⟩

Integer specifying the number of digits used as a target when rounding: this may be interpreted as decimal places or significant figures, depending on active `round-mode`. The standard setting is 2.

round-zero-positive `round-zero-positive = true|false`

Switch to control whether a value rounded to zero is regarded as a positive number if the input was negative. The standard setting is `true`.

simplify-uncertainty `simplify-uncertainty = true|false`

Switch to control whether uncertainties given with two equal components are printed as a single value.

tight-spacing `tight-spacing = true|false`

uncertainty-descriptor-mode `uncertainty-descriptor = bracket|bracket-separator|separator|subscript`

Selects how uncertainty descriptors are formatted: a choice from the options `bracket`, `text` and `subscript`. The option `bracket` wraps the descriptor in parenthesis, `bracket-separator` does the same but also includes a separator between the uncertainty and opening bracket, `separator` places the descriptor after the uncertainty and a separator, and `subscript` formats the descriptor as a subscript. The standard setting is `bracket-separator`.

uncertainty-descriptor-separator `uncertainty-descriptor-separator = ⟨separator⟩`

Separateor inserted between the uncertainty and descriptor when one is required by `uncertainty-separator-mode`. The standard setting is `\␣`.

uncertainty-descriptors `uncertainty-descriptors = ⟨clist⟩`

Stores the list of descriptors used when there are multiple uncertainty components given. This is not used when there is only a single uncertainty component present. The standard setting is `empty`.

uncertainty-mode `uncertainty-mode = compact|compact-marker|full|separate`

Switch to determine how single symmetrical uncertainties are formatted. When this is set to `separate`, the uncertainty is printed as an entirely separate number preceded by `\pm`. Other settings all place the uncertainty in parentheses directly attached to the main value. The standard setting of `compact` prints digits of uncertainty in the least-significant digits. It does *not* print a decimal marker if the uncertainty crosses the decimal. The setting `full` prints the full value of the uncertainty. The setting `compact-marker` is available to print in the `compact` style except where the uncertainty crosses the decimal, in which case the `full` style is used. The standard setting is `compact`.

uncertainty-round-direction `uncertainty-round-direction = down|nearest|up`

Choice which determines how uncertainty values are rounded. The setting `up` means that the uncertainty is always rounded away from zero, whereas the setting `down` means that the uncertainty will be rounded toward zero. The setting `nearest` means that the uncertainty will be rounded to the nearest (either up or down), taking account of the setting of `round-half`. The standard setting is `nearest`.

uncertainty-separator `uncertainty-separator = ⟨separator⟩`

Stores the separator used between the main value and uncertainty when using the `compact` or `compact-marker` style setting for `uncertainty-mode`.

zero-decimal-as-symbol `zero-decimal-as-symbol = true|false`

Switch to determine if an entirely zero decimal part is replaced by a symbol. Does not apply if the decimal part is marked as entirely absent.

zero-symbol `zero-symbol = ⟨symbol⟩`

Material printed when a zero numerical component is replaced by a symbol.

5 Printing quantities

This submodule is focussed on providing controlled printing for numbers and units. Key to this is control of font: conventions for printing quantities mean that the exact nature of the output is important. At the same time, this module provides flexibility for the user in terms of which aspects of the font are responsive to the surrounding general text. Printing material may also take place in text or math mode.

The printing routines assume that normal LaTeX 2_ε font selection commands are available, in particular

- `\bfseries`,
- `\mathrm`,
- `\mathversion`,
- `\fontfamily`,
- `\fontseries`,
- `\fontshape`,
- `\familydefault`,
- `\seriesdefault`,
- `\shapedefault` and
- `\selectfont`.

It also requires the standard LaTeX 2_ε kernel commands

- `\ensuremath`,
- `\mbox`,
- `\textsubscript` and
- `\textsuperscript`

for printing in text mode. The following packages are also required to provide the functionality detailed.

- color: support for color using `\textcolor`
- textcomp: `\textminus`, `\textpm`,
- `\texttimes` and `\textcenteredperiod` for printing in text mode
- amstext: the `\text` command for printing in text mode

For detection of math mode fonts, as well as `\mathrm`, the existence of `\symoperators` is assumed; other math font commands are not *required* to exist.

`\siunitx_print_number:n`
`\siunitx_print_number:(V|e)`
`\siunitx_print_unit:n`
`\siunitx_print_unit:(V|e|o)`

`\siunitx_print_number:n {⟨material⟩}`
`\siunitx_print_unit:n {⟨material⟩}`

Prints the ⟨*material*⟩ according the prevailing settings for the submodule as applicable to the ⟨*type*⟩ of content (`number` or `unit`). The ⟨*material*⟩ should comprise normal LaTeX mark-up for numbers or units. In particular, units will typically use `\mathrm` to indicate material to be printed in the current upright roman font, and `^` and `_` will typically be used to indicate super- and subscripts, respectively. These elements will be correctly handled when printing for example using `\mathsf` in math mode, or using only text fonts. No printing takes place if the `\material` is entirely empty after a single expansion.

\siunitx_print_match:n `\siunitx_print_match:n {⟨material⟩}`
\siunitx_print_math:n `\siunitx_print_math:n {⟨material⟩}`
\siunitx_print_text:n `\siunitx_print_text:n {⟨material⟩}`

Prints the ⟨*material*⟩ as described for `\siunitx_print_…:n` but with a fixed text or math mode output. The printing does *not* set color (which is managed on a `unit`/`number` basis), but otherwise sets the font as described above. The `match` function uses either the prevailing math or text mode. No printing takes place if the `\material` is entirely empty after a single expansion.

color `color = ⟨color⟩`

Color to apply to printed output: the latter should be a named color defined for use with `\textcolor`. The standard setting is empty (no color).

mode `mode = match|math|text`

Selects which mode (math or text) the output is printed in: a choice from the options `match`, `math` or `text`. The option `match` matches the mode prevailing at the point `\siunitx_print_…:n` is called. The `math` and `text` options choose the relevant TEX mode for printing. The standard setting is `math`.

number-color `number-color = ⟨color⟩`

Color to apply to numbers in output: the latter should be a named color defined for use with `\textcolor`. The standard setting is empty (no color).

number-mode `number-mode = match|math|text`

Selects which mode (math or text) the numbers are printed in: a choice from the options `match`, `math` or `text`. The option `match` matches the mode prevailing at the point `\siunitx_prin_number:n` is called. The `math` and `text` options choose the relevant TEX mode for printing. The standard setting is `math`.

propagate-math-font `propagate-math-font = true|false`

Switch to determine if the currently-active math font is applied within printed output. This is relevant only when `\siunitx_print_…:n` is called from within math mode: in text mode there is not active math font. When not active, math mode material will be typeset using standard math mode fonts without any changes being made to the supplied argument. The standard setting is `false`.

reset-math-version `reset-math-version = true|false`

Switch to determine whether the active `\mathversion` is reset to `normal` when printing in math mode. Note that math version is typically used to select `\boldmath`, though it is also be used by *e.g.* sansmath. The standard setting is `true`.

reset-text-family `reset-text-family = true|false`

Switch to determine whether the active text family is reset to `\rmfamily` when printing in text mode. The standard setting is `true`.

`reset-text-series` `reset-text-series = true|false`

Switch to determine whether the active text series is reset to `\mdseries` when printing in text mode. The standard setting is `true`.

`reset-text-shape` `reset-text-shape = true|false`

Switch to determine whether the active text shape is reset to `\upshape` when printing in text mode. The standard setting is `true`.

`text-family-to-math` `text-family-to-math = true|false`

Switch to determine if the family of the current text font should be applied (where possible) to printing in math mode. The standard setting is `false`.

`text-font-command` `text-font-command =` ⟨*cmd*⟩

Command applied to text during output, inserted after any reset of font set-up. This can therefore be used to apply non-standard font set up when printing in text mode. The standard setting is empty.

`text-series-to-math` `text-series-to-math = true|false`

Switch to determine if the weight of the current text font should be applied (where possible) to printing in math mode. This is achieved by setting the `\mathversion`, and so will override `reset-math-version`. The mappings between text and math weight are set . The standard setting is `false`.

`text-subscript-command` `text-subscript-command =` ⟨*cmd*⟩
`text-superscript-command` `text-superscript-command =` ⟨*cmd*⟩

Sets the command used when printing material in sub- or superscript positions in text mode. The standard settings are `\textsubscript` and `\textsuperscript`, respectively.

`unit-color` `unit-color =` ⟨*color*⟩

Color to apply to units in output: the latter should be a named color defined for use with `\textcolor`. The standard setting is empty (no color).

`unit-mode` `unit-mode = match|math|text`

Selects which mode (math or text) units are printed in: a choice from the options `match`, `math` or `text`. The option `match` matches the mode prevailing at the point `\siunitx_print_…:n` is called. The `math` and `text` options choose the relevant TEX mode for printing. The standard setting is `math`.

series-version-mapping `series-version-mapping / ⟨weight⟩ = ⟨version⟩`

Defines how siunitx maps from text font weight to math font version. The pre-defined weights are those used as-standard by autoinst:

- `ul`
- `el`
- `l`
- `sl`
- `m`
- `sb`
- `b`
- `eb`
- `ub`

As standard, the `m` weight maps to `normal` math version whilst all of the `b` weights map to `bold` and all of the `l` weights map to `light`.

6 Quantities

This submodule is focussed on providing controlled printing for quantities: the combination of a number and a unit. It largely builds on the submodules siunitx-number and siunitx-unit. A small number of adjustments are made to standard set up in the latter to reflect additional functionality added here.

`\siunitx_quantity:nn` `\siunitx_quantity:nn {⟨number⟩} {⟨unit⟩}`

Parses the ⟨*number*⟩ and the ⟨*unit*⟩ as detailed for `\siunitx_number_parse:nN` and `\siunitx_unit_format:nN`, the prints the results using `\siunitx_print_unit:n`.

`\siunitx_quantity_print:nn` `\siunitx_quantity_print:nn {⟨number⟩} {⟨unit⟩}`
`\siunitx_quantity_print:(nV|VV|eV)`

A low-level function which prints the quantity directly: there is no processing applied to either the ⟨*number*⟩ or ⟨*unit*⟩. The two parts are printed using `\siunitx_print_unit:n` and appropriate spacing and break-prevention is applied.

`\l_siunitx_quantity_prefix_mode_str`

The active mode for prefix handling: one of `combine-exponent`, `extract-exponent` or `input`.

allow-quantity-breaks `allow-quantity-breaks = true|false`

Specifies whether breaks are permitted between units. The standard setting is `false`.

prefix-mode `prefix-mode = combine-exponent|extract-exponent|input`

Selects the method used for producing prefixes: a choice from the options `combine-exponent`, `extract-exponent` and `input`. The option `combine-exponent` combines any exponent from the number with the prefix of the first unit, and prints the updated prefix. The option `extract-exponent` removes all prefixes from the unit, and combines them with the exponent of number. The option `input` prints prefixes and exponent as given in the source. The standard setting is `input`.

quantity-product `quantity-product =` ⟨*tokens*⟩

The product marker used between a number and the unit. The standard setting is `\,`.

separate-uncertainty-units `separate-uncertainty-units = bracket|repeat|single`

Specifies how units are applied when a separated uncertainty is present: a choice from `bracket`, `repeat` and `single`. The option `bracket` places brackets around the number, with the unit given after these. The option `repeat` means that the unit it printed with the main value and with the uncertainty. When `single` is set, the unit is printed only once and no brackets are applied. The standard setting is `bracket`.

7 Formatting sexagesimal values

The generic sexagesimal formatting code is designed for values which have three components, each related by a factor of 60 to the next higher component.

`\siunitx_sexagesimal:n` `\siunitx_sexagesimal:n {`⟨*decimal*⟩`}`
`\siunitx_sexagesimal:e` `\siunitx_sexagesimal:nnn {`⟨*major*⟩`} {`⟨*intermediate*⟩`} {`⟨*minor*⟩`}`
`\siunitx_sexagesimal:nnn`
`\siunitx_sexagesimal:eee`

Typeset the ⟨*sexagesimal*⟩ value (which may be given as separate ⟨*major*⟩, ⟨*intermediate*⟩ and ⟨*minor*⟩ components). The ⟨*sexagesimal*⟩ (or components) may be given as expressions. The ⟨*sexagesimal*⟩ should be a number as understood by `\siunitx_format_number:nN`, with no uncertainty, exponent or imaginary part. The unit symbols for ⟨*major*⟩, ⟨*intermediate*⟩, ⟨*minor*⟩ are set by the options described below.

sexagesimal-mode `sexagesimal-mode =` ⟨*choice*⟩

Selects how sexagesimal values are formatted: a choice from the options `parts`, `decimal` and `input`. The option `components` means that sexagesimal values will always be typeset in components (major, intermediate, minor) format, whilst `decimal` means that angles are typeset as a single decimal value. The `input` setting means that the input format (*i.e.* difference between `\siunitx_sexagesimal:n` and `\siunitx_sexagesimal:nnn`) is maintained. The standard setting is `input`.

fill-sexagesimal-major `fill-sexagesimal-major = true|false`
fill-sexagesimal-intermediate
fill-sexagesimal-minor

Determines whether a missing component is zero-filled when printing an arc. The standard setting is `false`.

sexagesimal-separator `sexagesimal-separator = ⟨separator⟩`

Inserted between component parts (major, intermediate and minor components). The standard setting is empty.

sexagesimal-unit-major, sexagesimal-unit-intermediate, sexagesimal-unit-minor `sexagesimal-unit-major = ⟨unit⟩`

Sets the unit symbol(s) used for major, intermediate and minor components, respectively. The standard settings are empty.

sexagesimal-unit-over-decimal `sexagesimal-unit-over-decimal = true|false`

Determines if the component separator is printed over the decimal marker. The standard setting is `false`.

8 Formatting angles

\siunitx_angle:n, \siunitx_angle:e, \siunitx_angle:nnn, \siunitx_angle:eee

`\siunitx_angle:n {⟨angle⟩}`
`\siunitx_angle:nnn {⟨degrees⟩} {⟨minutes⟩} {⟨seconds⟩}`

Typeset the ⟨*angle*⟩ (which may be given as separate ⟨*degree*⟩, ⟨*minute*⟩ and ⟨*second*⟩ components). The ⟨*angle*⟩ (or components) may be given as expressions. The ⟨*angle*⟩ should be a number as understood by `\siunitx_format_number:nN`, with no uncertainty, exponent or imaginary part. The unit symbols for degrees, minutes and seconds are `\degree`, `\arcminute` and `\arcsecond`, respectively.

angle-mode `angle-mode = ⟨choice⟩`

Selects how angles are formatted: a choice from the options `arc`, `decimal` and `input`. The option `arc` means that angles will always be typeset in arc (degree, minute, second) format, whilst `decimal` means that angles are typeset as a single decimal value. The `input` setting means that the input format (*i.e.* difference between `\siunitx_angle:n` and `\siunitx_angle:nnn`) is maintained. The standard setting is `input`.

angle-symbol-degree, angle-symbol-minute, angle-symbol-second `angle-symbol-degree = ⟨symbol⟩`

Sets the symbol used for arc degrees, minutes or seconds, respectively.

angle-symbol-over-decimal `angle-symbol-over-decimal = true|false`

Determines if the arc separator is printed over the decimal marker, a format used in astronomy. The standard setting is `false`.

angle-separator `angle-separator = ⟨separator⟩`

Inserted between angle parts (degree, minute and second components). The standard setting is empty.

fill-angle-degrees, fill-angle-minutes, fill-angle-seconds `fill-arc-degrees = true|false`

Determines whether a missing angle part is zero-filled when printing an arc. The standard setting is `false`.

9 Formatting durations

`\siunitx_duration:n` `\siunitx_duration:e` `\siunitx_duration:nnn` `\siunitx_duration:eee`

```
\siunitx_duration:n {⟨decimal⟩}
\siunitx_duration:nnn {⟨hours⟩} {⟨minutes⟩} {⟨seconds⟩}
```

Typeset the ⟨*duration*⟩ (which may be given as separate ⟨*hour*⟩, ⟨*minute*⟩ and ⟨*second*⟩ components). The ⟨*duration*⟩ (or components) may be given as expressions. The ⟨*duration*⟩ should be a number as understood by `\siunitx_format_number:nN`, with no uncertainty, exponent or imaginary part. The unit symbols for hours, minutes and seconds are `\degree`, `\arcminute` and `\arcsecond`, respectively.

`component-separator`

```
component-separator = ⟨separator⟩
```

Inserted between arc parts (degree, minute and second components). The standard setting is `\,`.

`duration-mode`

```
duration-mode = ⟨choice⟩
```

Selects how durations are formatted: a choice from the options `component`, `decimal` and `input`. The option `component` means that durations will always be typeset in component (hour, minute, second) format, whilst `decimal` means that angles are typeset as a single decimal value. The `input` setting means that the input format (*i.e.* difference between `\siunitx_duration:n` and `\siunitx_duration:nnn`) is maintained. The standard setting is `input`.

`duration-unit-hour` `duration-unit-minute` `duration-unit-second`

```
duration-unit-hour = ⟨unit⟩
```

Sets the symbol used for component hours, minutes or seconds, respectively.

`fill-duration-hours` `fill-duration-minutes` `fill-duration-seconds`

```
fill-duration-hours = true|false
```

Determines whether a missing part is zero-filled when printing an duration in component form. The standard setting is `false`.

10 Numbers in tables

This submodule is concerned with formatting numbers in table cells or similar fixed-width contexts. The main function, `\siunitx_cell_begin:w`, is designed to work with the normal LaTeX 2$_\varepsilon$ tabular cell construct featuring `\ignorespaces`. Therefore, if used outside of a LaTeX 2$_\varepsilon$ tabular, it is necessary to provide this token.

`\siunitx_cell_begin:w` `\siunitx_cell_end:`

```
\siunitx_cell_begin:w ⟨preamble⟩ \ignorespaces
⟨content⟩
\siunitx_cell_end:
```

Collects the ⟨*preamble*⟩ and ⟨*content*⟩ tokens, and determines if it is text or a number (as parsed by `\siunitx_number_parse:nN`). It produces output of a fixed width suitable for alignment in a table, although it is not *required* that the code is used within a cell. Note that `\ignorespaces` must occur in the "cell": it marks the end of the TeX `\halign` template.

`table-align-comparator` `table-align-comparator = true|false`

Switch which determines whether alignment of comparators is attempted within table cells. The standard setting is `true`.

`table-align-exponent` `table-align-exponent = true|false`

Switch which determines whether alignment of exponents is attempted within table cells. The standard setting is `true`.

`table-align-text-after` `table-align-text-after = true|false`

Switch which determines whether alignment of text falling after a number is attempted within table cells. The standard setting is `true`.

`table-align-text-before` `table-align-text-before = true|false`

Switch which determines whether alignment of text falling before a number is attempted within table cells. The standard setting is `true`.

`table-align-uncertainty` `table-align-uncertainty = true|false`

Switch which determines whether alignment of separated uncertainty values is attempted within table cells. The standard setting is `true`.

`table-alignment` `table-alignment = center|left|right`

Selects the alignment of all tabular content with the margins of the table cell (or other boundary). See also `table-number-alignment` and `table-text-alignment`. The standard setting is `center`.

`table-alignment-mode` `table-alignment-mode = format|marker|none`

Selects the method used to align numbers with the desired position in the cell (set by `table-alignment`). When set to `format`, a dedicated amount of space is calculated from the `table-format`. When `marker` is selected, alignment is carried out symmetrically around the decimal marker. Finally, `none` switches off all alignment: numbers are parsed and formatted but with no attempt at placement within the cell. The standard setting is `marker`.

`table-auto-round` `table-auto-round = true|false`

Switch which determines whether numbers are rounded to fit within the `table-format` specification (if possible). The standard setting is `false`.

`table-column-width` `table-column-width =` ⟨*width*⟩

Sets the width of the table column used for numbers. This is only used when `table-fixed-width` is `true`.

`table-fixed-width` `table-fixed-width = true|false`

Switch which determines whether a fixed-width column is used for numbers in tables. When `true`, the width is taken from `table-column-width`. The standard setting is `false`.

table-format `table-format = ⟨format⟩`

Describes the amount of space that should be reserved when `table-alignment-mode` is set to `format`. The ⟨*format*⟩ takes the same general form as input for a table cell, with the numerical parts describing how many digits to reserve space for. For example, `1.2e3` would allow space for one digit in the integer part, two in the decimal part and three in the exponent part. Signs can be allowed for using any valid input sign, so for example `+1.2 \pm 1.2` would allow for a sign, a number with one integer and two decimal digits and an uncertainty of the same size.

table-model-setup `table-model-setup = ⟨commands⟩`

Additional commands to be inserted when using the `table-format` to create a model for alignment of cells. Typically this will be used to handle variable-width fonts in columns. The standard setting is empty.

table-number-alignment `table-number-alignment = center|left|right`

Selects the alignment of numerical content with the margins of the table cell (or other boundary). See also `table-alignment` and `table-text-alignment`. The standard setting is `center`.

table-text-alignment `table-text-alignment = center|left|none|right`

Selects the alignment of non-numerical content with the margins of the table cell (or other boundary). See also `table-alignment` and `table-number-alignment`. Notice the additional support for `none` here. The standard setting is `center`.

11 Parsing and formatting units

This submodule is dedicated to formatting physical units. The main function, `\siunitx_unit_format:nN`, takes user input specifying physical units and converts it into a formatted token list suitable for typesetting in math mode. While the formatter will deal correctly with "literal" user input, the key strength of the module is providing a method to describe physical units in a "symbolic" manner. The output format of these symbolic units can then be controlled by a number of key–value options made available by the module.

A small number of LaTeX 2_ε math mode commands are assumed to be available as part of the formatted output. The `\mathchoice` command (normally the TeX primitive) is needed when using different settings for inline and display `per-mode`. The commands `\frac`, `\mathrm`, `\mbox`, `\␣` and `\,` are used by the standard module settings. For the display of colored (highlighted) and cancelled units, the commands `\textcolor` and `\cancel` are assumed to be available.

11.1 Formatting units

\siunitx_unit_format:nN
\siunitx_unit_format:VN

\siunitx_unit_format:nN {⟨*units*⟩} ⟨*tl var*⟩

This function converts the input ⟨*units*⟩ into a processed ⟨*tl var*⟩ which can then be inserted in math mode to typeset the material. Where the ⟨*units*⟩ are given in symbolic form, described elsewhere, this formatting process takes place in two stages: the ⟨*units*⟩ are parsed into a structured form before the generation of the appropriate output form based on the active settings. When the ⟨*units*⟩ are given as literals, processing is minimal: the characters `.` and `~` are converted to unit products (boundaries). In both cases, the result is a series of tokens intended to be typeset in math mode with appropriate choice of font for typesetting of the textual parts.

For example,

```
\siunitx_unit_format:nN { \kilo \metre \per \second } \l_tmpa_tl
```

will, with standard settings, result in `\l_tmpa_tl` being set to

```
\mathrm{km}\,\mathrm{s}^{-1}
```

\siunitx_unit_format_extract_prefixes:nNN

\siunitx_unit_format_extract_prefixes:nNN {⟨*units*⟩} ⟨*tl var*⟩ ⟨*fp var*⟩

This function formats the ⟨*units*⟩ in the same way as described for `\siunitx_unit_format:nN`. When the input is given in symbolic form, any decimal unit prefixes will be extracted and the overall power of ten that these represent will be stored in the ⟨*fp var*⟩.

For example,

```
\siunitx_unit_format_extract_prefixes:nNN { \kilo \metre \per \second }
  \l_tmpa_tl \l_tmpa_fp
```

will, with standard settings, result in `\l_tmpa_tl` being set to

```
\mathrm{m}\,\mathrm{s}^{-1}
```

with `\l_tmpa_fp` taking value 3. Note that the latter is a floating point variable: it is possible for non-integer values to be obtained here.

\siunitx_unit_format_combine_exponent:nnN

\siunitx_unit_format_combine_exponent:nnN {⟨*units*⟩} {⟨*exponent*⟩} ⟨*tl var*⟩

This function formats the ⟨*units*⟩ in the same way as described for `\siunitx_unit_format:nN`. The ⟨*exponent*⟩ is combined with any prefix for the *first* unit of the ⟨*units*⟩, and an updated prefix is introduced.

For example,

```
\siunitx_unit_format_combine_exponent:nnN { \metre \per \second }
  { 3 } \l_tmpa_tl
```

will, with standard settings, result in `\l_tmpa_tl` being set to

```
\mathrm{km}\,\mathrm{s}^{-1}
```

\siunitx_unit_format_multiply:nnN
\siunitx_unit_format_multiply_extract_prefixes:nnNN
\siunitx_unit_format_multiply_combine_exponent:nnnN

\siunitx_unit_format_multiply:nnN {⟨*units*⟩} {⟨*factor*⟩} ⟨*tl var*⟩
\siunitx_unit_format_multiply_extract_prefixes:nnNN {⟨*units*⟩} {⟨*factor*⟩} ⟨*tl var*⟩ ⟨*fp var*⟩
\siunitx_unit_format_multiply_combine_exponent:nnnN {⟨*units*⟩} {⟨*factor*⟩} {⟨*exponent*⟩} ⟨*tl var*⟩

These function formats the ⟨*units*⟩ in the same way as described for `\siunitx_unit_format:nN`. The units are multiplied by the ⟨*factor*⟩, and further processing takes place as previously described.

For example,

```
\siunitx_unit_format_multiply:nnN { \metre \per \second }
  { 3 } \l_tmpa_tl
```

will, with standard settings, result in `\l_tmpa_tl` being set to

```
\mathrm{km}^{3}\,\mathrm{s}^{-3}
```

11.2 Defining symbolic units

\siunitx_declare_prefix:Nnn
\siunitx_declare_prefix:Nne

\siunitx_declare_prefix:Nnn ⟨*prefix*⟩ {⟨*power*⟩} {⟨*symbol*⟩}

Defines a symbolic ⟨*prefix*⟩ (which should be a control sequence such as `\kilo`) to be converted by the parser to the ⟨*symbol*⟩. The latter should consist of literal content (*e.g.* `k`). In literal mode the ⟨*symbol*⟩ will be typeset directly. The prefix should represent an integer ⟨*power*⟩ of 10, and this information may be used to convert from one or more ⟨*prefix*⟩ symbols to an overall power applying to a unit. See also `\siunitx_declare_prefix:Nn`.

\siunitx_declare_prefix:Nn

\siunitx_declare_prefix:Nn ⟨*prefix*⟩ {⟨*symbol*⟩}

Defines a symbolic ⟨*prefix*⟩ (which should be a control sequence such as `\kilo`) to be converted by the parser to the ⟨*symbol*⟩. The latter should consist of literal content (*e.g.* `k`). In literal mode the ⟨*symbol*⟩ will be typeset directly. In contrast to `\siunitx_declare_prefix:Nnn`, there is no assumption about the mathematical nature of the ⟨*prefix*⟩, *i.e.* the prefix may represent a power of any base. As a result, no conversion of the ⟨*prefix*⟩ to a numerical power will be possible.

\siunitx_declare_power:NNn

\siunitx_declare_power:NNn ⟨*pre-power*⟩ ⟨*post-power*⟩ {⟨*value*⟩}

Defines *two* symbolic ⟨*powers*⟩ (which should be control sequences such as `\squared`) to be converted by the parser to the ⟨*value*⟩. The latter should be an integer or floating point number in the format defined for l3fp. Powers may precede a unit or be give after it: both forms are declared at once, as indicated by the argument naming. In literal mode, the ⟨*value*⟩ will be applied as a superscript to either the next token in the input (for the ⟨*pre-power*⟩) or appended to the previously-typeset material (for the ⟨*post-power*⟩).

`\siunitx_declare_qualifier:Nn` `\siunitx_declare_qualifier:Nn ⟨qualifier⟩ {⟨meaning⟩}`

Defines a symbolic ⟨*qualifier*⟩ (which should be a control sequence such as `\catalyst`) to be converted by the parser to the ⟨*meaning*⟩. The latter should consist of literal content (*e.g.* `cat`). In literal mode the ⟨*meaning*⟩ will be typeset following a space after the unit to which it applies.

`\siunitx_declare_unit:Nn`
`\siunitx_declare_unit:Ne`
`\siunitx_declare_unit:Nnn`
`\siunitx_declare_unit:Nen`

`\siunitx_declare_unit:Nn ⟨unit⟩ {⟨meaning⟩}`
`\siunitx_declare_unit:Nnn ⟨unit⟩ {⟨meaning⟩} {⟨options⟩}`

Defines a symbolic ⟨*unit*⟩ (which should be a control sequence such as `\kilogram`) to be converted by the parser to the ⟨*meaning*⟩. The latter may consist of literal content (*e.g.* `kg`), other symbolic unit commands (*e.g.* `\kilo\gram`) or a mixture of the two. In literal mode the ⟨*meaning*⟩ will be typeset directly. The version taking an ⟨*options*⟩ argument may be used to support per-unit options: these are applied at the top level or using `\siunitx_unit_options_apply:n`.

`\l_siunitx_unit_font_tl` The font function which is applied to the text of units when constructing formatted units: set by `font-command`.

`\l_siunitx_unit_fraction_tl`

The fraction function which is applied when constructing fractional units: set by `fraction-command`.

`\l_siunitx_unit_symbolic_seq`

This sequence contains all of the symbolic names defined: these will be in the form of control sequences such as `\kilogram`. The order of the sequence is unimportant. This includes prefixes and powers as well as units themselves.

`\l_siunitx_unit_seq` This sequence contains all of the symbolic *unit* names defined: these will be in the form of control sequences such as `\kilogram`. In contrast to `\l_siunitx_unit_symbolic_seq`, it *only* holds units themselves

11.3 Per-unit options

`\siunitx_unit_options_declare:Nn` `\siunitx_unit_options_declare:Nn ⟨unit⟩ {⟨options⟩}`

Declares that the ⟨*options*⟩ should be applied with the ⟨*unit*⟩ is used. The ⟨*options*⟩ stored override any saved when the unit was declared. This function is intended for adjusting options when the unit command is otherwise unchanged.

`\siunitx_unit_options_apply:n` `\siunitx_unit_options_apply:n {⟨unit(s)⟩}`

Applies any unit-specific options set up using `\siunitx_declare_unit:Nnn`. This allows there use outside of unit formatting, for example to influence spacing in quantities. The options are applied only once at a given group level, which allows for user override *via* `\keys_set:nn { siunitx } { ... }`.

11.4 Units in (PDF) strings

\siunitx_unit_pdfstring_context:

```
\group_begin:
\siunitx_unit_pdfstring_context:
⟨Expansion context⟩ ⟨units⟩
\group_end:
```

Sets symbol unit macros to generate text directly. This is needed in expansion contexts where units must be converted to simple text. This function is itself not expandable, so must be using within a surrounding group as show in the example.

11.5 Pre-defined symbolic unit components

The unit parser is defined to recognise a number of pre-defined units, prefixes and powers, and also interpret a small selection of "generic" symbolic parts.

Broadly, the pre-defined units are those defined by the BIPM in the documentation for the *International System of Units* (SI) [1]. As far as possible, the names given to the command names for units are those used by the BIPM, omitting spaces and using only ASCII characters. The standard symbols are also taken from the same documentation. In the following documentation, the order of the description of units broadly follows the SI Brochure.

\kilogram
\metre
\meter
\mole
\kelvin
\candela
\second
\ampere

The base units as defined in the SI Brochure [2]. Notice that `\meter` is defined as an alias for `\metre` as the former spelling is common in the US (although the latter is the official spelling).

\gram

The base unit `\kilogram` is defined using an SI prefix: as such the (derived) unit `\gram` is required by the module to correctly produce output for the `\kilogram`.

\quecto
\ronto
\yocto
\zepto
\atto
\femto
\pico
\nano
\micro
\milli
\centi
\deci
\deca
\deka
\hecto
\kilo
\mega
\giga
\tera
\peta
\exa
\zetta
\yotta
\ronna
\quetta

Prefixes, all of which are integer powers of 10: the powers are stored internally by the module and can be used for conversion from prefixes to their numerical equivalent. These prefixes are documented in Section 3.1 of the SI Brochure.

Note that the `\kilo` prefix is required to define the base `\kilogram` unit. Also note the two spellings available for `\deca`/`\deka`.

\becquerel
\degreeCelsius
\coulomb
\farad
\gray
\hertz
\henry
\joule
\katal
\lumen
\lux
\newton
\ohm
\pascal
\radian
\siemens
\sievert
\steradian
\tesla
\volt
\watt
\weber

The defined SI units with defined names and symbols, as given in Table 4 of the SI Brochure. Notice that the names of the units are lower case with the exception of `\degreeCelsius`, and that this unit name includes "degree".

`\astronomicalunit`
`\bel`
`\dalton`
`\day`
`\decibel`
`\electronvolt`
`\hectare`
`\hour`
`\litre`
`\liter`
`\neper`
`\minute`
`\tonne`

Units accepted for use with the SI: here `\minute` is a unit of time not of plane angle. These units are taken from Table 8 of the SI Brochure.

For the unit `\litre`, both `l` and `L` are listed as acceptable symbols: the latter is the standard setting of the module. The alternative spelling `\liter` is also given for this unit for US users (as with `\metre`, the official spelling is "re").

`\arcminute`
`\arcsecond`
`\degree`

Units for plane angles accepted for use with the SI: to avoid a clash with units for time, here `\arcminute` and `\arcsecond` are used in place of `\minute` and `\second`. These units are taken from Table 8 of the SI Brochure.

`\percent`

The mathematical concept of percent, usable with the SI as detailed in Section 5.4.7 of the SI Brochure.

`\square` `\square` ⟨*prefix*⟩ ⟨*unit*⟩
`\cubic` `\cubic` ⟨*prefix*⟩ ⟨*unit*⟩

Pre-defined unit powers which apply to the next ⟨*prefix*⟩/⟨*unit*⟩ combination.

`\squared` ⟨*prefix*⟩ ⟨*unit*⟩ `\squared`
`\cubed` ⟨*prefix*⟩ ⟨*unit*⟩ `\cubed`

Pre-defined unit powers which apply to the preceding ⟨*prefix*⟩/⟨*unit*⟩ combination.

`\per` `\per` ⟨*prefix*⟩ ⟨*unit*⟩ ⟨*power*⟩

Indicates that the next ⟨*prefix*⟩/⟨*unit*⟩/⟨*power*⟩ combination is reciprocal, *i.e.* raises it to the power −1. This symbolic representation may be applied in addition to a `\power`, and will work correctly if the `\power` itself is negative. In literal mode `\per` will print a slash ("/").

`\cancel` `\cancel` ⟨*prefix*⟩ ⟨*unit*⟩ ⟨*power*⟩

Indicates that the next ⟨*prefix*⟩/⟨*unit*⟩/⟨*power*⟩ combination should be "cancelled out". In the parsed output, the entire unit combination will be given as the argument to a function `\cancel`, which is assumed to be available at a higher level. In literal mode, the same higher-level `\cancel` will be applied to the next token. It is the responsibility of the calling code to provide an appropriate definition for `\cancel` outside of the scope of the unit parser.

\highlight `\highlight {⟨color⟩} ⟨prefix⟩ ⟨unit⟩ ⟨power⟩`

Indicates that the next ⟨*prefix*⟩/⟨*unit*⟩/⟨*power*⟩ combination should be highlighted in the specified ⟨*color*⟩. In the parsed output, the entire unit combination will be given as the argument to a function `\textcolor`, which is assumed to be available at a higher level. In literal mode, the same higher-level `\textcolor` will be applied to the next token. It is the responsibility of the calling code to provide an appropriate definition for `\textcolor` outside of the scope of the unit parser.

\of `⟨prefix⟩ ⟨unit⟩ ⟨power⟩ \of {⟨qualifier⟩}`

Indicates that the ⟨*qualifier*⟩ applies to the current ⟨*prefix*⟩/⟨*unit*⟩/⟨*power*⟩ combination. In parsed mode, the display of the result will depend upon module options. In literal mode, the ⟨*qualifier*⟩ will be printed in parentheses following the preceding ⟨*unit*⟩ and a full-width space.

\raiseto `\raiseto {⟨power⟩} ⟨prefix⟩ ⟨unit⟩`
\tothe `⟨prefix⟩ ⟨unit⟩ \tothe {⟨power⟩}`

Indicates that the ⟨*power*⟩ applies to the current ⟨*prefix*⟩/⟨*unit*⟩ combination. As shown, `\raiseto` applies to the next ⟨*unit*⟩ whereas `\tothe` applies to the preceding unit. In literal mode the `\power` will be printed as a superscript attached to the next token (`\raiseto`) or preceding token (`\tothe`) as appropriate.

11.6 Key–value options for units

bracket-unit-denominator `bracket-unit-denominator = true|false`

Switch to determine whether brackets are added to the denominator part of a unit when printed using inline fractional form (with `per-mode` as `repeated-symbol` or `symbol`). The standard setting is `true`.

extract-mass-in-kilograms `extract-mass-in-kilograms = true|false`

Determines whether prefix extraction treats kilograms as a base unit; when set `false`, grams are used. The standard setting is `true`.

forbid-literal-units `forbid-literal-units = true|false`

Switch which determines if literal units are allowed when parsing is active; does not apply when `parse-units` is `false`.

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

Command used to create fractional output when `per-mode` is set to `fraction`. The standard setting is `\frac`.

inter-unit-product `inter-unit-product = ⟨separator⟩`

Inserted between unit combinations in parsed mode, and used to replace `.` and `~` in literal mode. The standard setting is `\,`.

`parse-units` `parse-units = true|false`

Determines whether parsing of unit symbols is attempted or literal mode is used directly. The standard setting is `true`.

`per-mode` `inline-per-mode` `display-per-mode` `per-mode = fraction|power|power-positive-first|repeated-symbol|single-symbol|symbol`

Selects how the negative powers (`\per`) are formatted: a choice from the options `fraction`, `power`, `power-positive-first`, `repeated-symbol`, `single-symbol` and `symbol`. The option `fraction` generates fractional output when appropriate using the command specified by the `fraction-command` option. The setting `power` uses reciprocal powers leaving the units in the order of input, while `power-positive-first` uses the same display format but sorts units such that the positive powers come before negative ones. The `symbol` setting uses a symbol (specified by `per-symbol`) between positive and negative powers, while `repeated-symbol` uses the same symbol but places it before *every* unit with a negative power (this is mathematically "wrong" but often seen in real work). The option `single-symbol` will use a symbol if exactly one is required (*i.e.* with a single negative power), and will otherwise use powers. The standard setting is `power`.

The `inline-...` and `display-...` settings take the same options and work in exactly the same way, but are restricted in where they apply. The `display` version only applies in display math contexts, and the `inline` version applies in all others.

`per-symbol` `per-symbol = ⟨symbol⟩`

Specifies the symbol to be used to denote negative powers when the option `per-mode` is set to `repeated-symbol` or `symbol`. The standard setting is `/`.

`per-symbol-script-correction` `per-symbol-script-correction = ⟨insert⟩`

Specifies the tokens used to correct spacing when the symbol set by `per-symbol` is immediately preceded by a superscript power. The standard setting is `\!`.

`power-half-as-sqrt` `power-half-as-sqrt = true|false`

Used to determine whether a power of exactly half is converted to `\sqrt` in the output. The standard setting is `false`.

`qualifier-mode` `qualifier-mode = bracket|combine|phrase|subscript`

Selects how qualifiers are formatted: a choice from the options `bracket`, `combine`, `phrase` and `subscript`. The option `bracket` wraps the qualifier in parenthesis, `combine` joins the qualifier with the unit directly, `phrase` joins the material using `qualifier-phrase` as a link, and `subscript` formats the qualifier as a subscript. The standard setting is `subscript`.

`qualifier-phrase` `qualifier-phrase = ⟨phrase⟩`

Defines the ⟨*phrase*⟩ used when `qualifier-mode` is set to `phrase`.

`sticky-per` `sticky-per = true|false`

Used to determine whether `\per` should be applied one a unit-by-unit basis (when `false`) or should apply to all following units (when `true`). The latter mode is somewhat akin conceptually to the TEX `\over` primitive. The standard setting is `false`.

unit-font-command `unit-font-command = ⟨command⟩`

Command applied to text during output of units: should be command usable in math mode for font selection. Notice that in a typical unit this does not (necessarily) apply to all output, for example powers or brackets. The standard setting is `\mathrm`.

References

[1] *The International System of Units (SI)*, https://www.bipm.org/en/measurement-units/.

[2] *SI base units*, https://www.bipm.org/en/measurement-units/si-base-units.

12 Abbreviations

`\A` `\pA` `\nA` `\uA` `\mA` `\kA`

Abbreviations for currents.

`\fg` `\pg` `\ng` `\ug` `\mg` `\g` `\kg`

Abbreviations for masses.

`\K`

Abbreviations for temperature.

`\m` `\pm` `\nm` `\um` `\mm` `\cm` `\dm` `\km`

Abbreviations for lengths.

`\s`
`\as`
`\fs`
`\ps`
`\ns`
`\us`
`\ms`
	Abbreviations for times.

`\Hz`
`\mHz`
`\kHz`
`\MHz`
`\GHz`
`\THz`
	Abbreviations for frequencies.

`\mol`
`\fmol`
`\pmol`
`\nmol`
`\umol`
`\mmol`
`\kmol`
	Abbreviations for moles.

`\V`
`\pV`
`\nV`
`\uV`
`\mV`
`\kV`
	Abbreviations for potentials.

`\hl`
`\l`
`\ml`
`\ul`
`\hL`
`\L`
`\mL`
`\uL`
	Abbreviations for volumes.

`\W`
`\nW`
`\uW`
`\mW`
`\kW`
`\MW`
`\GW`
	Abbreviations for powers.

`\kJ`
`\J`
`\mJ`
`\uJ`
`\eV`
`\meV`
`\keV`
`\MeV`
`\GeV`
`\TeV`
	Abbreviations for energies.

`\N`
`\mN`
`\kN`
`\MN`
	Abbreviations for forces.

`\Pa`
`\kPa`
`\MPa`
`\GPa`
	Abbreviations for pressures.

`\mohm`
`\kohm`
`\Mohm`
	Abbreviations for resistance.

`\F`
`\fF`
`\pF`
`\nF`
`\uF`
`\mF`
	Abbreviations for capacitance.

`\H`
`\fH`
`\pH`
`\nH`
`\uH`
`\mH`
	Abbreviations for inductance.

`\C`
`\nC` `\uC`
`\mC`
	Abbreviations for charge.

`\T` `\mT` `\uT` Abbreviations for magneticc field.

`\dB` Abbreviation for decibel.

`\kWh` Abbreviation for kilowatt–hours.

13 Additional binary units

This submodule provides binary units and prefixes. These are not formally part of the SI but are recommended by BIPM as units of information.

`\kibi` `\mebi` `\gibi` `\tebi` `\pebi` `\exbi` `\zebi` `\yobi` Prefixes, all of which are integer powers of 2: the powers are *not* stored or available for conversion.

`\bit` `\byte` Units for bits and bytes.

This submodule provides support for creating free-standing document commands for unit macros.

14 Creating units as document commands

`\siunitx_command_create:` `\siunitx_command_create:`

Maps over the list of know unit commands and creates the appropriate document command to support them, as controlled by the options below.

These options are all preamble-only.

`free-standing-units` `free-standing-units = true|false`

Switch to determine whether free standing document commands are created for symbolic units. This will include not only units themselves but also prefixes, *etc.* The standard setting is `false`.

`overwrite-commands` `overwrite-commands = true|false`

Switch to determine whether when creating free standing document commands, any existing document commands are overwritten. The standard setting is `false`.

`space-before-unit` `space-before-unit = true|false`

Switch to determine whether a space is inserted before free standing document commands. The standard setting is `false`.

`unit-optional-argument` `unit-optional-argument = true|false`

Switch to determine whether free standing document commands take an optional argument (a number). The standard setting is `false`.

`use-xspace` `use-xspace = true|false`

Switch to determine whether free standing document commands use the xparse package to insert space after the command names. The standard setting is `false`. When set `true`, the xparse package will be loaded at the start of the document if not already available.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

M

N

O

P

Q

R

S

T

U

V

W

Y

Z