

# siunitx-unit – Parsing and formatting units\*

Joseph Wright<sup>†</sup>

Released 2021-06-22

This submodule is dedicated to formatting physical units. The main function, `\siunitx_unit_format:nN`, takes user input specify 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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> math mode commands are assumed to be available as part of the formatted output. The `\mathchoice` command (normally the T<sub>E</sub>X primitive) is needed when using `per-mode = symbol-or-fraction`. 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.

## 1 Formatting units

---

`\siunitx_unit_format:nN`  
`\siunitx_unit_format:xN`

---

`\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}
```

---

\*This file describes v3.0.15, last revised 2021-06-22.

<sup>†</sup>E-mail: [joseph.wright@morningstar2.co.uk](mailto:joseph.wright@morningstar2.co.uk)

---

<code>\siunitx_unit_format_extract_prefixes:nnN</code>	<code>\siunitx_unit_format_extract_prefixes:nnN {&lt;units&gt;} &lt;tl var&gt; &lt;fp var&gt;</code>
--	--

---

This function formats the  $\langle units \rangle$  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  $\langle fp var \rangle$ .

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.

---

<code>\siunitx_unit_format_combine_exponent:nnN</code>	<code>\siunitx_unit_format_combine_exponent:nnN {&lt;units&gt;} &lt;exponent&gt; &lt;tl var&gt;</code>
--	--

---

This function formats the  $\langle units \rangle$  in the same way as described for `\siunitx_unit_format:nN`. The  $\langle exponent \rangle$  is combined with any prefix for the *first* unit of the  $\langle units \rangle$ , 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}
```

---

<code>\siunitx_unit_format_multiply:nnN</code>	<code>\siunitx_unit_format_multiply:nnN {&lt;units&gt;} &lt;factor&gt; &lt;tl var&gt;</code>
<code>\siunitx_unit_format_multiply_extract_prefixes:nnNN</code>	<code>{&lt;factor&gt;} &lt;tl var&gt;</code>
<code>\siunitx_unit_format_multiply_combine_exponent:nnnN</code>	<code>\siunitx_unit_format_multiply_extract_prefixes:nnNN {&lt;units&gt;} {&lt;factor&gt;} &lt;tl var&gt; &lt;fp var&gt; \siunitx_unit_format_multiply_combine_exponent:nnnN {&lt;units&gt;} {&lt;factor&gt;} {&lt;exponent&gt;} &lt;tl var&gt;</code>

---

These function formats the  $\langle units \rangle$  in the same way as described for `\siunitx_unit_format:nN`. The units are multiplied by the  $\langle factor \rangle$ , 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}
```

## 2 Defining symbolic units

---

<code>\siunitx_declare_prefix:Nnn</code>	<code>\siunitx_declare_prefix:Nnn &lt;prefix&gt; {&lt;power&gt;} {&lt;symbol&gt;}</code>
<code>\siunitx_declare_prefix:Nnx</code>	

---

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

---

<code>\siunitx_declare_prefix:Nn</code>	<code>\siunitx_declare_prefix:Nn &lt;prefix&gt; {&lt;symbol&gt;}</code>
---	---

---

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

---

<code>\siunitx_declare_power:NNn</code>	<code>\siunitx_declare_power:NNn &lt;pre-power&gt; &lt;post-power&gt; {&lt;value&gt;}</code>
---	--

---

Defines *two* symbolic  $\langle powers \rangle$  (which should be control sequences such as `\squared`) to be converted by the parser to the  $\langle value \rangle$ . 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  $\langle value \rangle$  will be applied as a superscript to either the next token in the input (for the  $\langle pre-power \rangle$ ) or appended to the previously-typeset material (for the  $\langle post-power \rangle$ ).

---

<code>\siunitx_declare_qualifier:Nn</code>	<code>\siunitx_declare_qualifier:Nn &lt;qualifier&gt; {&lt;meaning&gt;}</code>
--	--

---

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

---

<code>\siunitx_declare_unit:Nn</code>	<code>\siunitx_declare_unit:Nn &lt;unit&gt; {&lt;meaning&gt;}</code>
<code>\siunitx_declare_unit:Nx</code>	<code>\siunitx_declare_unit:Nnn &lt;unit&gt; {&lt;meaning&gt;} {&lt;options&gt;}</code>
<code>\siunitx_declare_unit:Nnn</code>	
<code>\siunitx_declare_unit:Nxn</code>	

---

Defines a symbolic  $\langle unit \rangle$  (which should be a control sequence such as `\kilogram`) to be converted by the parser to the  $\langle meaning \rangle$ . 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  $\langle meaning \rangle$  will be typeset directly. The version taking an  $\langle options \rangle$  argument may be used to support per-unit options: these are applied at the top level or using `\siunitx_unit_options_apply:n`.

---

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

---

---

**`\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

### 3 Per-unit options

---

**`\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 their 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 over-ride *via* `\keys_set:nn { siunitx } { ... }`.

### 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 used within a surrounding group as shown in the example.

### 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) [?]. 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.

---

<code>\kilogram</code>	The base units as defined in the SI Brochure [?]. Notice that <code>\meter</code> is defined as an alias for <code>\metre</code> as the former spelling is common in the US (although the latter is the official spelling).
<code>\metre</code>	
<code>\meter</code>	
<code>\mole</code>	
<code>\kelvin</code>	
<code>\candela</code>	
<code>\second</code>	
<code>\ampere</code>	

---



---

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

---



---

<code>\yocto</code>	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.
<code>\zepto</code>	
<code>\atto</code>	Note that the <code>\kilo</code> prefix is required to define the base <code>\kilogram</code> unit. Also note the two spellings available for <code>\deca</code> / <code>\deka</code> .
<code>\femto</code>	
<code>\pico</code>	
<code>\nano</code>	
<code>\micro</code>	
<code>\milli</code>	
<code>\centi</code>	
<code>\deci</code>	
<code>\deca</code>	
<code>\deka</code>	
<code>\hecto</code>	
<code>\kilo</code>	
<code>\mega</code>	
<code>\giga</code>	
<code>\tera</code>	
<code>\peta</code>	
<code>\exa</code>	
<code>\zetta</code>	
<code>\yotta</code>	

---

<hr/>	
<code>\becquerel</code>	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 <code>\degreeCelsius</code> , and that this unit name includes “degree”.
<code>\degreeCelsius</code>	
<code>\coulomb</code>	
<code>\farad</code>	
<code>\gray</code>	
<code>\hertz</code>	
<code>\henry</code>	
<code>\joule</code>	
<code>\katal</code>	
<code>\lumen</code>	
<code>\lux</code>	
<code>\newton</code>	
<code>\ohm</code>	
<code>\pascal</code>	
<code>\radian</code>	
<code>\siemens</code>	
<code>\sievert</code>	
<code>\steradian</code>	
<code>\tesla</code>	
<code>\volt</code>	
<code>\watt</code>	
<code>\weber</code>	
<hr/>	

<hr/>	
<code>\astronomicalunit</code>	Units accepted for use with the SI: here <code>\minute</code> is a unit of time not of plane angle. These units are taken from Table 8 of the SI Brochure.
<code>\bel</code>	
<code>\dalton</code>	For the unit <code>\litre</code> , both l and L are listed as acceptable symbols: the latter is the standard setting of the module. The alternative spelling <code>\liter</code> is also given for this unit for US users (as with <code>\metre</code> , the official spelling is “re”).
<code>\day</code>	
<code>\decibel</code>	
<code>\electronvolt</code>	
<code>\hectare</code>	
<code>\hour</code>	
<code>\litre</code>	
<code>\liter</code>	
<code>\neper</code>	
<code>\minute</code>	
<code>\tonne</code>	
<hr/>	

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

<hr/>	
<code>\percent</code>	The mathematical concept of percent, usable with the SI as detailed in Section 5.4.7 of the SI Brochure.
<hr/>	

<hr/>	
<code>\square</code>	<code>\square</code> <i>&lt;prefix&gt;</i> <i>&lt;unit&gt;</i>
<code>\cubic</code>	<code>\cubic</code> <i>&lt;prefix&gt;</i> <i>&lt;unit&gt;</i>
<hr/>	
	Pre-defined unit powers which apply to the next <i>&lt;prefix&gt;/&lt;unit&gt;</i> combination.

<hr/>	<code>\squared</code>	<code>&lt;prefix&gt; &lt;unit&gt; \squared</code>
<hr/>	<code>\cubed</code>	<code>&lt;prefix&gt; &lt;unit&gt; \cubed</code>

Pre-defined unit powers which apply to the preceding `<prefix>/<unit>` combination.

<hr/>	<code>\per</code>	<code>\per &lt;prefix&gt; &lt;unit&gt; &lt;power&gt;</code>
-------	-------------------	---

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 (“/”).

<hr/>	<code>\cancel</code>	<code>\cancel &lt;prefix&gt; &lt;unit&gt; &lt;power&gt;</code>
-------	----------------------	--

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.

<hr/>	<code>\highlight</code>	<code>\highlight {&lt;color&gt;} &lt;prefix&gt; &lt;unit&gt; &lt;power&gt;</code>
-------	-------------------------	---

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.

<hr/>	<code>\of</code>	<code>&lt;prefix&gt; &lt;unit&gt; &lt;power&gt; \of {&lt;qualifier&gt;}</code>
-------	------------------	--

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.

<hr/>	<code>\raiseto</code>	<code>\raiseto {&lt;power&gt;} &lt;prefix&gt; &lt;unit&gt;</code>
<hr/>	<code>\tothe</code>	<code>&lt;prefix&gt; &lt;unit&gt; \tothe {&lt;power&gt;}</code>

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.

## 5.1 Key-value options

The options defined by this submodule are available within the l3keys `siunitx` tree.

<hr/>	<code>bracket-unit-denominator</code>	<code>bracket-unit-denominator = true false</code>
-------	---------------------------------------	--

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`, `symbol` or `symbol-or-fraction`). The standard setting is `true`.

<hr/> <hr/> extract-mass-in-kilograms	extract-mass-in-kilograms = true false
	Determines whether prefix extraction treats kilograms as a base unit; when set <b>false</b> , grams are used. The standard setting is <b>true</b> .
<hr/> <hr/> forbid-literal-units	forbid-literal-units = true false
	Switch which determines if literal units are allowed when parsing is active; does not apply when <b>parse-units</b> is <b>false</b> .
<hr/> <hr/> fraction-command	fraction-command = $\langle command \rangle$
	Command used to create fractional output when <b>per-mode</b> is set to <b>fraction</b> . The standard setting is <code>\frac</code> .
<hr/> <hr/> inter-unit-product	inter-unit-product = $\langle separator \rangle$
	Inserted between unit combinations in parsed mode, and used to replace <code>.</code> and <code>~</code> in literal mode. The standard setting is <code>\,</code> .
<hr/> <hr/> parse-units	parse-units = true false
	Determines whether parsing of unit symbols is attempted or literal mode is used directly. The standard setting is <b>true</b> .
<hr/> <hr/> per-mode	per-mode = fraction power power-positive-first repeated-symbol symbol symbol-or-fraction
	Selects how the negative powers ( <code>\per</code> ) are formatted: a choice from the options <b>fraction</b> , <b>power</b> , <b>power-positive-first</b> , <b>repeated-symbol</b> , <b>symbol</b> and <b>symbol-or-fraction</b> . The option <b>fraction</b> generates fractional output when appropriate using the command specified by the <b>fraction-command</b> option. The setting <b>power</b> uses reciprocal powers leaving the units in the order of input, while <b>power-positive-first</b> uses the same display format but sorts units such that the positive powers come before negative ones. The <b>symbol</b> setting uses a symbol (specified by <b>per-symbol</b> ) between positive and negative powers, while <b>repeated-symbol</b> uses the same symbol but places it before <i>every</i> unit with a negative power (this is mathematically “wrong” but often seen in real work). Finally, <b>symbol-or-fraction</b> acts like <b>symbol</b> for inline output and like <b>fraction</b> when the output is used in a display math environment. The standard setting is <b>power</b> .
<hr/> <hr/> per-symbol	per-symbol = $\langle symbol \rangle$
	Specifies the symbol to be used to denote negative powers when the option <b>per-mode</b> is set to <b>repeated-symbol</b> , <b>symbol</b> or <b>symbol-or-fraction</b> . The standard setting is <code>/</code> .
<hr/> <hr/> qualifier-mode	qualifier-mode = bracket combine phrase subscript
	Selects how qualifiers are formatted: a choice from the options <b>bracket</b> , <b>combine</b> , <b>phrase</b> and <b>subscript</b> . The option <b>bracket</b> wraps the qualifier in parenthesis, <b>combine</b> joins the qualifier with the unit directly, <b>phrase</b> joins the material using <b>qualifier-phrase</b> as a link, and <b>subscript</b> formats the qualifier as a subscript. The standard setting is <b>subscript</b> .
<hr/> <hr/> qualifier-phrase	qualifier-phrase = $\langle phrase \rangle$
	Defines the $\langle phrase \rangle$ used when <b>qualifier-mode</b> is set to <b>phrase</b> .



---

<code>sticky-per</code>	<code>sticky-per = true false</code>
-------------------------	--------------------------------------

---

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 T<sub>E</sub>X `\over` primitive. The standard setting is `false`.

---

<code>unit-font-command</code>	<code>unit-font-command = &lt;command&gt;</code>
--------------------------------	--

---

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`.

## 6 siunitx-unit implementation

Start the DocStrip guards.

```
1 \*package
```

Identify the internal prefix (L<sup>A</sup>T<sub>E</sub>X3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2 @@=siunitx_unit
```

### 6.1 Initial set up

The mechanisms defined here need a few variables to exist and to be correctly set: these don't belong to one subsection and so are created in a small general block.

Variants not provided by expl3.

```
3 \cs_generate_variant:Nn \tl_replace_all:Nnn { NnV }
```

```
\l__siunitx_unit_tmp_fp
\l__siunitx_unit_tmp_int
\l__siunitx_unit_tmp_tl
```

Scratch space.

```
4 \fp_new:N \l__siunitx_unit_tmp_fp
5 \int_new:N \l__siunitx_unit_tmp_int
6 \tl_new:N \l__siunitx_unit_tmp_tl
```

(End definition for `\l__siunitx_unit_tmp_fp`, `\l__siunitx_unit_tmp_int`, and `\l__siunitx_unit_tmp_tl`.)

```
\c__siunitx_unit_math_subscript_tl
```

Useful tokens with awkward category codes.

```
7 \tl_const:Nx \c__siunitx_unit_math_subscript_tl
8 { \char_generate:nm { '\_ } { 8 } }
```

(End definition for `\c__siunitx_unit_math_subscript_tl`.)

```
\l__siunitx_unit_parsing_bool
```

A boolean is used to indicate when the symbolic unit functions should produce symbolic or literal output. This is used when the symbolic names are used along with literal input, and ensures that there is a sensible fall-back for these cases.

```
9 \bool_new:N \l__siunitx_unit_parsing_bool
```

(End definition for `\l__siunitx_unit_parsing_bool`.)

```
\l__siunitx_unit_test_bool
```

A switch used to indicate that the code is testing the input to find if there is any typeset output from individual unit macros. This is needed to allow the “base” macros to be found, and also to pick up the difference between symbolic and literal unit input.

```
10 \bool_new:N \l__siunitx_unit_test_bool
```

(End definition for \l\_\_siunitx\_unit\_test\_bool.)

\\_siunitx\_unit\_if\_symbolic:nTF

The test for symbolic units is needed in two places. First, there is the case of “pre-parsing” input to check if it can be parsed. Second, when parsing there is a need to check if the current unit is built up from others (symbolic) or is defined in terms of some literals. To do this, the approach used is to set all of the symbolic unit commands expandable and to do nothing, with the few special cases handled manually.

```

11 \prg_new_protected_conditional:Npnn \_siunitx_unit_if_symbolic:n #1 { TF }
12 {
13   \group_begin:
14     \bool_set_true:N \l__siunitx_unit_test_bool
15     \protected@edef \l__siunitx_unit_tmp_tl {#1}
16     \exp_args:NNV \group_end:
17     \tl_if_blank:nTF \l__siunitx_unit_tmp_tl
18       { \prg_return_true: }
19       { \prg_return_false: }
20 }

```

(End definition for \\_siunitx\_unit\_if\_symbolic:nTF.)

## 6.2 Defining symbolic unit

Unit macros and related support are created here. These exist only within the scope of the unit processor code, thus not polluting document-level namespace and allowing overlap with other areas in the case of useful short names (for example \pm). Setting up the mechanisms to allow this requires a few additional steps on top of simply saving the data given by the user in creating the unit.

\l\_siunitx\_unit\_symbolic\_seq

A list of all of the symbolic units, *etc.*, set up. This is needed to allow the symbolic names to be defined within the scope of the unit parser but not elsewhere using simple mappings.

```

21 \seq_new:N \l_siunitx_unit_symbolic_seq

```

(End definition for \l\_siunitx\_unit\_symbolic\_seq. This variable is documented on page ??.)

\l\_siunitx\_unit\_seq

A second list featuring only the units themselves.

```

22 \seq_new:N \l_siunitx_unit_seq

```

(End definition for \l\_siunitx\_unit\_seq. This variable is documented on page ??.)

\\_siunitx\_unit\_set\_symbolic:Nnn

\\_siunitx\_unit\_set\_symbolic:Npnn

\\_siunitx\_unit\_set\_symbolic:Nnnn

The majority of the work for saving each symbolic definition is the same irrespective of the item being defined (unit, prefix, power, qualifier). This is therefore all carried out in a single internal function which does the common tasks. The three arguments here are the symbolic macro name, the literal output and the code to insert when doing full unit parsing. To allow for the “special cases” (where arguments are required) the entire mechanism is set up in a two-part fashion allowing for flexibility at the slight cost of additional functions.

Importantly, notice that the unit macros are declared as expandable. This is required so that literals can be correctly converted into a token list of material which does not depend on local redefinitions for the unit macros. That is required so that the unit formatting system can be grouped.

```

23 \cs_new_protected:Npn \_siunitx_unit_set_symbolic:Nnn #1
24 { \_siunitx_unit_set_symbolic:Nnnn #1 { } }

```

```

25 \cs_new_protected:Npn \__siunitx_unit_set_symbolic:Npnn #1#2#
26 { \__siunitx_unit_set_symbolic:Nnnn #1 {#2} }
27 \cs_new_protected:Npn \__siunitx_unit_set_symbolic:Nnnn #1#2#3#4
28 {
29   \seq_put_right:Nn \l_siunitx_unit_symbolic_seq {#1}
30   \cs_set:cpn { \__siunitx_unit_ \token_to_str:N #1 :w } #2
31   {
32     \bool_if:NF \l__siunitx_unit_test_bool
33     {
34       \bool_if:NTF \l__siunitx_unit_parsing_bool
35       {#4}
36       {#3}
37     }
38   }
39 }

```

(End definition for `\__siunitx_unit_set_symbolic:Nnn`, `\__siunitx_unit_set_symbolic:Npnn`, and `\__siunitx_unit_set_symbolic:Nnnn`.)

`\siunitx_declare_power:Nnn` Powers can come either before or after the unit. As they always come (logically) in matching, we handle this by declaring two commands, and setting each up separately.

```

40 \cs_new_protected:Npn \siunitx_declare_power:Nnn #1#2#3
41 {
42   \__siunitx_unit_set_symbolic:Nnn #1
43   { \__siunitx_unit_literal_power:nn {#3} }
44   { \__siunitx_unit_parse_power:nnN {#1} {#3} \c_true_bool }
45   \__siunitx_unit_set_symbolic:Nnn #2
46   { ~ {#3} }
47   { \__siunitx_unit_parse_power:nnN {#2} {#3} \c_false_bool }
48 }

```

(End definition for `\siunitx_declare_power:Nnn`. This function is documented on page ??.)

`\siunitx_declare_prefix:Nn` For prefixes there are a couple of options. In all cases, the basic requirement is to set up to parse the prefix using the appropriate internal function. For prefixes which are powers of 10, there is also the need to be able to do conversion to/from the numerical equivalent. That is handled using two properly lists which can be used to supply the conversion data later.

`\l_siunitx_unit_prefixes_forward_prop`  
`\l_siunitx_unit_prefixes_reverse_prop`

```

49 \cs_new_protected:Npn \siunitx_declare_prefix:Nn #1#2
50 {
51   \__siunitx_unit_set_symbolic:Nnn #1
52   {#2}
53   { \__siunitx_unit_parse_prefix:Nn #1 {#2} }
54 }
55 \cs_new_protected:Npn \siunitx_declare_prefix:Nnn #1#2#3
56 {
57   \siunitx_declare_prefix:Nn #1 {#3}
58   \prop_put:Nnn \l__siunitx_unit_prefixes_forward_prop {#3} {#2}
59   \prop_put:Nnn \l__siunitx_unit_prefixes_reverse_prop {#2} {#3}
60 }
61 \cs_generate_variant:Nn \siunitx_declare_prefix:Nnn { Nnx }
62 \prop_new:N \l__siunitx_unit_prefixes_forward_prop
63 \prop_new:N \l__siunitx_unit_prefixes_reverse_prop

```

(End definition for `\siunitx_declare_prefix:Nn` and others. These functions are documented on page ??.)

`\siunitx_declare_qualifier:Nn` Qualifiers are relatively easy to handle: nothing to do other than save the input appropriately.

```

64 \cs_new_protected:Npn \siunitx_declare_qualifier:Nn #1#2
65 {
66   \__siunitx_unit_set_symbolic:Nnn #1
67   { ~ ( #2 ) }
68   { \__siunitx_unit_parse_qualifier:nn {#1} {#2} }
69 }

```

(End definition for `\siunitx_declare_qualifier:Nn`. This function is documented on page ??.)

`\siunitx_declare_unit:Nn` For the unit parsing, allowing for variations in definition order requires that a test is made for the output of each unit at point of use.

```

\siunitx_declare_unit:Nx
\siunitx_declare_unit:Nnn
\siunitx_declare_unit:Nxn
70 \cs_new_protected:Npn \siunitx_declare_unit:Nn #1#2
71 { \siunitx_declare_unit:Nnn #1 {#2} { } }
72 \cs_generate_variant:Nn \siunitx_declare_unit:Nn { Nx }
73 \cs_new_protected:Npn \siunitx_declare_unit:Nnn #1#2#3
74 {
75   \seq_put_right:Nn \l_siunitx_unit_seq {#1}
76   \__siunitx_unit_set_symbolic:Nnn #1
77   {#2}
78   {
79     \__siunitx_unit_if_symbolic:nTF {#2}
80     {#2}
81     { \__siunitx_unit_parse_unit:Nn #1 {#2} }
82   }
83   \tl_clear_new:c { l__siunitx_unit_options_ \token_to_str:N #1 _tl }
84   \tl_if_empty:nF {#3}
85   { \tl_set:cn { l__siunitx_unit_options_ \token_to_str:N #1 _tl } {#3} }
86 }
87 \cs_generate_variant:Nn \siunitx_declare_unit:Nnn { Nx }

```

(End definition for `\siunitx_declare_unit:Nn` and `\siunitx_declare_unit:Nnn`. These functions are documented on page ??.)

### 6.3 Applying unit options

`\l_siunitx_unit_options_bool`

```

88 \bool_new:N \l_siunitx_unit_options_bool

```

(End definition for `\l_siunitx_unit_options_bool`.)

`\siunitx_unit_options_apply:n` Options apply only if they have not already been set at this group level.

```

89 \cs_new_protected:Npn \siunitx_unit_options_apply:n #1
90 {
91   \bool_if:NF \l_siunitx_unit_options_bool
92   {
93     \tl_if_single_token:nT {#1}
94     {
95       \tl_if_exist:cT { l__siunitx_unit_options_ \token_to_str:N #1 _tl }
96       {

```

```

97         \keys_set:nv { siunitx }
98         { l__siunitx_unit_options_ \token_to_str:N #1 _tl }
99     }
100 }
101 }
102 \bool_set_true:N \l__siunitx_unit_options_bool
103 }

```

(End definition for `\siunitx_unit_options_apply:n`. This function is documented on page ??.)

## 6.4 Non-standard symbolic units

A few of the symbolic units require non-standard definitions: these are created here. They all use parts of the more general code but have particular requirements which can only be addressed by hand. Some of these could in principle be used in place of the dedicated definitions above, but at point of use that would then require additional expansions for each unit parsed: as the macro names would still be needed, this does not offer any real benefits.

**\per** The `\per` symbolic unit is a bit special: it has a mechanism entirely different from everything else, so has to be set up by hand. In literal mode it is represented by a very simple symbol!

```

104 \__siunitx_unit_set_symbolic:Nnn \per
105 { / }
106 { \__siunitx_unit_parse_per: }

```

(End definition for `\per`. This function is documented on page ??.)

**\cancel** The two special cases, `\cancel` and `\highlight`, are easy to deal with when parsing.  
**\highlight** When not parsing, a precaution is taken to ensure that the user level equivalents always get a braced argument.

```

107 \__siunitx_unit_set_symbolic:Npnn \cancel
108 { }
109 { \__siunitx_unit_parse_special:n { \cancel } }
110 \__siunitx_unit_set_symbolic:Npnn \highlight #1
111 { \__siunitx_unit_literal_special:nN { \textcolor {#1} } }
112 { \__siunitx_unit_parse_special:n { \textcolor {#1} } }

```

(End definition for `\cancel` and `\highlight`. These functions are documented on page ??.)

**\of** The generic qualifier is simply the same as the dedicated ones except for needing to grab an argument.

```

113 \__siunitx_unit_set_symbolic:Npnn \of #1
114 { \ ( #1 ) }
115 { \__siunitx_unit_parse_qualifier:nn { \of {#1} } {#1} }

```

(End definition for `\of`. This function is documented on page ??.)

**\raiseto** Generic versions of the pre-defined power macros. These require an argument and so  
**\tothe** cannot be handled using the general approach. Other than that, the code here is very similar to that in `\siunitx_unit_power_set:NnN`.

```

116 \__siunitx_unit_set_symbolic:Npnn \raiseto #1
117 { \__siunitx_unit_literal_power:nn {#1} }
118 { \__siunitx_unit_parse_power:nnN { \raiseto {#1} } {#1} \c_true_bool }

```

```

119 \__siunitx_unit_set_symbolic:Npnn \tothe #1
120 { ^ {#1} }
121 { \__siunitx_unit_parse_power:nnN { \tothe {#1} } {#1} \c_false_bool }

```

(End definition for `\raiseto` and `\tothe`. These functions are documented on page ??.)

## 6.5 Main formatting routine

Unit input can take two forms, “literal” units (material to be typeset directly) or “symbolic” units (macro-based). Before any parsing or typesetting is carried out, a small amount of pre-parsing has to be carried out to decide which of these cases applies.

```

\l_siunitx_unit_font_tl Options which apply to the main formatting routine, and so are not tied to either symbolic
\l__siunitx_unit_product_tl or literal input.
\l__siunitx_unit_mass_kilogram_bool
122 \keys_define:nn { siunitx }
123 {
124     extract-mass-in-kilograms .bool_set:N =
125     \l__siunitx_unit_mass_kilogram_bool ,
126     inter-unit-product .tl_set:N =
127     \l__siunitx_unit_product_tl ,
128     unit-font-command .tl_set:N =
129     \l_siunitx_unit_font_tl
130 }

```

(End definition for `\l_siunitx_unit_font_tl`, `\l__siunitx_unit_product_tl`, and `\l__siunitx_unit_mass_kilogram_bool`. This variable is documented on page ??.)

```

\l__siunitx_unit_formatted_tl A token list for the final formatted result: may or may not be generated by the parser,
                             depending on the nature of the input.
131 \tl_new:N \l__siunitx_unit_formatted_tl
(End definition for \l__siunitx_unit_formatted_tl.)

```

```

\siunitx_unit_format:nN Formatting parsed units can take place either with the prefixes printed or separated out
\siunitx_unit_format_extract_prefixes:nnN into a power of ten. This variation is handled using two separate functions: as this
\siunitx_unit_format_combine_exponent:nnN submodule does not really deal with numbers, formatting the numeral part here would
\siunitx_unit_format_multiply:nnN be tricky and it is better therefore to have a mechanism to return a simple numerical
\unit_format_multiply_extract_prefixes:nnN power. At the same time, most uses will no want this more complex return format and
\unit_format_multiply_combine_exponent:nnN so a version of the code which does not do this is also provided.
\__siunitx_unit_format:nNN The main unit formatting routine groups all of the parsing/formatting, so that the
\__siunitx_unit_format_aux: only value altered will be the return token list. As definitions for the various unit macros
                             are not globally created, the first step is to map over the list of names and active the
                             unit definitions: these do different things depending on the switches set. There is then
                             a decision to be made: is the unit input one that can be parsed (“symbolic”), or is it
                             one containing one or more literals. In the latter case, there is a still the need to convert
                             the input into an expanded token list as some parts of the input could still be using unit
                             macros.

```

Notice that for `\siunitx_unit_format:nN` a second return value from the auxiliary has to be allowed for, but is simply discarded.

```

132 \cs_new_protected:Npn \siunitx_unit_format:nN #1#2
133 {
134     \bool_set_false:N \l__siunitx_unit_prefix_exp_bool
135     \fp_zero:N \l__siunitx_unit_combine_exp_fp

```

```

136     \fp_set:Nn \l__siunitx_unit_multiple_fp { \c_one_fp }
137     \__siunitx_unit_format:nNN {#1} #2 \l__siunitx_unit_tmp_fp
138   }
139 \cs_new_protected:Npn \siunitx_unit_format_extract_prefixes:nNN #1#2#3
140 {
141   \bool_set_true:N \l__siunitx_unit_prefix_exp_bool
142   \fp_zero:N \l__siunitx_unit_combine_exp_fp
143   \fp_set:Nn \l__siunitx_unit_multiple_fp { \c_one_fp }
144   \__siunitx_unit_format:nNN {#1} #2 #3
145 }
146 \cs_new_protected:Npn \siunitx_unit_format_combine_exponent:nnN #1#2#3
147 {
148   \bool_set_false:N \l__siunitx_unit_prefix_exp_bool
149   \fp_set:Nn \l__siunitx_unit_combine_exp_fp {#2}
150   \fp_set:Nn \l__siunitx_unit_multiple_fp { \c_one_fp }
151   \__siunitx_unit_format:nNN {#1} #3 \l__siunitx_unit_tmp_fp
152 }
153 \cs_new_protected:Npn \siunitx_unit_format_multiply:nnN #1#2#3
154 {
155   \bool_set_false:N \l__siunitx_unit_prefix_exp_bool
156   \fp_zero:N \l__siunitx_unit_combine_exp_fp
157   \fp_set:Nn \l__siunitx_unit_multiple_fp {#2}
158   \__siunitx_unit_format:nNN {#1} #3 \l__siunitx_unit_tmp_fp
159 }
160 \cs_new_protected:Npn \siunitx_unit_format_multiply_extract_prefixes:nnNN
161 #1#2#3#4
162 {
163   \bool_set_true:N \l__siunitx_unit_prefix_exp_bool
164   \fp_zero:N \l__siunitx_unit_combine_exp_fp
165   \fp_set:Nn \l__siunitx_unit_multiple_fp {#2}
166   \__siunitx_unit_format:nNN {#1} #3 #4
167 }
168 \cs_new_protected:Npn \siunitx_unit_format_multiply_combine_exponent:nnnN
169 #1#2#3#4
170 {
171   \bool_set_false:N \l__siunitx_unit_prefix_exp_bool
172   \fp_set:Nn \l__siunitx_unit_combine_exp_fp {#3}
173   \fp_set:Nn \l__siunitx_unit_multiple_fp {#2}
174   \__siunitx_unit_format:nNN {#1} #4 \l__siunitx_unit_tmp_fp
175 }
176 \cs_new_protected:Npn \__siunitx_unit_format:nNN #1#2#3
177 {
178   \group_begin:
179   \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
180   { \cs_set_eq:Nc ##1 { __siunitx_unit_ \token_to_str:N ##1 :w } }
181   \tl_clear:N \l__siunitx_unit_formatted_tl
182   \fp_zero:N \l__siunitx_unit_prefix_fp
183   \bool_if:NTF \l__siunitx_unit_parse_bool
184   {
185     \__siunitx_unit_if_symbolic:nTF {#1}
186     {
187       \__siunitx_unit_parse:n {#1}
188       \prop_if_empty:NF \l__siunitx_unit_parsed_prop
189       { \__siunitx_unit_format_parsed: }

```

```

190     }
191     {
192         \bool_if:NTF \l__siunitx_unit_forbid_literal_bool
193         { \msg_error:nnn { siunitx } { unit / literal } {#1} }
194         { \__siunitx_unit_format_literal:n {#1} }
195     }
196 }
197 { \__siunitx_unit_format_literal:n {#1} }
198 \cs_set_protected:Npx \__siunitx_unit_format_aux:
199 {
200     \tl_set:Nn \exp_not:N #2
201     { \exp_not:V \l__siunitx_unit_formatted_tl }
202     \fp_set:Nn \exp_not:N #3
203     { \fp_use:N \l__siunitx_unit_prefix_fp }
204 }
205 \exp_after:wN \group_end:
206 \__siunitx_unit_format_aux:
207 }
208 \cs_new_protected:Npn \__siunitx_unit_format_aux: { }

```

(End definition for `\siunitx_unit_format:nN` and others. These functions are documented on page ??.)

## 6.6 Formatting literal units

While in literal mode no parsing occurs, there is a need to provide a few auxiliary functions to handle one or two special cases.

`\_siunitx_unit_literal_power:nn` For printing literal units which are given before the unit they apply to, there is a slight rearrangement. This is ex[EXP]pandable to cover the case of creation of a PDF string.

```

209 \cs_new:Npn \__siunitx_unit_literal_power:nn #1#2 { #2 ^ {#1} }

```

(End definition for `\_siunitx_unit_literal_power:nn`.)

`\_siunitx_unit_literal_special:nN` When dealing with the special cases, there is an argument to absorb. This should be braced to be passed up to the user level, which is dealt with here.

```

210 \cs_new:Npn \__siunitx_unit_literal_special:nN #1#2 { #1 {#2} }

```

(End definition for `\_siunitx_unit_literal_special:nN`.)

`\_siunitx_unit_format_literal:n` To format literal units, there are two tasks to do. The input is x-type expanded to force any symbolic units to be converted into their literal representation: this requires setting the appropriate switch. In the resulting token list, all `.` and `~` tokens are then replaced by the current unit product token list. To enable this to happen correctly with a normal (active) `~`, a small amount of “protection” is needed first. To cover active sub- and superscript tokens, appropriate definitions are provided at this stage. Those have to be expandable macros rather than implicit character tokens.

As with other code dealing with user input, `\protected@edef` is used here rather than `\tl_set:Nx` as L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> robust commands may be present.

```

211 \group_begin:
212   \char_set_catcode_active:n { '~ }
213   \cs_new_protected:Npx \__siunitx_unit_format_literal:n #1
214   {
215     \group_begin:

```



```

216 \exp_not:n { \bool_set_false:N \l__siunitx_unit_parsing_bool }
217 \tl_set:Nn \exp_not:N \l__siunitx_unit_tmp_tl {#1}
218 \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
219 { \token_to_str:N ^ } { ^ }
220 \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
221 { \token_to_str:N _ } { \c__siunitx_unit_math_subscript_tl }
222 \char_set_active_eq:NN ^
223 \exp_not:N \__siunitx_unit_format_literal_superscript:
224 \char_set_active_eq:NN _
225 \exp_not:N \__siunitx_unit_format_literal_subscript:
226 \char_set_active_eq:NN \exp_not:N ~
227 \exp_not:N \__siunitx_unit_format_literal_tilde:
228 \exp_not:n
229 {
230   \protected@edef \l__siunitx_unit_tmp_tl
231   { \l__siunitx_unit_tmp_tl }
232   \tl_clear:N \l__siunitx_unit_formatted_tl
233   \tl_if_empty:NF \l__siunitx_unit_tmp_tl
234   {
235     \exp_after:wN \__siunitx_unit_format_literal_auxi:w
236     \l__siunitx_unit_tmp_tl .
237     \q_recursion_tail . \q_recursion_stop
238   }
239   \exp_args:NNNV \group_end:
240   \tl_set:Nn \l__siunitx_unit_formatted_tl
241   \l__siunitx_unit_formatted_tl
242 }
243 }
244 \group_end:
245 \cs_new:Npx \__siunitx_unit_format_literal_subscript: { \c__siunitx_unit_math_subscript_tl }
246 \cs_new:Npn \__siunitx_unit_format_literal_superscript: { ^ }
247 \cs_new:Npn \__siunitx_unit_format_literal_tilde: { . }

To introduce the font changing commands while still allowing for line breaks in literal
units, a loop is needed to replace one . at a time. To also allow for division, a second
loop is used within that to handle /: as a result, the separator between parts has to be
tracked.

248 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxi:w #1 .
249 {
250   \quark_if_recursion_tail_stop:n {#1}
251   \__siunitx_unit_format_literal_auxii:n {#1}
252   \tl_set_eq:NN \l__siunitx_unit_separator_tl \l__siunitx_unit_product_tl
253   \__siunitx_unit_format_literal_auxi:w
254 }
255 \cs_set_protected:Npn \__siunitx_unit_format_literal_auxii:n #1
256 {
257   \__siunitx_unit_format_literal_auxiii:w
258   #1 / \q_recursion_tail / \q_recursion_stop
259 }
260 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxiii:w #1 /
261 {
262   \quark_if_recursion_tail_stop:n {#1}
263   \__siunitx_unit_format_literal_auxiv:n {#1}
264   \tl_set:Nn \l__siunitx_unit_separator_tl { / }

```

```

265     \_siunitx_unit_format_literal_auxiii:w
266   }
267 \cs_new_protected:Npn \_siunitx_unit_format_literal_auxiv:n #1
268 {
269     \_siunitx_unit_format_literal_auxv:nw { }
270     #1 \q_recursion_tail \q_recursion_stop
271 }

```

To deal properly with literal formatting, we have to worry about super- and subscript markers. That can be complicated as they could come anywhere in the input: we handle that by iterating through the input and picking them out. This avoids any issue with losing braces for mid-input scripts. We also have to deal with fractions, hence needing a series of nested loops and a change of separator.

```

272 \cs_new_protected:Npn \_siunitx_unit_format_literal_auxv:nw
273   #1#2 \q_recursion_stop
274 {
275     \tl_if_head_is_N_type:nTF {#2}
276     { \_siunitx_unit_format_literal_auxvi:nN }
277     {
278         \tl_if_head_is_group:nTF {#2}
279         { \_siunitx_unit_format_literal_auxix:nn }
280         { \_siunitx_unit_format_literal_auxx:nw }
281     }
282     {#1} #2 \q_recursion_stop
283 }
284 \cs_new_protected:Npx \_siunitx_unit_format_literal_auxvi:nN #1#2
285 {
286     \exp_not:N \quark_if_recursion_tail_stop_do:Nn #2
287     { \exp_not:N \_siunitx_unit_format_literal_add:n {#1} }
288     \exp_not:N \token_if_eq_meaning:NNTF #2 ^
289     { \exp_not:N \_siunitx_unit_format_literal_super:nn {#1} }
290     {
291         \exp_not:N \token_if_eq_meaning:NNTF
292         #2 \c__siunitx_unit_math_subscript_tl
293         { \exp_not:N \_siunitx_unit_format_literal_sub:nn {#1} }
294         { \exp_not:N \_siunitx_unit_format_literal_auxvii:nN {#1} #2 }
295     }
296 }

```

We need to make sure `\protect` sticks with the next token.

```

297 \cs_new_protected:Npn \_siunitx_unit_format_literal_auxvii:nN #1#2
298 {
299     \str_if_eq:nnTF {#2} { \protect }
300     { \_siunitx_unit_format_literal_auxviii:nN {#1} }
301     { \_siunitx_unit_format_literal_auxv:nw {#1#2} }
302 }
303 \cs_new_protected:Npn \_siunitx_unit_format_literal_auxviii:nN #1#2
304 { \_siunitx_unit_format_literal_auxv:nw { #1 \protect #2 } }
305 \cs_new_protected:Npn \_siunitx_unit_format_literal_super:nn #1#2
306 {
307     \quark_if_recursion_tail_stop:n {#2}
308     \_siunitx_unit_format_literal_add:n {#1}
309     \tl_put_right:Nn \l__siunitx_unit_formatted_tl { ^ {#2} }
310     \_siunitx_unit_format_literal_auxvi:nN { }
311 }

```

```

312 \cs_new_protected:Npx \__siunitx_unit_format_literal_sub:nn #1#2
313 {
314   \exp_not:N \quark_if_recursion_tail_stop:n {#2}
315   \exp_not:N \__siunitx_unit_format_literal_add:n {#1}
316   \tl_put_right:Nx \exp_not:N \l__siunitx_unit_formatted_tl
317   {
318     \c__siunitx_unit_math_subscript_tl
319     {
320       \exp_not:N \exp_not:V
321       \exp_not:N \l_siunitx_unit_font_tl
322       { \exp_not:N \exp_not:n {#2} }
323     }
324   }
325   \exp_not:N \__siunitx_unit_format_literal_auxvi:nN { }
326 }
327 \cs_new_protected:Npn \__siunitx_unit_format_literal_add:n #1
328 {
329   \tl_put_right:Nx \l__siunitx_unit_formatted_tl
330   {
331     \tl_if_empty:NF \l__siunitx_unit_formatted_tl
332     { \exp_not:V \l__siunitx_unit_separator_tl }
333     \tl_if_empty:nF {#1}
334     { \exp_not:V \l_siunitx_unit_font_tl { \exp_not:n {#1} } }
335   }
336   \tl_clear:N \l__siunitx_unit_separator_tl
337 }
338 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxix:nn #1#2
339 { \__siunitx_unit_format_literal_auxvi:nN { #1 {#2} } }
340 \use:x
341 {
342   \cs_new_protected:Npn \exp_not:N \__siunitx_unit_format_literal_auxx:nw
343   ##1 \c_space_tl
344 }
345 { \__siunitx_unit_format_literal_auxv:nw {#1} }
346 \tl_new:N \l__siunitx_unit_separator_tl

```

(End definition for \\_\_siunitx\_unit\_format\_literal:n and others.)

## 6.7 (PDF) String creation

\siunitx\_unit\_pdfstring\_context: A simple function that sets up to make units equal to their text representation.

```

347 \cs_new_protected:Npn \siunitx_unit_pdfstring_context:
348 {
349   \bool_set_false:N \l__siunitx_unit_parsing_bool
350   \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
351   { \cs_set_eq:Nc ##1 { __siunitx_unit_ \token_to_str:N ##1 :w } }
352 }

```

(End definition for \siunitx\_unit\_pdfstring\_context:.. This function is documented on page ??.)

## 6.8 Parsing symbolic units

Parsing units takes place by storing information about each unit in a `prop`. As well as the unit itself, there are various other optional data points, for example a prefix or a power.

Some of these can come before the unit, others only after. The parser therefore tracks the number of units read and uses the current position to allocate data to individual units.

The result of parsing is a property list (`\l__siunitx_unit_parsed_prop`) which contains one or more entries for each unit:

- **prefix-*n*** The symbol for the prefix which applies to this unit, *e.g.* for `\kilo` with (almost certainly) would be `k`.
- **unit-*n*** The symbol for the unit itself, *e.g.* for `\metre` with (almost certainly) would be `m`.
- **power-*n*** The power which a unit is raised to. During initial parsing this will (almost certainly) be positive, but is combined with **per-*n*** to give a “fully qualified” power before any formatting takes place
- **per-*n*** Indicates that **per** applies to the current unit: stored during initial parsing then combined with **power-*n*** (and removed from the list) before further work.
- **qualifier-*n*** Any qualifier which applies to the current unit.
- **special-*n*** Any “special effect” to apply to the current unit.
- **command-*l*** The command corresponding to **unit-*n***: needed to track base units; used for `\gram` only.

`\l__siunitx_unit_sticky_per_bool` There is one option when *parsing* the input (as opposed to *formatting* for output): how to deal with `\per`.

```

353 \keys_define:nn { siunitx }
354   {
355     sticky-per .bool_set:N = \l__siunitx_unit_sticky_per_bool
356   }

```

(End definition for `\l__siunitx_unit_sticky_per_bool`.)

`\l__siunitx_unit_parsed_prop`  
`\l__siunitx_unit_per_bool`  
`\l__siunitx_unit_position_int` Parsing units requires a small number of variables are available: a **prop** for the parsed units themselves, a **bool** to indicate if `\per` is active and an **int** to track how many units have be parsed.

```

357 \prop_new:N \l__siunitx_unit_parsed_prop
358 \bool_new:N \l__siunitx_unit_per_bool
359 \int_new:N \l__siunitx_unit_position_int

```

(End definition for `\l__siunitx_unit_parsed_prop`, `\l__siunitx_unit_per_bool`, and `\l__siunitx_unit_position_int`.)

`\__siunitx_unit_parse:n` The main parsing function is quite simple. After initialising the variables, each symbolic unit is set up. The input is then simply inserted into the input stream: the symbolic units themselves then do the real work of placing data into the parsing system. There is then a bit of tidying up to ensure that later stages can rely on the nature of the data here.

```

360 \cs_new_protected:Npn \__siunitx_unit_parse:n #1
361   {
362     \prop_clear:N \l__siunitx_unit_parsed_prop
363     \bool_set_true:N \l__siunitx_unit_parsing_bool
364     \bool_set_false:N \l__siunitx_unit_per_bool
365     \bool_set_false:N \l__siunitx_unit_test_bool

```

```

366     \int_zero:N \l__siunitx_unit_position_int
367     \siunitx_unit_options_apply:n {#1}
368     #1
369     \int_step_inline:nn \l__siunitx_unit_position_int
370     { \__siunitx_unit_parse_finalise:n {##1} }
371     \__siunitx_unit_parse_finalise:
372 }

```

(End definition for \\_\_siunitx\_unit\_parse:n.)

\\_\_siunitx\_unit\_parse\_add:nnnn In all cases, storing a data item requires setting a temporary `tl` which will be used as the key, then using this to store the value. The `tl` is set using `x-type` expansion as this will expand the unit index and any additional calculations made for this.

```

373 \cs_new_protected:Npn \__siunitx_unit_parse_add:nnnn #1#2#3#4
374 {
375     \tl_set:Nx \l__siunitx_unit_tmp_tl { #1 - #2 }
376     \prop_if_in:NVTF \l__siunitx_unit_parsed_prop
377     \l__siunitx_unit_tmp_tl
378     {
379         \msg_error:nnxx { siunitx } { unit / duplicate-part }
380         { \exp_not:n {#1} } { \token_to_str:N #3 }
381     }
382     {
383         \prop_put:NVn \l__siunitx_unit_parsed_prop
384         \l__siunitx_unit_tmp_tl {#4}
385     }
386 }

```

(End definition for \\_\_siunitx\_unit\_parse\_add:nnnn.)

\\_\_siunitx\_unit\_parse\_prefix:Nn  
\\_\_siunitx\_unit\_parse\_power:nnN  
\\_\_siunitx\_unit\_parse\_qualifier:nn  
\\_\_siunitx\_unit\_parse\_special:nn

Storage of the various optional items follows broadly the same pattern in each case. The data to be stored is passed along with an appropriate key name to the underlying storage system. The details for each type of item should be relatively clear. For example, prefixes have to come before their “parent” unit and so there is some adjustment to do to add them to the correct unit.

```

387 \cs_new_protected:Npn \__siunitx_unit_parse_prefix:Nn #1#2
388 {
389     \int_set:Nn \l__siunitx_unit_tmp_int { \l__siunitx_unit_position_int + 1 }
390     \__siunitx_unit_parse_add:nnnn { prefix }
391     { \int_use:N \l__siunitx_unit_tmp_int } {#1} {#2}
392 }
393 \cs_new_protected:Npn \__siunitx_unit_parse_power:nnN #1#2#3
394 {
395     \tl_set:Nx \l__siunitx_unit_tmp_tl
396     { unit- \int_use:N \l__siunitx_unit_position_int }
397     \bool_lazy_or:nnTF
398     {#3}
399     {
400         \prop_if_in_p:N
401         \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
402     }
403     {
404         \__siunitx_unit_parse_add:nnnn { power }
405         {

```

```

406         \int_eval:n
407         { \l__siunitx_unit_position_int \bool_if:NT #3 { + 1 } }
408     }
409     {#1} {#2}
410 }
411 {
412     \msg_error:nxxx { siunitx }
413     { unit / part-before-unit } { power } { \token_to_str:N #1 }
414 }
415 }
416 \cs_new_protected:Npn \__siunitx_unit_parse_qualifier:nn #1#2
417 {
418     \tl_set:Nx \l__siunitx_unit_tmp_tl
419     { unit- \int_use:N \l__siunitx_unit_position_int }
420     \prop_if_in:NVTF \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
421     {
422         \__siunitx_unit_parse_add:nnnn { qualifier }
423         { \int_use:N \l__siunitx_unit_position_int } {#1} {#2}
424     }
425     {
426         \msg_error:nnnn { siunitx }
427         { unit / part-before-unit } { qualifier } { \token_to_str:N #1 }
428     }
429 }

```

Special (exceptional) items should always come before the relevant units.

```

430 \cs_new_protected:Npn \__siunitx_unit_parse_special:n #1
431 {
432     \__siunitx_unit_parse_add:nnnn { special }
433     { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
434     {#1} {#1}
435 }

```

(End definition for \\_\_siunitx\_unit\_parse\_prefix:Nn and others.)

\\_\_siunitx\_unit\_parse\_unit:Nn Parsing units is slightly more involved than the other cases: this is the one place where the tracking value is incremented. If the switch \l\_\_siunitx\_unit\_per\_bool is set true then the current unit is also reciprocal: this can only happen if \l\_\_siunitx\_unit\_sticky\_per\_bool is also true, so only one test is required.

```

436 \cs_new_protected:Npn \__siunitx_unit_parse_unit:Nn #1#2
437 {
438     \int_incr:N \l__siunitx_unit_position_int
439     \tl_if_eq:nT {#1} { \gram }
440     {
441         \__siunitx_unit_parse_add:nnnn { command }
442         { \int_use:N \l__siunitx_unit_position_int }
443         {#1} {#1}
444     }
445     \__siunitx_unit_parse_add:nnnn { unit }
446     { \int_use:N \l__siunitx_unit_position_int }
447     {#1} {#2}
448     \bool_if:NT \l__siunitx_unit_per_bool
449     {
450         \__siunitx_unit_parse_add:nnnn { per }

```

```

451         { \int_use:N \l__siunitx_unit_position_int }
452         { \per } { true }
453     }
454 }

```

(End definition for \\_\_siunitx\_unit\_parse\_unit:Nn.)

\\_\_siunitx\_unit\_parse\_per: Storing the \per command requires adding a data item separate from the power which applies: this makes later formatting much more straight-forward. This data could in principle be combined with the power, but depending on the output format required that may make life more complex. Thus this information is stored separately for later retrieval. If \per is set to be “sticky” then after parsing the first occurrence, any further uses are in error.

```

455 \cs_new_protected:Npn \__siunitx_unit_parse_per:
456 {
457     \bool_if:NTF \l__siunitx_unit_sticky_per_bool
458     {
459         \bool_set_true:N \l__siunitx_unit_per_bool
460         \cs_set_protected:Npn \per
461         { \msg_error:nn { siunitx } { unit / duplicate-sticky-per } }
462     }
463     {
464         \__siunitx_unit_parse_add:nnnn
465         { per } { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
466         { \per } { true }
467     }
468 }

```

(End definition for \\_\_siunitx\_unit\_parse\_per:.)

\\_\_siunitx\_unit\_parse\_finalise:n If \per applies to the current unit, the power needs to be multiplied by  $-1$ . That is done using an fp operation so that non-integer powers are supported. The flag for \per is also removed as this means we don’t have to check that the original power was positive. To be on the safe side, there is a check for a trivial power at this stage.

```

469 \cs_new_protected:Npn \__siunitx_unit_parse_finalise:n #1
470 {
471     \tl_set:Nx \l__siunitx_unit_tmp_tl { per- #1 }
472     \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
473     {
474         \prop_remove:NV \l__siunitx_unit_parsed_prop
475         \l__siunitx_unit_tmp_tl
476         \tl_set:Nx \l__siunitx_unit_tmp_tl { power- #1 }
477         \prop_get:NVNTF
478         \l__siunitx_unit_parsed_prop
479         \l__siunitx_unit_tmp_tl
480         \l__siunitx_unit_part_tl
481         {
482             \tl_set:Nx \l__siunitx_unit_part_tl
483             { \fp_eval:n { \l__siunitx_unit_part_tl * -1 } }
484             \fp_compare:nNnTF \l__siunitx_unit_part_tl = 1
485             {
486                 \prop_remove:NV \l__siunitx_unit_parsed_prop
487                 \l__siunitx_unit_tmp_tl
488             }
489         }
490     }

```

```

489         {
490             \prop_put:NVV \l__siunitx_unit_parsed_prop
491             \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
492         }
493     }
494     {
495         \prop_put:NVN \l__siunitx_unit_parsed_prop
496         \l__siunitx_unit_tmp_tl { -1 }
497     }
498 }
499 }

```

(End definition for `\__siunitx_unit_parse_finalise:n`.)

`\__siunitx_unit_parse_finalise:` The final task is to check that there is not a “dangling” power or prefix: these are added to the “next” unit so are easy to test for.

```

500 \cs_new_protected:Npn \__siunitx_unit_parse_finalise:
501 {
502     \clist_map_inline:nn { per , power , prefix }
503     {
504         \tl_set:Nx \l__siunitx_unit_tmp_tl
505         { ##1 - \int_eval:n { \l__siunitx_unit_position_int + 1 } }
506         \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
507         { \msg_error:nnn { siunitx } { unit / dangling-part } { ##1 } }
508     }
509 }

```

(End definition for `\__siunitx_unit_parse_finalise:.`)

## 6.9 Formatting parsed units

```

\l_siunitx_unit_fraction_tl Set up the options which apply to formatting.
\l__siunitx_unit_denominator_bracket_bool
\l__siunitx_unit_forbid_literal_bool
\l__siunitx_unit_parse_bool
\l__siunitx_unit_per_symbol_tl
\l__siunitx_unit_qualifier_mode_tl
\l__siunitx_unit_qualifier_phrase_tl
510 \keys_define:nn { siunitx }
511 {
512     bracket-unit-denominator .bool_set:N =
513     \l__siunitx_unit_denominator_bracket_bool ,
514     forbid-literal-units .bool_set:N =
515     \l__siunitx_unit_forbid_literal_bool ,
516     fraction-command .tl_set:N =
517     \l_siunitx_unit_fraction_tl ,
518     parse-units .bool_set:N =
519     \l__siunitx_unit_parse_bool ,
520     per-mode .choice: ,
521     per-mode / fraction .code:n =
522     {
523         \bool_set_false:N \l__siunitx_unit_autofrac_bool
524         \bool_set_false:N \l__siunitx_unit_per_symbol_bool
525         \bool_set_true:N \l__siunitx_unit_powers_positive_bool
526         \bool_set_true:N \l__siunitx_unit_two_part_bool
527     } ,
528     per-mode / power .code:n =
529     {
530         \bool_set_false:N \l__siunitx_unit_autofrac_bool
531         \bool_set_false:N \l__siunitx_unit_per_symbol_bool

```



```

532     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
533     \bool_set_false:N \l__siunitx_unit_two_part_bool
534   } ,
535   per-mode / power-positive-first .code:n =
536   {
537     \bool_set_false:N \l__siunitx_unit_autofrac_bool
538     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
539     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
540     \bool_set_true:N \l__siunitx_unit_two_part_bool
541   } ,
542   per-mode / repeated-symbol .code:n =
543   {
544     \bool_set_false:N \l__siunitx_unit_autofrac_bool
545     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
546     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
547     \bool_set_false:N \l__siunitx_unit_two_part_bool
548   } ,
549   per-mode / symbol .code:n =
550   {
551     \bool_set_false:N \l__siunitx_unit_autofrac_bool
552     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
553     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
554     \bool_set_true:N \l__siunitx_unit_two_part_bool
555   } ,
556   per-mode / symbol-or-fraction .code:n =
557   {
558     \bool_set_true:N \l__siunitx_unit_autofrac_bool
559     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
560     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
561     \bool_set_true:N \l__siunitx_unit_two_part_bool
562   } ,
563   per-symbol .tl_set:N =
564     \l__siunitx_unit_per_symbol_tl ,
565   qualifier-mode .choices:nn =
566     { bracket , combine , phrase , subscript }
567     { \tl_set_eq:NN \l__siunitx_unit_qualifier_mode_tl \l_keys_choice_tl } ,
568   qualifier-phrase .tl_set:N =
569     \l__siunitx_unit_qualifier_phrase_tl
570 }

```

*(End definition for \l\_siunitx\_unit\_fraction\_tl and others. This variable is documented on page ??.)*

`\l_siunitx_unit_bracket_bool` A flag to indicate that the unit currently under construction will require brackets if a power is added.

```

571 \bool_new:N \l__siunitx_unit_bracket_bool

```

*(End definition for \l\_siunitx\_unit\_bracket\_bool.)*

`\l_siunitx_unit_bracket_open_tl` Abstracted out but currently purely internal.

```

\l_siunitx_unit_bracket_close_tl
572 \tl_new:N \l__siunitx_unit_bracket_open_tl
573 \tl_new:N \l__siunitx_unit_bracket_close_tl
574 \tl_set:Nn \l__siunitx_unit_bracket_open_tl { ( }
575 \tl_set:Nn \l__siunitx_unit_bracket_close_tl { ) }

```

(End definition for \l\_\_siunitx\_unit\_bracket\_open\_tl and \l\_\_siunitx\_unit\_bracket\_close\_tl.)

\l\_\_siunitx\_unit\_font\_bool A flag to control when font wrapping is applied to the output.

576 \bool\_new:N \l\_\_siunitx\_unit\_font\_bool

(End definition for \l\_\_siunitx\_unit\_font\_bool.)

\l\_\_siunitx\_unit\_autofrac\_bool Dealing with the various ways that reciprocal (\per) can be handled requires a few different switches.

\l\_\_siunitx\_unit\_powers\_positive\_bool 577 \bool\_new:N \l\_\_siunitx\_unit\_autofrac\_bool

\l\_\_siunitx\_unit\_per\_symbol\_bool 578 \bool\_new:N \l\_\_siunitx\_unit\_per\_symbol\_bool

579 \bool\_new:N \l\_\_siunitx\_unit\_powers\_positive\_bool

580 \bool\_new:N \l\_\_siunitx\_unit\_two\_part\_bool

(End definition for \l\_\_siunitx\_unit\_autofrac\_bool and others.)

\l\_\_siunitx\_unit\_numerator\_bool Indicates that the current unit should go into the numerator when splitting into two parts (fractions or other “sorted” styles).

581 \bool\_new:N \l\_\_siunitx\_unit\_numerator\_bool

(End definition for \l\_\_siunitx\_unit\_numerator\_bool.)

\l\_\_siunitx\_unit\_qualifier\_mode\_tl For storing the text of options which are best handled by picking function names.

582 \tl\_new:N \l\_\_siunitx\_unit\_qualifier\_mode\_tl

(End definition for \l\_\_siunitx\_unit\_qualifier\_mode\_tl.)

\l\_\_siunitx\_unit\_combine\_exp\_fp For combining an exponent with the first unit.

583 \fp\_new:N \l\_\_siunitx\_unit\_combine\_exp\_fp

(End definition for \l\_\_siunitx\_unit\_combine\_exp\_fp.)

\l\_\_siunitx\_unit\_prefix\_exp\_bool Used to determine if prefixes are converted into powers. Note that while this may be set as an option “higher up”, at this point it is handled as an internal switch (see the two formatting interfaces for reasons).

584 \bool\_new:N \l\_\_siunitx\_unit\_prefix\_exp\_bool

(End definition for \l\_\_siunitx\_unit\_prefix\_exp\_bool.)

\l\_\_siunitx\_unit\_prefix\_fp When converting prefixes to powers, the calculations are done as an fp.

585 \fp\_new:N \l\_\_siunitx\_unit\_prefix\_fp

(End definition for \l\_\_siunitx\_unit\_prefix\_fp.)

\l\_\_siunitx\_unit\_multiple\_fp For multiplying units.

586 \fp\_new:N \l\_\_siunitx\_unit\_multiple\_fp

(End definition for \l\_\_siunitx\_unit\_multiple\_fp.)

\l\_\_siunitx\_unit\_current\_tl Building up the (partial) formatted unit requires some token list storage. Each part of the unit combination that is recovered also has to be placed in a token list: this is a dedicated one to leave the scratch variables available.

\l\_\_siunitx\_unit\_part\_tl

587 \tl\_new:N \l\_\_siunitx\_unit\_current\_tl

588 \tl\_new:N \l\_\_siunitx\_unit\_part\_tl

(End definition for \l\_\_siunitx\_unit\_current\_tl and \l\_\_siunitx\_unit\_part\_tl.)

`\l_siunitx_unit_denominator_tl` For fraction-like units, space is needed for the denominator as well as the numerator (which is handled using `\l__siunitx_unit_formatted_tl`).

589 `\tl_new:N \l__siunitx_unit_denominator_tl`

(End definition for `\l_siunitx_unit_denominator_tl`.)

`\l__siunitx_unit_total_int` The formatting routine needs to know both the total number of units and the current unit. Thus an `int` is required in addition to `\l__siunitx_unit_position_int`.

590 `\int_new:N \l__siunitx_unit_total_int`

(End definition for `\l__siunitx_unit_total_int`.)

`\_siunitx_unit_format_parsed:`  
`\_siunitx_unit_format_parsed_aux:n` The main formatting routine is essentially a loop over each position, reading the various parts of the unit to build up complete unit combination.

591 `\cs_new_protected:Npn \_siunitx_unit_format_parsed:`  
592 `{`  
593 `\int_set_eq:NN \l__siunitx_unit_total_int \l__siunitx_unit_position_int`  
594 `\tl_clear:N \l__siunitx_unit_denominator_tl`  
595 `\tl_clear:N \l__siunitx_unit_formatted_tl`  
596 `\fp_zero:N \l__siunitx_unit_prefix_fp`  
597 `\int_zero:N \l__siunitx_unit_position_int`  
598 `\fp_compare:nNnF \l__siunitx_unit_combine_exp_fp = \c_zero_fp`  
599 `{ \_siunitx_unit_format_combine_exp: }`  
600 `\fp_compare:nNnF \l__siunitx_unit_multiple_fp = \c_one_fp`  
601 `{ \_siunitx_unit_format_multiply: }`  
602 `\bool_lazy_and:nnT`  
603 `{ \l__siunitx_unit_prefix_exp_bool }`  
604 `{ \l__siunitx_unit_mass_kilogram_bool }`  
605 `{ \_siunitx_unit_format_mass_to_kilogram: }`  
606 `\int_do_while:nNnn`  
607 `\l__siunitx_unit_position_int < \l__siunitx_unit_total_int`  
608 `{`  
609 `\bool_set_false:N \l__siunitx_unit_bracket_bool`  
610 `\tl_clear:N \l__siunitx_unit_current_tl`  
611 `\bool_set_false:N \l__siunitx_unit_font_bool`  
612 `\bool_set_true:N \l__siunitx_unit_numerator_bool`  
613 `\int_incr:N \l__siunitx_unit_position_int`  
614 `\clist_map_inline:nn { prefix , unit , qualifier , power , special }`  
615 `{ \_siunitx_unit_format_parsed_aux:n {##1} }`  
616 `\_siunitx_unit_format_output:`  
617 `}`  
618 `\_siunitx_unit_format_finalise:`  
619 `}`  
620 `\cs_new_protected:Npn \_siunitx_unit_format_parsed_aux:n #1`  
621 `{`  
622 `\tl_set:Nx \l__siunitx_unit_tmp_tl`  
623 `{ #1 - \int_use:N \l__siunitx_unit_position_int }`  
624 `\prop_get:NVNT \l__siunitx_unit_parsed_prop`  
625 `\l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl`  
626 `{ \use:c { __siunitx_unit_format_ #1 : } }`  
627 `}`

(End definition for `\_siunitx_unit_format_parsed:` and `\_siunitx_unit_format_parsed_aux:n`.)

\\_siunitx\_unit\_format\_combine\_exp: To combine an exponent into the first prefix, we first adjust for any power, then deal with any existing prefix, before looking up the final result.

```

628 \cs_new_protected:Npn \__siunitx_unit_format_combine_exp:
629 {
630   \prop_get:NnNF \l__siunitx_unit_parsed_prop { power-1 } \l__siunitx_unit_tmp_tl
631   { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
632   \fp_set:Nn \l__siunitx_unit_tmp_fp
633   { \l__siunitx_unit_combine_exp_fp / \l__siunitx_unit_tmp_tl }
634   \prop_get:NnNTF \l__siunitx_unit_parsed_prop { prefix-1 } \l__siunitx_unit_tmp_tl
635   {
636     \prop_get:NVNF \l__siunitx_unit_prefixes_forward_prop
637     \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
638     {
639       \prop_get:NnN \l__siunitx_unit_parsed_prop { prefix-1 } \l__siunitx_unit_tmp_tl
640       \msg_error:nnx { siunitx } { unit / non-numeric-exponent }
641       { \l__siunitx_unit_tmp_tl }
642       \tl_set:Nn \l__siunitx_unit_tmp_tl { 0 }
643     }
644   }
645   { \tl_set:Nn \l__siunitx_unit_tmp_tl { 0 } }
646   \tl_set:Nx \l__siunitx_unit_tmp_tl
647   { \fp_eval:n { \l__siunitx_unit_tmp_fp + \l__siunitx_unit_tmp_tl } }
648   \fp_compare:nNnTF \l__siunitx_unit_tmp_tl = \c_zero_fp
649   { \prop_remove:Nn \l__siunitx_unit_parsed_prop { prefix-1 } }
650   {
651     \prop_get:NVNTF \l__siunitx_unit_prefixes_reverse_prop
652     \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
653     { \prop_put:NnV \l__siunitx_unit_parsed_prop { prefix-1 } \l__siunitx_unit_tmp_tl }
654     {
655       \msg_error:nnx { siunitx } { unit / non-convertible-exponent }
656       { \l__siunitx_unit_tmp_tl }
657     }
658   }
659 }

```

(End definition for \\_siunitx\_unit\_format\_combine\_exp:.)

\\_siunitx\_unit\_format\_multiply: A simple mapping.

```

660 \cs_new_protected:Npn \__siunitx_unit_format_multiply:
661 {
662   \int_step_inline:nn { \prop_count:N \l__siunitx_unit_parsed_prop }
663   {
664     \prop_get:NnNF \l__siunitx_unit_parsed_prop { power- ##1 } \l__siunitx_unit_tmp_tl
665     { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
666     \fp_set:Nn \l__siunitx_unit_tmp_fp
667     { \l__siunitx_unit_tmp_tl * \l__siunitx_unit_multiple_fp }
668     \fp_compare:nNnTF \l__siunitx_unit_tmp_fp = \c_one_fp
669     { \prop_remove:N \l__siunitx_unit_parsed_prop { power- ##1 } }
670     {
671       \prop_put:Nnx \l__siunitx_unit_parsed_prop { power- ##1 }
672       { \fp_use:N \l__siunitx_unit_tmp_fp }
673     }
674   }
675 }

```

(End definition for \\_siunitx\_unit\_format\_multiply:.)

\\_siunitx\_unit\_format\_mass\_to\_kilogram: To deal correctly with prefix extraction in combination with kilograms, we need to coerce the prefix for grams. Currently, only this one special case is recorded in the property list, so we do not actually need to check the value. If there is then no prefix we do a bit of gymnastics to create one and then shift the starting point for the prefix extraction.

```

676 \cs_new_protected:Npn \_siunitx_unit_format_mass_to_kilogram:
677 {
678   \int_step_inline:nn \l__siunitx_unit_total_int
679   {
680     \prop_if_in:NnT \l__siunitx_unit_parsed_prop { command- ##1 }
681     {
682       \prop_if_in:NnF \l__siunitx_unit_parsed_prop { prefix- ##1 }
683       {
684         \group_begin:
685         \bool_set_false:N \l__siunitx_unit_parsing_bool
686         \tl_set:Nx \l__siunitx_unit_tmp_tl { \kilo }
687         \exp_args:NNNV \group_end:
688         \tl_set:Nn \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
689         \prop_put:NnV \l__siunitx_unit_parsed_prop { prefix- ##1 }
690         \l__siunitx_unit_tmp_tl
691         \prop_get:NnNF \l__siunitx_unit_parsed_prop { power- ##1 }
692         \l__siunitx_unit_tmp_tl
693         { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
694         \fp_set:Nn \l__siunitx_unit_prefix_fp
695         { \l__siunitx_unit_prefix_fp - 3 * \l__siunitx_unit_tmp_tl }
696       }
697     }
698   }
699 }

```

(End definition for \\_siunitx\_unit\_format\_mass\_to\_kilogram:.)

\\_siunitx\_unit\_format\_bracket:N A quick utility function which wraps up a token list variable in brackets if they are required.

```

700 \cs_new:Npn \_siunitx_unit_format_bracket:N #1
701 {
702   \bool_if:NTF \l__siunitx_unit_bracket_bool
703   {
704     \exp_not:V \l__siunitx_unit_bracket_open_tl
705     \exp_not:V #1
706     \exp_not:V \l__siunitx_unit_bracket_close_tl
707   }
708   { \exp_not:V #1 }
709 }

```

(End definition for \\_siunitx\_unit\_format\_bracket:N.)

\\_siunitx\_unit\_format\_power: Formatting powers requires a test for negative numbers and depending on output format requests some adjustment to the stored value. This could be done using an fp function, but that would be slow compared to a dedicated if lower-level approach based on delimited arguments.

```

710 \cs_new_protected:Npn \_siunitx_unit_format_power:
711 {

```

```

712 \__siunitx_unit_format_font:
713 \exp_after:wN \__siunitx_unit_format_power_aux:wTF
714 \l__siunitx_unit_part_tl - \q_stop
715 { \__siunitx_unit_format_power_negative: }
716 { \__siunitx_unit_format_power_positive: }
717 }
718 \cs_new:Npn \__siunitx_unit_format_power_aux:wTF #1 - #2 \q_stop
719 { \tl_if_empty:nTF {#1} }

```

In the case of positive powers, there is little to do: add the power as a subscript (must be required as the parser ensures it's  $\neq 1$ ).

```

720 \cs_new_protected:Npn \__siunitx_unit_format_power_positive:
721 { \__siunitx_unit_format_power_superscript: }

```

Dealing with negative powers starts by flipping the switch used to track where in the final output the current part should get added to. For the case where the output is fraction-like, strip off the  $\sim$  then ensure that the result is not the trivial power 1. Assuming all is well, addition to the current unit combination goes ahead.

```

722 \cs_new_protected:Npn \__siunitx_unit_format_power_negative:
723 {
724   \bool_set_false:N \l__siunitx_unit_numerator_bool
725   \bool_if:NTF \l__siunitx_unit_powers_positive_bool
726   {
727     \tl_set:Nx \l__siunitx_unit_part_tl
728     {
729       \exp_after:wN \__siunitx_unit_format_power_negative_aux:w
730       \l__siunitx_unit_part_tl \q_stop
731     }
732     \str_if_eq:VnF \l__siunitx_unit_part_tl { 1 }
733     { \__siunitx_unit_format_power_superscript: }
734   }
735   { \__siunitx_unit_format_power_superscript: }
736 }
737 \cs_new:Npn \__siunitx_unit_format_power_negative_aux:w - #1 \q_stop
738 { \exp_not:n {#1} }

```

Adding the power as a superscript has the slight complication that there is the possibility of needing some brackets. The superscript itself uses `\sp` as that avoids any category code issues and also allows redirection at a higher level more readily.

```

739 \cs_new_protected:Npn \__siunitx_unit_format_power_superscript:
740 {
741   \exp_after:wN \__siunitx_unit_format_power_superscript:w
742   \l__siunitx_unit_part_tl . . \q_stop
743 }
744 \cs_new_protected:Npn \__siunitx_unit_format_power_superscript:w #1 . #2 . #3 \q_stop
745 {
746   \tl_if_blank:nTF {#2}
747   {
748     \tl_set:Nx \l__siunitx_unit_current_tl
749     {
750       \__siunitx_unit_format_bracket:N \l__siunitx_unit_current_tl
751       ^ { \exp_not:n {#1} }
752     }
753   }
754   {

```

```

755     \tl_set:Nx \l__siunitx_unit_tmp_tl
756     {
757       { }
758       \tl_if_head_eq_charcode:nNTF {#1} -
759       { { - } { \exp_not:o { \use_none:n #1 } } }
760       { { } { \exp_not:n {#1} } }
761       {#2}
762       { }
763       { }
764       { 0 }
765     }
766     \tl_set:Nx \l__siunitx_unit_current_tl
767     {
768       \__siunitx_unit_format_bracket:N \l__siunitx_unit_current_tl
769       ^ { \siunitx_number_output:N \l__siunitx_unit_tmp_tl }
770     }
771   }
772   \bool_set_false:N \l__siunitx_unit_bracket_bool
773 }

```

(End definition for `\__siunitx_unit_format_power:` and others.)

`\__siunitx_unit_format_prefix:` Formatting for prefixes depends on whether they are to be expressed as symbols or collected up to be returned as a power of 10. The latter case requires a bit of processing, which includes checking that the conversion is possible and allowing for any power that applies to the current unit.

```

774 \cs_new_protected:Npn \__siunitx_unit_format_prefix:
775 {
776   \bool_if:NTF \l__siunitx_unit_prefix_exp_bool
777   { \__siunitx_unit_format_prefix_exp: }
778   { \__siunitx_unit_format_prefix_symbol: }
779 }
780 \cs_new_protected:Npn \__siunitx_unit_format_prefix_exp:
781 {
782   \prop_get:NVNTF \l__siunitx_unit_prefixes_forward_prop
783   \l__siunitx_unit_part_tl \l__siunitx_unit_part_tl
784   {
785     \bool_if:NT \l__siunitx_unit_mass_kilogram_bool
786     {
787       \tl_set:Nx \l__siunitx_unit_tmp_tl
788       { command- \int_use:N \l__siunitx_unit_position_int }
789       \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
790       { \__siunitx_unit_format_prefix_gram: }
791     }
792     \tl_set:Nx \l__siunitx_unit_tmp_tl
793     { power- \int_use:N \l__siunitx_unit_position_int }
794     \prop_get:NVNF \l__siunitx_unit_parsed_prop
795     \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
796     { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
797     \fp_add:Nn \l__siunitx_unit_prefix_fp
798     { \l__siunitx_unit_tmp_tl * \l__siunitx_unit_part_tl }
799   }
800   { \__siunitx_unit_format_prefix_symbol: }
801 }

```

When the units in use are grams, we may need to deal with conversion to kilograms.

```

802 \cs_new_protected:Npn \__siunitx_unit_format_prefix_gram:
803 {
804   \tl_set:Nx \l__siunitx_unit_part_tl
805     { \int_eval:n { \l__siunitx_unit_part_tl - 3 } }
806   \group_begin:
807     \bool_set_false:N \l__siunitx_unit_parsing_bool
808     \tl_set:Nx \l__siunitx_unit_current_tl { \kilo }
809     \exp_args:NNNV \group_end:
810     \tl_set:Nn \l__siunitx_unit_current_tl \l__siunitx_unit_current_tl
811   }
812 \cs_new_protected:Npn \__siunitx_unit_format_prefix_symbol:
813 { \tl_set_eq:NN \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl }

```

(End definition for `\__siunitx_unit_format_prefix:` and others.)

`\__siunitx_unit_format_qualifier:` There are various ways that a qualifier can be added to the output. The idea here is to modify the “base” text appropriately and then add to the current unit. Notice that when the qualifier is just treated as “text”, the auxiliary is actually a no-op.

```

814 \cs_new_protected:Npn \__siunitx_unit_format_qualifier:
815 {
816   \use:c
817   {
818     __siunitx_unit_format_qualifier_
819     \l__siunitx_unit_qualifier_mode_tl :
820   }
821   \tl_put_right:NV \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
822 }
823 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_bracket:
824 {
825   \__siunitx_unit_format_font:
826   \tl_set:Nx \l__siunitx_unit_part_tl
827     {
828       \exp_not:V \l__siunitx_unit_bracket_open_tl
829       \exp_not:V \l__siunitx_unit_font_tl
830       { \exp_not:V \l__siunitx_unit_part_tl }
831       \exp_not:V \l__siunitx_unit_bracket_close_tl
832     }
833 }
834 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_combine: { }
835 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_phrase:
836 {
837   \__siunitx_unit_format_font:
838   \tl_set:Nx \l__siunitx_unit_part_tl
839     {
840       \exp_not:V \l__siunitx_unit_qualifier_phrase_tl
841       \exp_not:V \l__siunitx_unit_font_tl
842       { \exp_not:V \l__siunitx_unit_part_tl }
843     }
844 }
845 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_subscript:
846 {
847   \__siunitx_unit_format_font:
848   \tl_set:Nx \l__siunitx_unit_part_tl

```



```

849     {
850       \c__siunitx_unit_math_subscript_tl
851       {
852         \exp_not:V \l_siunitx_unit_font_tl
853         { \exp_not:V \l__siunitx_unit_part_tl }
854       }
855     }
856   }

```

(End definition for \\_siunitx\_unit\_format\_qualifier: and others.)

\\_siunitx\_unit\_format\_special: Any special odds and ends are handled by simply making the current combination into an argument for the recovered code. Font control needs to be *inside* the special formatting here.

```

857 \cs_new_protected:Npn \_siunitx_unit_format_special:
858   {
859     \tl_set:Nx \l__siunitx_unit_current_tl
860     {
861       \exp_not:V \l__siunitx_unit_part_tl
862       {
863         \bool_if:NTF \l__siunitx_unit_font_bool
864         { \use:n }
865         { \exp_not:V \l_siunitx_unit_font_tl }
866         { \exp_not:V \l__siunitx_unit_current_tl }
867       }
868     }
869     \bool_set_true:N \l__siunitx_unit_font_bool
870   }

```

(End definition for \\_siunitx\_unit\_format\_special:.)

\\_siunitx\_unit\_format\_unit: A very simple task: add the unit to the output currently being constructed.

```

871 \cs_new_protected:Npn \_siunitx_unit_format_unit:
872   {
873     \tl_put_right:NV
874     \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
875   }

```

(End definition for \\_siunitx\_unit\_format\_unit:.)

\\_siunitx\_unit\_format\_output: The first step here is to make a choice based on whether the current part should be stored as part of the numerator or denominator of a fraction. In all cases, if the switch \l\_\_siunitx\_unit\_numerator\_bool is true then life is simple: add the current part to the numerator with a standard separator

```

\__siunitx_unit_format_output_aux:n
\__siunitx_unit_format_output_aux:nV
\__siunitx_unit_format_output_aux:nV
876 \cs_new_protected:Npn \_siunitx_unit_format_output:
877   {
878     \__siunitx_unit_format_font:
879     \bool_set_false:N \l__siunitx_unit_bracket_bool
880     \use:c
881     {
882       \__siunitx_unit_format_output_
883       \bool_if:NTF \l__siunitx_unit_numerator_bool
884       { aux: }
885       { denominator: }

```

```

886     }
887   }
888   \cs_new_protected:Npn \__siunitx_unit_format_output_aux:
889   {
890     \__siunitx_unit_format_output_aux:nV { formatted }
891     \l__siunitx_unit_product_tl
892   }

```

There are a few things to worry about at this stage if the current part is in the denominator. Powers have already been dealt with and some formatting outcomes only need a branch at the final point of building the entire unit. That means that there are three possible outcomes here: if collecting two separate parts, add to the denominator with a product separator, or if only building one token list there may be a need to use a symbol separator. When the `repeated-symbol` option is in use there may be a need to add a leading 1 to the output in the case where the first unit is in the denominator: that can be picked up by looking for empty output in combination with the flag for using a symbol in the output but not a two-part strategy.

```

893   \cs_new_protected:Npn \__siunitx_unit_format_output_denominator:
894   {
895     \bool_if:NTF \l__siunitx_unit_two_part_bool
896     {
897       \bool_lazy_and:nnT
898       { \l__siunitx_unit_denominator_bracket_bool }
899       { ! \tl_if_empty_p:N \l__siunitx_unit_denominator_tl }
900       { \bool_set_true:N \l__siunitx_unit_bracket_bool }
901       \__siunitx_unit_format_output_aux:nV { denominator }
902       \l__siunitx_unit_product_tl
903     }
904     {
905       \bool_lazy_and:nnT
906       { \l__siunitx_unit_per_symbol_bool }
907       { \tl_if_empty_p:N \l__siunitx_unit_formatted_tl }
908       { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
909       \__siunitx_unit_format_output_aux:nv { formatted }
910       {
911         \l__siunitx_unit_
912         \bool_if:NTF \l__siunitx_unit_per_symbol_bool
913         { per_symbol }
914         { product }
915         _tl
916       }
917     }
918   }
919   \cs_new_protected:Npn \__siunitx_unit_format_output_aux:nn #1#2
920   {
921     \tl_set:cx { l__siunitx_unit_ #1 _tl }
922     {
923       \exp_not:v { l__siunitx_unit_ #1 _tl }
924       \tl_if_empty:cF { l__siunitx_unit_ #1 _tl }
925       { \exp_not:n {#2} }
926       \exp_not:V \l__siunitx_unit_current_tl
927     }
928   }
929   \cs_generate_variant:Nn \__siunitx_unit_format_output_aux:nn { nV , nv }

```

(End definition for `\_siunitx_unit_format_output:` and others.)

`\_siunitx_unit_format_font:` A short auxiliary which checks if the font has been applied to the main part of the output: if not, add it and set the flag.

```

930 \cs_new_protected:Npn \_siunitx_unit_format_font:
931 {
932   \bool_if:NF \l__siunitx_unit_font_bool
933   {
934     \tl_set:Nx \l__siunitx_unit_current_tl
935     {
936       \exp_not:V \l_siunitx_unit_font_tl
937       { \exp_not:V \l__siunitx_unit_current_tl }
938     }
939     \bool_set_true:N \l__siunitx_unit_font_bool
940   }
941 }

```

(End definition for `\_siunitx_unit_format_font:`)

`\_siunitx_unit_format_finalise:` Finalising the unit format is really about picking up the cases involving fractions: these require assembly of the parts with the need to add additional material in some cases

```

\siunitx_unit_format_finalise_autofrac:
\siunitx_unit_format_finalise_fractional:
\siunitx_unit_format_finalise_power:
942 \cs_new_protected:Npn \_siunitx_unit_format_finalise:
943 {
944   \tl_if_empty:NF \l__siunitx_unit_denominator_tl
945   {
946     \bool_if:NTF \l__siunitx_unit_powers_positive_bool
947     { \_siunitx_unit_format_finalise_fractional: }
948     { \_siunitx_unit_format_finalise_power: }
949   }
950 }

```

For fraction-like output, there are three possible choices and two actual styles. In all cases, if the numerator is empty then it is set here to 1. To deal with the “auto-format” case, the two styles (fraction and symbol) are handled in auxiliaries: this allows both to be used at the same time! Beyond that, the key here is to use a single `\tl_set:Nx` to keep down the number of assignments.

```

951 \cs_new_protected:Npn \_siunitx_unit_format_finalise_fractional:
952 {
953   \tl_if_empty:NT \l__siunitx_unit_formatted_tl
954   { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
955   \bool_if:NTF \l__siunitx_unit_autofrac_bool
956   { \_siunitx_unit_format_finalise_autofrac: }
957   {
958     \bool_if:NTF \l__siunitx_unit_per_symbol_bool
959     { \_siunitx_unit_format_finalise_symbol: }
960     { \_siunitx_unit_format_finalise_fraction: }
961   }
962 }

```

For the “auto-selected” fraction method, the two other auxiliary functions are used to do both forms of formatting. So that everything required is available, this needs one group so that the second auxiliary receives the correct input. After that it is just a case of applying `\mathchoice` to the formatted output.

```

963 \cs_new_protected:Npn \_siunitx_unit_format_finalise_autofrac:

```

```

964 {
965   \group_begin:
966   \__siunitx_unit_format_finalise_fraction:
967   \exp_args:NNNV \group_end:
968   \tl_set:Nn \l__siunitx_unit_tmp_tl \l__siunitx_unit_formatted_tl
969   \__siunitx_unit_format_finalise_symbol:
970   \tl_set:Nx \l__siunitx_unit_formatted_tl
971     {
972       \mathchoice
973       { \exp_not:V \l__siunitx_unit_tmp_tl }
974       { \exp_not:V \l__siunitx_unit_formatted_tl }
975       { \exp_not:V \l__siunitx_unit_formatted_tl }
976       { \exp_not:V \l__siunitx_unit_formatted_tl }
977     }
978 }

```

When using a fraction function the two parts are now assembled.

```

979 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fraction:
980 {
981   \tl_set:Nx \l__siunitx_unit_formatted_tl
982     {
983       \exp_not:V \l__siunitx_unit_fraction_tl
984       { \exp_not:V \l__siunitx_unit_formatted_tl }
985       { \exp_not:V \l__siunitx_unit_denominator_tl }
986     }
987 }
988 \cs_new_protected:Npn \__siunitx_unit_format_finalise_symbol:
989 {
990   \tl_set:Nx \l__siunitx_unit_formatted_tl
991     {
992       \exp_not:V \l__siunitx_unit_formatted_tl
993       \exp_not:V \l__siunitx_unit_per_symbol_tl
994       \__siunitx_unit_format_bracket:N \l__siunitx_unit_denominator_tl
995     }
996 }

```

In the case of sorted powers, there is a test to make sure there was at least one positive power, and if so a simple join of the two parts with the appropriate product.

```

997 \cs_new_protected:Npn \__siunitx_unit_format_finalise_power:
998 {
999   \tl_if_empty:NTF \l__siunitx_unit_formatted_tl
1000   {
1001     \tl_set_eq:NN
1002     \l__siunitx_unit_formatted_tl
1003     \l__siunitx_unit_denominator_tl
1004   }
1005   {
1006     \tl_set:Nx \l__siunitx_unit_formatted_tl
1007     {
1008       \exp_not:V \l__siunitx_unit_formatted_tl
1009       \exp_not:V \l__siunitx_unit_product_tl
1010       \exp_not:V \l__siunitx_unit_denominator_tl
1011     }
1012   }
1013 }

```

(End definition for `\_siunitx_unit_format_finalise:` and others.)

## 6.10 Non-Latin character support

`\_siunitx_unit_non_latin:n` A small amount of code to make it convenient to include non-Latin characters in units without having to directly include them in the sources directly.

```

1014 \bool_lazy_or:nnTF
1015 { \sys_if_engine luatex_p: }
1016 { \sys_if_engine xetex_p: }
1017 {
1018   \cs_new:Npn \_siunitx_unit_non_latin:n #1
1019     { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
1020 }
1021 {
1022   \cs_new:Npn \_siunitx_unit_non_latin:n #1
1023     {
1024       \exp_last_unbraced:Nf \_siunitx_unit_non_latin:nnnn
1025       { \char_to_utfviii_bytes:n {#1} }
1026     }
1027   \cs_new:Npn \_siunitx_unit_non_latin:nnnn #1#2#3#4
1028     {
1029       \exp_after:wN \exp_after:wN \exp_after:wN
1030       \exp_not:N \char_generate:nn {#1} { 13 }
1031       \exp_after:wN \exp_after:wN \exp_after:wN
1032       \exp_not:N \char_generate:nn {#2} { 13 }
1033     }
1034 }

```

(End definition for `\_siunitx_unit_non_latin:n` and `\_siunitx_unit_non_latin:nnnn`.)

## 6.11 Pre-defined unit components

Quite a number of units can be predefined: while this is a code-level module, there is little point having a unit parser which does not start off able to parse any units!

`\kilogram` The basic SI units: technically the correct spelling is `\metre` but US users tend to use `\meter`.

```

\meter 1035 \siunitx_declare_unit:Nn \kilogram { \kilo \gram }
\mole   1036 \siunitx_declare_unit:Nn \metre   { m }
\kelvin 1037 \siunitx_declare_unit:Nn \meter   { \metre }
\candela 1038 \siunitx_declare_unit:Nn \mole     { mol }
\second 1039 \siunitx_declare_unit:Nn \second   { s }
\ampere  1040 \siunitx_declare_unit:Nn \ampere     { A }
         1041 \siunitx_declare_unit:Nn \kelvin    { K }
         1042 \siunitx_declare_unit:Nn \candela   { cd }

```

(End definition for `\kilogram` and others. These functions are documented on page ??.)

`\gram` The gram is an odd unit as it is needed for the base unit kilogram.

```

1043 \siunitx_declare_unit:Nn \gram { g }

```

(End definition for `\gram`. This function is documented on page ??.)

\yocto The various SI multiple prefixes are defined here: first the small ones.

```

\zepto 1044 \siunitx_declare_prefix:Nnn \yocto { -24 } { y }
\atto 1045 \siunitx_declare_prefix:Nnn \zepto { -21 } { z }
\femto 1046 \siunitx_declare_prefix:Nnn \atto { -18 } { a }
\pico 1047 \siunitx_declare_prefix:Nnn \femto { -15 } { f }
\nano 1048 \siunitx_declare_prefix:Nnn \pico { -12 } { p }
\micro 1049 \siunitx_declare_prefix:Nnn \nano { -9 } { n }
\milli 1050 \siunitx_declare_prefix:Nnn \micro { -6 } { \_siunitx_unit_non_latin:n { "03BC } }
\centi 1051 \siunitx_declare_prefix:Nnn \milli { -3 } { m }
\deci 1052 \siunitx_declare_prefix:Nnn \centi { -2 } { c }
1053 \siunitx_declare_prefix:Nnn \deci { -1 } { d }

```

(End definition for \yocto and others. These functions are documented on page ??.)

\deca Now the large ones.

```

\deka 1054 \siunitx_declare_prefix:Nnn \deca { 1 } { da }
\hecto 1055 \siunitx_declare_prefix:Nnn \deka { 1 } { da }
\kilo 1056 \siunitx_declare_prefix:Nnn \hecto { 2 } { h }
\mega 1057 \siunitx_declare_prefix:Nnn \kilo { 3 } { k }
\giga 1058 \siunitx_declare_prefix:Nnn \mega { 6 } { M }
\tera 1059 \siunitx_declare_prefix:Nnn \giga { 9 } { G }
\peta 1060 \siunitx_declare_prefix:Nnn \tera { 12 } { T }
\exa 1061 \siunitx_declare_prefix:Nnn \peta { 15 } { P }
\zetta 1062 \siunitx_declare_prefix:Nnn \exa { 18 } { E }
\yotta 1063 \siunitx_declare_prefix:Nnn \zetta { 21 } { Z }
1064 \siunitx_declare_prefix:Nnn \yotta { 24 } { Y }

```

(End definition for \deca and others. These functions are documented on page ??.)

\becquerel Named derived units: first half of alphabet.

```

\degreeCelsius 1065 \siunitx_declare_unit:Nn \becquerel { Bq }
\coulomb 1066 \siunitx_declare_unit:Nx \degreeCelsius { \_siunitx_unit_non_latin:n { "00B0 } C }
\farad 1067 \siunitx_declare_unit:Nn \coulomb { C }
\gray 1068 \siunitx_declare_unit:Nn \farad { F }
\hertz 1069 \siunitx_declare_unit:Nn \gray { Gy }
\henry 1070 \siunitx_declare_unit:Nn \hertz { Hz }
\joule 1071 \siunitx_declare_unit:Nn \henry { H }
\katal 1072 \siunitx_declare_unit:Nn \joule { J }
\katal 1073 \siunitx_declare_unit:Nn \katal { kat }
\lumen 1074 \siunitx_declare_unit:Nn \lumen { lm }
\lux 1075 \siunitx_declare_unit:Nn \lux { lx }

```

(End definition for \becquerel and others. These functions are documented on page ??.)

\newton Named derived units: second half of alphabet.

```

\ohm 1076 \siunitx_declare_unit:Nn \newton { N }
\pascal 1077 \siunitx_declare_unit:Nx \ohm { \_siunitx_unit_non_latin:n { "03A9 } }
\radian 1078 \siunitx_declare_unit:Nn \pascal { Pa }
\siemens 1079 \siunitx_declare_unit:Nn \radian { rad }
\sievert 1080 \siunitx_declare_unit:Nn \siemens { S }
\steradian 1081 \siunitx_declare_unit:Nn \sievert { Sv }
\tesla 1082 \siunitx_declare_unit:Nn \steradian { sr }
\volt 1083 \siunitx_declare_unit:Nn \tesla { T }
\watt 1084 \siunitx_declare_unit:Nn \volt { V }
\watt 1085 \siunitx_declare_unit:Nn \watt { W }
\weber 1086 \siunitx_declare_unit:Nn \weber { Wb }

```

(End definition for `\newton` and others. These functions are documented on page ??.)

`\astronomicalunit` Non-SI, but accepted for general use. Once again there are two spellings, here for litre and with different output in this case.

```

\dalton 1087 \siunitx_declare_unit:Nn \astronomicalunit { au }
\day     1088 \siunitx_declare_unit:Nn \bel           { B }
\decibel 1089 \siunitx_declare_unit:Nn \decibel       { \deci \bel }
\electronvolt 1090 \siunitx_declare_unit:Nn \dalton     { Da }
\hectare 1091 \siunitx_declare_unit:Nn \day           { d }
\hour    1092 \siunitx_declare_unit:Nn \electronvolt  { eV }
\litre   1093 \siunitx_declare_unit:Nn \hectare       { ha }
\liter   1094 \siunitx_declare_unit:Nn \hour          { h }
\minute  1095 \siunitx_declare_unit:Nn \litre         { L }
\neper   1096 \siunitx_declare_unit:Nn \liter        { \litre }
\tonne   1097 \siunitx_declare_unit:Nn \minute       { min }
         1098 \siunitx_declare_unit:Nn \neper        { Np }
         1099 \siunitx_declare_unit:Nn \tonne        { t }

```

(End definition for `\astronomicalunit` and others. These functions are documented on page ??.)

`\arcminute` Arc units: again, non-SI, but accepted for general use.

```

\arcsecond 1100 \siunitx_declare_unit:Nx \arcminute { \_siunitx_unit_non_latin:n { "02B9 } }
\degree    1101 \siunitx_declare_unit:Nx \arcsecond { \_siunitx_unit_non_latin:n { "02BA } }
          1102 \siunitx_declare_unit:Nx \degree { \_siunitx_unit_non_latin:n { "00B0 } }

```

(End definition for `\arcminute`, `\arcsecond`, and `\degree`. These functions are documented on page ??.)

`\percent` For percent, the raw character is the most flexible way of handling output.

```
1103 \siunitx_declare_unit:Nx \percent { \cs_to_str:N \% }
```

(End definition for `\percent`. This function is documented on page ??.)

`\square` Basic powers.

```

\squared 1104 \siunitx_declare_power:NNn \square \squared { 2 }
\cubic   1105 \siunitx_declare_power:NNn \cubic \cubed { 3 }
\cubed

```

(End definition for `\square` and others. These functions are documented on page ??.)

## 6.12 Messages

```

1106 \msg_new:nnnn { siunitx } { unit / dangling-part }
1107 { Found~#1~part~with~no~unit. }
1108 {
1109   Each~#1~part~must~be~associated~with~a~unit:~a~#1~part~was~found~
1110   but~no~following~unit~was~given.
1111 }
1112 \msg_new:nnnn { siunitx } { unit / duplicate-part }
1113 { Duplicate~#1~part:~#2. }
1114 {
1115   Each~unit~may~have~only~one~#1:\\
1116   the~additional~#1~part~'~#2'~will~be~ignored.
1117 }
1118 \msg_new:nnnn { siunitx } { unit / duplicate-sticky-per }
1119 { Duplicate~\token_to_str:N \per. }

```

```

1120 {
1121   When~the~'sticky-per'~option~is~active,~only~one~
1122   \token_to_str:N \per \ may~appear~in~a~unit.
1123 }
1124 \msg_new:nnnn { siunitx } { unit / literal }
1125 { Literal~units~disabled. }
1126 {
1127   You~gave~the~literal~input~'#1'~
1128   but~literal~unit~output~is~disabled.
1129 }
1130 \msg_new:nnnn { siunitx } { unit / non-convertible-exponent }
1131 { Exponent~'#1'~cannot~be~converted~into~a~symbolic~prefix. }
1132 {
1133   The~exponent~'#1'~does~not~match~with~any~of~the~symbolic~prefixes~
1134   set~up.
1135 }
1136 \msg_new:nnnn { siunitx } { unit / non-numeric-exponent }
1137 { Prefix~'#1'~does~not~have~a~numerical~value. }
1138 {
1139   The~prefix~'#1'~needs~to~be~combined~with~a~number,~but~it~has~no
1140   numerical~value.
1141 }
1142 \msg_new:nnnn { siunitx } { unit / part-before-unit }
1143 { Found~'#1'~part~before~first~unit:~'#2'. }
1144 {
1145   The~'#1'~part~'#2'~must~follow~after~a~unit:~
1146   it~cannot~appear~before~any~units~and~will~therefore~be~ignored.
1147 }

```

## 6.13 Standard settings for module options

Some of these follow naturally from the point of definition (*e.g.* boolean variables are always false to begin with), but for clarity everything is set here.

```

1148 \keys_set:nn { siunitx }
1149 {
1150   bracket-unit-denominator = true      ,
1151   forbid-literal-units     = false     ,
1152   fraction-command         = \frac     ,
1153   inter-unit-product       = \,        ,
1154   extract-mass-in-kilograms = true      ,
1155   parse-units              = true      ,
1156   per-mode                 = power      ,
1157   per-symbol               = /          ,
1158   qualifier-mode           = subscript ,
1159   qualifier-phrase         =           ,
1160   sticky-per               = false     ,
1161   unit-font-command        = \mathrm
1162 }
1163 \</package>

```



## 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>.