

siunitx – A comprehensive (SI) units package*

Joseph Wright[†]

Released 2017/11/26

Contents

I	siunitx – Overall set up	1
1	siunitx implementation	1
1.1	Initial set up	1
1.2	Safety checks	1
1.3	Top-level scratch space	2
1.4	User interfaces	2
1.4.1	Preamble commands	2
1.4.2	Document commands	3
1.4.3	Table column	3
II	siunitx-angle – Formatting angles	6
1	siunitx-angle implementation	6
III	siunitx-number – Parsing and formatting numbers	7
1	siunitx-number implementation	7
1.1	Initial set-up	7
1.2	Main formatting routine	8
1.3	Parsing numbers	8
1.4	Processing numbers	23
1.5	Formatting parsed numbers	25
1.6	Miscellaneous tools	32
1.7	Messages	33
1.8	Standard settings for module options	33
IV	siunitx-print – Printing material with font control	35
0.1	Key-value options	35

*This file describes v3.0alpha, last revised 2017/11/26.

[†]E-mail: joseph.wright@morningstar2.co.uk

1	siunitx-print implementation	37
1.1	Initial set up	37
1.2	Printing routines	38
1.3	Standard settings for module options	44
V	siunitx-table – Formatting numbers in tables	46
1	siunitx-table implementation	46
1.1	Interface functions	46
1.2	Collecting tokens	47
1.3	Separating collected material	48
1.4	Printing numbers in cells: spacing	50
1.5	Printing just text	52
1.6	Reserving space: the table format	52
1.7	Directly printing without collection	52
1.8	Printing numbers in cells: main functions	53
1.9	Standard settings for module options	54
VI	siunitx-unit – Parsing and formatting units	55
1	Formatting units	55
2	Defining symbolic units	56
3	Pre-defined symbolic unit components	57
3.1	Key–value options	60
4	siunitx-unit implementation	62
4.1	Initial set up	62
4.2	Defining symbolic unit	63
4.3	Non-standard symbolic units	65
4.4	Main formatting routine	66
4.5	Formatting literal units	67
4.6	Parsing symbolic units	70
4.7	Formatting parsed units	75
4.8	Pre-defined unit components	84
4.9	Messages	88
4.10	Standard settings for module options	88
VII	siunitx-v1 – Version 1 compatibility	90
1	siunitx-v1 implementation	90
VIII	siunitx-v2 – Version 2 compatibility	91
1	siunitx-v2 implementation	91

Part I

siunitx — Overall set up

1 siunitx implementation

Start the DocStrip guards.

```
1 <*package>
    Identify the internal prefix (LATEX3 DocStrip convention).
2 <@@=siunitx>
```

1.1 Initial set up

Load only the essential support (expl3) “up-front”.

```
3 \RequirePackage{expl3}
    Make sure that the version of l3kernel in use is sufficiently new. This will also trap
    any problems with l3packages (as the two are now tied together, version-wise).
4 \@ifpackagelater {expl3}{2015/11/15}
5 {}
6 {%
7     \PackageError{siunitx} {Support package expl3 too old}
8     {%
9         You need to update your installation of the bundles 'l3kernel' and
10         'l3packages'.\MessageBreak
11         Loading~siunitx~will~abort!%
12     }%
13 \endinput
14 }%
    Identify the package and give the over all version information.
15 \ProvidesExplPackage {siunitx} {2017/11/26} {3.0alpha}
16 {A comprehensive (SI) units package}
```

1.2 Safety checks

`__siunitx_load_check:` There are a number of packages that are incompatible with siunitx as they cover the same concepts and in some cases define the same command names. These are all tested at the point of loading to try to trap issues, and a couple are also tested later as it's possible for them to load without an obvious error if siunitx was loaded first.

The testing here is done in a group so that the tests do not add anything to the hash table.

```
17 \msg_new:nnnn { siunitx } { incompatible-package }
18 { Package~'#1'~incompatible. }
19 { The~#1~package~and~siunitx~are~incompatible. }
20 \cs_new_protected:Npn __siunitx_load_check:n #1
21 {
22     \group_begin:
23     \ifpackageloaded {#1}
24     {
25         \group_end:
```

```

26         \msg_error:nxx { siunitx } { incompatible-package } {#1}
27     }
28     { \group_end: }
29 }
30 \clist_map_function:nN
31   { SIunits , sistyle , unitsdef , fancyunits }
32   \__siunitx_load_check:n
33 \AtBeginDocument
34 {
35     \clist_map_function:nN { SIunits , sistyle }
36     \__siunitx_load_check:n
37 }

```

(End definition for __siunitx_load_check:.)

1.3 Top-level scratch space

\l__siunitx_tmp_tl Scratch space for the interfaces.

```
38 \tl_new:N \l__siunitx_tmp_tl
```

(End definition for \l__siunitx_tmp_tl.)

1.4 User interfaces

The user interfaces are defined in terms of documented code-level ones. This is all done here, and will appear in the .sty file before the relevant code. Things could be rearranged by DocStrip but there is no advantage.

User level interfaces are all created by xparse

```
39 \RequirePackage { xparse }
```

1.4.1 Preamble commands

```

\DeclareSIPower Pass data to the code layer.
\DeclareSIPrefix
\DeclareSIQualifier
\DeclareSIUnit
40 \NewDocumentCommand \DeclareSIPower { +m +m m }
41 {
42     \siunitx_declare_power:Nnn #1 #2 {#3}
43 }
44 \NewDocumentCommand \DeclareSIPrefix { +m m m }
45 {
46     \siunitx_declare_prefix:Nnn #1 {#2} {#3}
47 }
48 \NewDocumentCommand \DeclareSIQualifier { +m m m }
49 {
50     \siunitx_declare_qualifier:Nn #1 {#2}
51 }
52 \NewDocumentCommand \DeclareSIUnit { 0 { } +m m }
53 {
54     \siunitx_declare_unit:Nn #2 {#3}
55 }

```

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

1.4.2 Document commands

`\qty`

```

56 \NewDocumentCommand \qty { 0 { } m m }
57 {
58   \leavevmode
59   \group_begin:
60     \keys_set:nn { siunitx } {#1}
61     \siunitx_number_format:nN {#2} \l__siunitx_tmp_tl
62     \siunitx_print:nV { number } \l__siunitx_tmp_tl
63     \, \nobreak % TEMP
64     \siunitx_unit_format:nN {#3} \l__siunitx_tmp_tl
65     \siunitx_print:nV { unit } \l__siunitx_tmp_tl
66   \group_end:
67 }

```

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

`\num` All of a standard form: start a paragraph (if required), set local key values, do the
`\unit` formatting, print the result.

```

68 \NewDocumentCommand \num { 0 { } m }
69 {
70   \leavevmode
71   \group_begin:
72     \keys_set:nn { siunitx } {#1}
73     \siunitx_number_format:nN {#2} \l__siunitx_tmp_tl
74     \siunitx_print:nV { number } \l__siunitx_tmp_tl
75   \group_end:
76 }
77 \NewDocumentCommand \unit { 0 { } m }
78 {
79   \leavevmode
80   \group_begin:
81     \keys_set:nn { siunitx } {#1}
82     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
83     \siunitx_print:nV { unit } \l__siunitx_tmp_tl
84   \group_end:
85 }

```

(End definition for \num and \unit. These functions are documented on page ??.)

`\sisetup` A very thin wrapper.

```

86 \NewDocumentCommand \sisetup { m }
87 { \keys_set:nn { siunitx } {#1} }

```

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

1.4.3 Table column

User interfaces in tabular constructs are provided using the mechanisms from the `array` package.

```

88 \RequirePackage { array }

```

There is a slight problem with the `cellspace` package: it also uses `S` for a column type. Here, `S` seems to make more sense for `siunitx`, with `C` used for `cellspace`. To enable this to work well, the column rewriting code happens `\AtBeginDocument`. The `S` column is deleted from `\NC@list` so that no warning appears.

```

89 \AtBeginDocument
90 {
91   \@ifpackageloaded { cellspace }
92   {
93     \newcolumnntype { C } [ 1 ]
94     { > { \bcolumn #1 \@nil } #1 < { \ecolumn } }
95     \cs_set:Npn \__siunitx_tmp:w #1 \NC@do S #2 \q_stop
96     { \NC@list { #1 #2 } }
97     \exp_after:wN \__siunitx_tmp:w \tex_the:D \NC@list \q_stop
98     \cs_undefine:N \NC@find@S
99     \msg_warning:nnn { siunitx } { moved-cellspace-column } { C }
100    \ifcellspace@m
101      \def \env@matrix
102      {
103        \hskip -\arraycolsep
104        \let \@ifnextchar \new@ifnextchar
105        \array
106        {
107          * { \c@MaxMatrixCols }
108          { > { \bcolumn c \@nil $ } c < { $ \ecolumn } } @ { }
109        }
110      }
111    \fi
112  }
113  { }
114 }
115 \msg_new:nnn { siunitx } { moved-cellspace-column }
116 { Column~type~for~cellspace~package~moved~to~'#1'. }

```

`__siunitx_declare_column:Nnn` Creating numerical columns requires that these are declared before anything else in `\NC@list`: this is necessary to work with optional arguments. This means a bit of manual effort after the simple declaration of a new column type. The token assigned to the column type is not fixed as this allows the same code to be used in compatibility with version 2.

```

117 \cs_new_protected:Npn \__siunitx_declare_column:Nnn #1#2#3
118 {
119   \newcolumnntype {#1} { }
120   \cs_set_protected:Npn \__siunitx_tmp:w \NC@do ##1##2 \NC@do #1
121   { \NC@list { \NC@do ##1 \NC@do #1 ##2 } }
122   \exp_after:wN \__siunitx_tmp:w \tex_the:D \NC@list
123   \exp_args:Nnc \renewcommand * { \NC@rewrite@ #1 } [ 1 ] [ ]
124   {
125     \@temptokena \expandafter
126     {
127       \the \@temptokena
128       > {#2} c < {#3}
129     }
130     \NC@find
131   }

```

```
132 }
```

When `mdwtab` is loaded the syntax required is slightly different.

```
133 \AtBeginDocument
134 {
135   \@ifpackageloaded { mdwtab }
136   {
137     \cs_set_protected:Npn \__siunitx_declare_column:Nnn #1#2#3
138     {
139       \newcolumnntype {#1} [ 1 ] [ ]
140       { > {#2} c < {#3} }
141     }
142   }
143   { }
144 }
145 \AtBeginDocument
146 {
147   \__siunitx_declare_column:Nnn n
148   {
149     \keys_set:nn { siunitx } {#1}
150     \siunitx_cell_begin:
151   }
152   { \siunitx_cell_end: }
153 }
```

(End definition for __siunitx_declare_column:Nnn.)

```
154 \endpackage
```


Part II

siunitx-angle – Formatting angles

1 siunitx-angle implementation

Start the DocStrip guards.

¹ `*package`

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

² `\@@=siunitx_angle`

³ `\package`

Part III

siunitx-number – Parsing and formatting numbers

```
\siunitx_number_format:nN \siunitx_number_format:nN {\langle number \rangle} \langle tl var \rangle
\siunitx_number_format:VN
```

```
\siunitx_number_format:nNN \siunitx_number_format:nNN {\langle number \rangle} \langle tl var \rangle \langle marker \rangle
```

```
\siunitx_if_number:nTF \siunitx_if_number_token:NTF {\langle tokens \rangle}
{\langle true code \rangle} {\langle false code \rangle}
```

Determines if the $\langle tokens \rangle$ form a valid number which can be fully parsed by `siunitx`.

```
\siunitx_if_number_token:NTF \siunitx_if_number_token:NTF {\langle token \rangle}
{\langle true code \rangle} {\langle false code \rangle}
```

Determines if the $\langle token \rangle$ is valid in a number based on those tokens currently set up for detection in a number.

1 siunitx-number implementation

Start the DocStrip guards.

¹ $\langle *package \rangle$

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

² $\langle @@=siunitx_{\text{n}}umber \rangle$

1.1 Initial set-up

Variants not provided by `expl3`.

```
3 \cs_generate_variant:Nn \tl_if_blank:nTF { f }
4 \cs_generate_variant:Nn \tl_if_blank_p:n { f }
5 \cs_generate_variant:Nn \tl_if_in:NnTF { NV }
```

$\backslash l_siunitx_number_tmp_tl$ Scratch space.

⁶ $\backslash tl_new:N \backslash l_siunitx_number_tmp_tl$

(End definition for $\backslash l_siunitx_number_tmp_tl$.)

1.2 Main formatting routine

`\l_siunitx_number_formatted_tl` A token list for the final formatted result: may or may not be generated by the parser, depending on settings which are active.

7 `\tl_new:N \l_siunitx_number_formatted_tl`

(End definition for `\l_siunitx_number_formatted_tl`.)

`\l__siunitx_number_tab_tl` A token list for marking the position of tabular alignments in formatted output.

8 `\tl_new:N \l__siunitx_number_tab_tl`

(End definition for `\l__siunitx_number_tab_tl`.)

`\siunitx_number_format:nN`

`\siunitx_number_format:VN`

`\siunitx_number_format:nNN`

`__siunitx_number_format:nN`

9 `\cs_new_protected:Npn \siunitx_number_format:nN #1#2`

10 `{`

11 `\tl_clear:N \l__siunitx_number_tab_tl`

12 `__siunitx_number_format:nN {#1} #2`

13 `}`

14 `\cs_generate_variant:Nn \siunitx_number_format:nN { V }`

15 `\cs_new_protected:Npn \siunitx_number_format:nNN #1#2#3`

16 `{`

17 `\tl_set:Nn \l__siunitx_number_tab_tl {#3}`

18 `__siunitx_number_format:nN {#1} #2`

19 `}`

20 `\cs_new_protected:Npn __siunitx_number_format:nN #1#2`

21 `{`

22 `\group_begin:`

23 `__siunitx_number_parse:n {#1}`

24 `__siunitx_number_format:`

25 `\exp_args:NNNV \group_end:`

26 `\tl_set:Nn #2 \l_siunitx_number_formatted_tl`

27 `}`

(End definition for `\siunitx_number_format:nN`, `\siunitx_number_format:nNN`, and `__siunitx_number_format:nN`. These functions are documented on page 7.)

1.3 Parsing numbers

Before numbers can be manipulated or formatted they need to be parsed into an internal form. In particular, if multiple code paths are to be avoided, it is necessary to do such parsing even for relatively simple cases such as converting `1e10` to `1 \times 10^{\{10\}}`.

Storing the result of such parsing can be done in a number of ways. In the first version of `siunitx` a series of separate data stores were used. This is potentially quite fast (as recovery of items relies only on `TeX`'s hash table) but makes managing the various data entries somewhat tedious and error-prone. For version two of the package, a single data structure (property list) was used for each part of the parsed number. Whilst this is easy to manage and extend, it is somewhat slower as at a `TeX` level there are repeated pack-unpack steps. In particular, the fact that there are a limited number of items to track for a “number” means that a more efficient approach is desirable (contrast parsing units, which is open-ended and therefore fits well with using a property list).

To allow for complex numbers, two parallel data structures are used, one for the real part and one for the imaginary part. If the part is entirely absent then the data structures are left empty. Within each part, the structure is

$$\{\langle comparator \rangle\} \langle sign \rangle \{\langle integer \rangle\} \{\langle decimal \rangle\} \{\langle uncertainty \rangle\} \langle exponent sign \rangle \{\langle exponent \rangle\}$$

where the two sign parts must be single tokens and all other components must be given in braces. *All* of the components must be present in a stored number (*i.e.* at the end of parsing). The number must have at least one digit for both the $\langle integer \rangle$ and $\langle exponent \rangle$ parts.

Options which determine the various valid parts of a parsed number.

```

\l_siunitx_number_expression_bool
\l_siunitx_number_input_uncert_close_tl
\l_siunitx_number_input_complex_tl
\l_siunitx_number_input_comparator_tl
\l_siunitx_number_input_decimal_tl
\l_siunitx_number_input_digit_tl
\l_siunitx_number_input_exponent_tl
\l_siunitx_number_input_ignore_tl
\l_siunitx_number_input_uncert_open_tl
\l_siunitx_number_input_sign_tl
\l_siunitx_number_input_uncert_sign_tl
\l_siunitx_number_expression:n

28 \keys_define:nn { siunitx }
29 {
30   evaluate-expression .bool_set:N =
31     \l_siunitx_number_expression_bool ,
32   expression .code:n =
33     \cs_set:Npn \l_siunitx_number_expression:n ##1 {#1} ,
34   input-close-uncertainty .tl_set:N =
35     \l_siunitx_number_input_uncert_close_tl ,
36   input-complex-roots .tl_set:N =
37     \l_siunitx_number_input_complex_tl ,
38   input-comparators .tl_set:N =
39     \l_siunitx_number_input_comparator_tl ,
40   input-decimal-markers .tl_set:N =
41     \l_siunitx_number_input_decimal_tl ,
42   input-digits .tl_set:N =
43     \l_siunitx_number_input_digit_tl ,
44   input-exponent-markers .tl_set:N =
45     \l_siunitx_number_input_exponent_tl ,
46   input-ignore .tl_set:N =
47     \l_siunitx_number_input_ignore_tl ,
48   input-open-uncertainty .tl_set:N =
49     \l_siunitx_number_input_uncert_open_tl ,
50   input-signs .tl_set:N =
51     \l_siunitx_number_input_sign_tl ,
52   input-uncertainty-signs .code:n =
53     {
54       \tl_set:Nn \l_siunitx_number_input_uncert_sign_tl {#1}
55       \tl_map_inline:nn {#1}
56       {
57         \tl_if_in:NnF \l_siunitx_number_input_sign_tl {##1}
58         { \tl_put_right:Nn \l_siunitx_number_input_sign_tl {##1} }
59       }
60     }
61   }
62 \cs_new:Npn \l_siunitx_number_expression:n #1 { }

```

(End definition for $\l_siunitx_number_expression_bool$ and others.)

$\l_siunitx_number_arg_tl$ The input argument or a part thereof, depending on the position in the parsing routine.

```
63 \tl_new:N \l_siunitx_number_arg_tl
```

(End definition for $\l_siunitx_number_arg_tl$.)

$\l_siunitx_number_comparator_tl$ A comparator, if found, is held here.

```
64 \tl_new:N \l_siunitx_number_comparator_tl
```

(End definition for \l__siunitx_number_comparator_tl.)

\l__siunitx_number_exponent_tl The exponent part of a parsed number. It is easiest to find this relatively early in the parsing process, but as it needs to go at the end of the internal format is held separately until required.

65 \tl_new:N \l__siunitx_number_exponent_tl

(End definition for \l__siunitx_number_exponent_tl.)

\l__siunitx_number_flex_tl When parsing for a separate uncertainty or complex number, the nature of the grabbed part cannot be determined until the end of the number. To avoid abusing the storage areas, this dedicated one is used for “flexible” cases.

66 \tl_new:N \l__siunitx_number_flex_tl

(End definition for \l__siunitx_number_flex_tl.)

\l__siunitx_number_imaginary_tl Used to hold the real and imaginary parts of a number in the standardised format.

\l__siunitx_number_real_tl 67 \tl_new:N \l__siunitx_number_imaginary_tl

68 \tl_new:N \l__siunitx_number_real_tl

(End definition for \l__siunitx_number_imaginary_tl and \l__siunitx_number_real_tl.)

\l__siunitx_number_input_tl The numerical input exactly as given by the user.

69 \tl_new:N \l__siunitx_number_input_tl

(End definition for \l__siunitx_number_input_tl.)

\l__siunitx_number_partial_tl To avoid needing to worry about the fact that the final data stores are somewhat tricky to add to token-by-token, a simple store is used to build up the parsed part of a number before transferring in one go.

70 \tl_new:N \l__siunitx_number_partial_tl

(End definition for \l__siunitx_number_partial_tl.)

\l__siunitx_number_validate_bool Used to set up for validation with no error production.

71 \bool_new:N \l__siunitx_number_validate_bool

(End definition for \l__siunitx_number_validate_bool.)

__siunitx_number_parse:n After some initial set up, the parser expands the input and then replaces as far as possible tricky tokens with ones that can be handled using delimited arguments. The parser begins with the assumption that the input is a real number. To avoid multiple conditionals here, the parser is set up as a chain of commands initially, with a loop only later. This avoids more conditionals than are necessary.

72 **q**

73 \cs_new_protected:Npn __siunitx_number_parse:n #1

74 {

75 \tl_clear:N \l__siunitx_number_imaginary_tl

76 \tl_clear:N \l__siunitx_number_real_tl

77 \protected@edef \l__siunitx_number_arg_tl

78 {

79 \bool_if:NTF \l__siunitx_number_expression_bool

80 { \fp_eval:n { __siunitx_number_expression:n {#1} } }

81 {#1}

```

82     }
83     \tl_set_eq:NW \l__siunitx_number_input_tl \l__siunitx_number_arg_tl
84     \__siunitx_number_parse_replace:
85     \tl_if_empty:NF \l__siunitx_number_arg_tl
86     { \__siunitx_number_parse_comparator: }
87     \__siunitx_number_parse_check:
88 }

```

(End definition for __siunitx_number_parse:n.)

__siunitx_number_parse_check: After the loop there is one case that might need tidying up. If a separated uncertainty was found it will be currently in \l__siunitx_number_flex_tl and needs moving. A series of tests pick up that case, then the check is made that some content was found for at least one of the real or imaginary parts of the number.

```

89 \cs_new_protected:Npn \__siunitx_number_parse_check:
90 {
91     \tl_if_empty:NF \l__siunitx_number_flex_tl
92     {
93         \bool_lazy_and:nnTF
94         {
95             \tl_if_blank_p:f
96             { \exp_after:wN \use_iv:nnnn \l__siunitx_number_real_tl }
97         }
98         {
99             \tl_if_blank_p:f
100             { \exp_after:wN \use_iv:nnnn \l__siunitx_number_flex_tl }
101         }
102         {
103             \tl_set:Nx \l__siunitx_number_tmp_tl
104             { \exp_after:wN \use_i:nnnn \l__siunitx_number_flex_tl }
105             \tl_if_in:NVTF \l__siunitx_number_input_uncert_sign_tl
106             \l__siunitx_number_tmp_tl
107             { \__siunitx_number_parse_combine_uncert: }
108             { \tl_clear:N \l__siunitx_number_real_tl }
109         }
110         { \tl_clear:N \l__siunitx_number_real_tl }
111     }
112     \bool_lazy_and:nnTF
113     { \tl_if_empty_p:N \l__siunitx_number_real_tl }
114     { \tl_if_empty_p:N \l__siunitx_number_imaginary_tl }
115     {
116         \bool_if:NF \l__siunitx_number_validate_bool
117         {
118             \msg_error:nnx { siunitx } { number / invalid-input }
119             { \exp_not:V \l__siunitx_number_input_tl }
120         }
121     }
122     { \__siunitx_number_parse_finalise: }
123 }

```

(End definition for __siunitx_number_parse_check:.)

__siunitx_number_parse_combine_uncert: Conversion of a second numerical part to an uncertainty needs a bit of work. The first step is to extract the useful information from the two stores: the sign, integer and decimal

__siunitx_number_parse_combine_uncert_auxi:NnnnNnnn
__siunitx_number_parse_combine_uncert_auxii:nnnnn
__siunitx_number_parse_combine_uncert_auxiii:fnnnn
__siunitx_number_parse_combine_uncert_auxiiii:nnnnnn
__siunitx_number_parse_combine_uncert_auxv:nnnn
__siunitx_number_parse_combine_uncert_auxvi:w

parts from the real number and the integer and decimal parts from the second number. That is done using the input stack to avoid lots of assignments.

```

124 \cs_new_protected:Npn \__siunitx_number_parse_combine_uncert:
125 {
126   \exp_after:wN \exp_after:wN \exp_after:wN
127   \__siunitx_number_parse_combine_uncert_auxi:NnnnNnnn
128   \exp_after:wN \l__siunitx_number_real_tl \l__siunitx_number_flex_tl
129 }

```

Here, #4, #5 and #8 are all junk arguments simply there to mop up tokens, while #1 will be recovered later from \l__siunitx_number_real_tl so does not need to be passed about. The difference in places between the two decimal parts is now found: this is done just once to avoid having to parse token lists twice. The value is then used to generate a number of filler 0 tokens, and these are added to the appropriate part of the number. Finally, everything is recombined: the integer part only needs a test to avoid an empty main number.

```

130 \cs_new_protected:Npn
131   \__siunitx_number_parse_combine_uncert_auxi:NnnnNnnn #1#2#3#4#5#6#7#8
132 {
133   \int_compare:nNnTF { \tl_count:n {#6} } > { \tl_count:n {#2} } {
134     {
135       \tl_clear:N \l__siunitx_number_real_tl
136       \tl_clear:N \l__siunitx_number_flex_tl
137     }
138     {
139       \__siunitx_number_parse_combine_uncert_auxii:fnnnn
140       { \int_eval:n { \tl_count:n {#3} - \tl_count:n {#7} } } {
141         {#2} {#3} {#6} {#7}
142       }
143     }
144   }
145   \cs_new_protected:Npn
146     \__siunitx_number_parse_combine_uncert_auxii:nnnnn #1
147   {
148     \__siunitx_number_parse_combine_uncert_auxiii:fnnnnn
149     { \prg_replicate:nn { \int_abs:n {#1} } { 0 } } {#1}
150   }
151   \cs_generate_variant:Nn \__siunitx_number_parse_combine_uncert_auxii:nnnnn { f }
152   \cs_new_protected:Npn
153     \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn #1#2#3#4#5#6
154   {
155     \int_compare:nNnTF {#2} > 0
156     {
157       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
158       {#3} {#4} {#5} { #6 #1 }
159     }
160     {
161       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
162       {#3} { #4 #1 } {#5} {#6}
163     }
164   }
165   \cs_generate_variant:Nn
166     \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn { f }
167   \cs_new_protected:Npn

```

```

168 \__siunitx_number_parse_combine_uncert_auxiv:nnnn #1#2#3#4
169 {
170   \tl_set:Nx \l__siunitx_number_real_tl
171   {
172     \tl_head:V \l__siunitx_number_real_tl
173     { \exp_not:n {#1} }
174     {
175       \bool_lazy_and:nnTF
176       { \tl_if_blank_p:n {#2} }
177       { ! \tl_if_blank_p:n {#4} }
178       { 0 }
179       { \exp_not:n {#2} }
180     }
181     {
182       \__siunitx_number_parse_combine_uncert_auxv:w #3#4
183       \q_recursion_tail \q_recursion_stop
184     }
185   }
186 }

```

A short routine to remove any leading zeros in the uncertainty part, which are not needed for the compact representation used by the module.

```

187 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxv:w #1
188 {
189   \quark_if_recursion_tail_stop:N #1
190   \str_if_eq:nnTF {#1} { 0 }
191   { \__siunitx_number_parse_combine_uncert_auxv:w }
192   { \__siunitx_number_parse_combine_uncert_auxvi:w #1 }
193 }
194 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxvi:w
195 #1 \q_recursion_tail \q_recursion_stop
196 { \exp_not:n {#1} }

```

(End definition for `__siunitx_number_parse_combine_uncert:` and others.)

`__siunitx_number_parse_comparator:`
`__siunitx_number_parse_comparator_aux:Nw`

A comparator has to be the very first token in the input. A such, the test for this can be very fast: grab the first token, do a check and if appropriate store the result.

```

197 \cs_new_protected:Npn \__siunitx_number_parse_comparator:
198 {
199   \exp_after:wN \__siunitx_number_parse_comparator_aux:Nw
200   \l__siunitx_number_arg_tl \q_stop
201 }
202 \cs_new_protected:Npn \__siunitx_number_parse_comparator_aux:Nw #1#2 \q_stop
203 {
204   \tl_if_in:NnTF \l__siunitx_number_input_comparator_tl {#1}
205   {
206     \tl_set:Nn \l__siunitx_number_comparator_tl {#1}
207     \tl_set:Nn \l__siunitx_number_arg_tl {#2}
208   }
209   { \tl_clear:N \l__siunitx_number_comparator_tl }
210   \tl_if_empty:NF \l__siunitx_number_arg_tl
211   { \__siunitx_number_parse_sign: }
212 }

```

(End definition for `__siunitx_number_parse_comparator:` and `__siunitx_number_parse_comparator_aux:Nw`.)

`_siunitx_number_parse_exponent:` An exponent part of a number has to come at the end and can only occur once. Thus it
`_siunitx_number_parse_exponent_aux:w` is relatively easy to parse. First, there is a check that an exponent part is allowed, and
`_siunitx_number_parse_exponent_aux:nn` if so a split is made (the previous part of the chain checks that there is some content in
`_siunitx_number_parse_exponent_aux:Nw` `\l__siunitx_number_arg_tl` before calling this function). After splitting, if there is no
`_siunitx_number_parse_exponent_aux:Nn` exponent then simply save a default. Otherwise, check for a sign and then store either
`_siunitx_number_parse_exponent_zero_test:N` this or an assumed + and the digits after a check that nothing else is present after the `e`.
`_siunitx_number_parse_exponent_check:N` The only slight complication to all of this is allowing an arbitrary token in the input to
`_siunitx_number_parse_exponent_cleanup:N` represent the exponent: this is done by setting any exponent tokens to the first of the
allowed list, then using that in a delimited argument set up. Once an exponent part is
found, there is a loop to check that each of the tokens is a digit then a tidy up step to
remove any leading zeros.

```

213 \cs_new_protected:Npn \_siunitx_number_parse_exponent:
214 {
215   \tl_if_empty:NTF \l__siunitx_number_input_exponent_tl
216   { \tl_set:Nn \l__siunitx_number_exponent_tl { +0 } }
217   {
218     \tl_set:Nx \l__siunitx_number_tmp_tl
219     { \tl_head:V \l__siunitx_number_input_exponent_tl }
220     \tl_map_inline:Nn \l__siunitx_number_input_exponent_tl
221     {
222       \tl_replace_all:NnV \l__siunitx_number_arg_tl
223       {##1} \l__siunitx_number_tmp_tl
224     }
225     \use:x
226     {
227       \cs_set_protected:Npn
228       \exp_not:N \_siunitx_number_parse_exponent_aux:w
229       ###1 \exp_not:V \l__siunitx_number_tmp_tl
230       ###2 \exp_not:V \l__siunitx_number_tmp_tl
231       ###3 \exp_not:N \q_stop
232     }
233     { \_siunitx_number_parse_exponent_aux:nn {##1} {##2} }
234     \use:x
235     {
236       \_siunitx_number_parse_exponent_aux:w
237       \exp_not:V \l__siunitx_number_arg_tl
238       \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_nil
239       \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_stop
240     }
241   }
242 }
243 \cs_new_protected:Npn \_siunitx_number_parse_exponent_aux:w { }
244 \cs_new_protected:Npn \_siunitx_number_parse_exponent_aux:nn #1#2
245 {
246   \quark_if_nil:nTF {#2}
247   { \tl_set:Nn \l__siunitx_number_exponent_tl { +0 } }
248   {
249     \tl_set:Nn \l__siunitx_number_arg_tl {#1}
250     \tl_if_blank:nTF {#2}
251     { \tl_clear:N \l__siunitx_number_real_tl }
252     { \_siunitx_number_parse_exponent_aux:Nw #2 \q_stop }
253   }
254   \tl_if_empty:NF \l__siunitx_number_real_tl

```

```

255     { \_siunitx_number_parse_loop: }
256   }
257 \cs_new_protected:Npn \_siunitx_number_parse_exponent_aux:Nw #1#2 \q_stop
258 {
259   \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#1}
260     { \_siunitx_number_parse_exponent_aux:Nn #1 {#2} }
261     { \_siunitx_number_parse_exponent_aux:Nn + {#1#2} }
262   \tl_if_empty:NT \l__siunitx_number_exponent_tl
263     { \tl_clear:N \l__siunitx_number_real_tl }
264 }
265 \cs_new_protected:Npn \_siunitx_number_parse_exponent_aux:Nn #1#2
266 {
267   \tl_set:Nn \l__siunitx_number_exponent_tl { #1 }
268   \tl_if_blank:nTF {#2}
269     { \tl_clear:N \l__siunitx_number_real_tl }
270     {
271       \_siunitx_number_parse_exponent_zero_test:N #2
272       \q_recursion_tail \q_recursion_stop
273     }
274 }
275 \cs_new_protected:Npn \_siunitx_number_parse_exponent_zero_test:N #1
276 {
277   \quark_if_recursion_tail_stop_do:Nn #1
278     { \tl_set:Nn \l__siunitx_number_exponent_tl { +0 } }
279   \str_if_eq:nnTF {#1} { 0 }
280     { \_siunitx_number_parse_exponent_zero_test:N }
281     { \_siunitx_number_parse_exponent_check:N #1 }
282 }
283 \cs_new_protected:Npn \_siunitx_number_parse_exponent_check:N #1
284 {
285   \quark_if_recursion_tail_stop:N #1
286   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#1}
287     {
288       \tl_put_right:Nn \l__siunitx_number_exponent_tl {#1}
289       \_siunitx_number_parse_exponent_check:N
290     }
291     { \_siunitx_number_parse_exponent_cleanup:wN }
292 }
293 \cs_new_protected:Npn \_siunitx_number_parse_exponent_cleanup:wN
294   #1 \q_recursion_stop
295   { \tl_clear:N \l__siunitx_number_real_tl }

```

There are two parts to the replacement code. First, any active hyphens signs are normalised: these can come up with some packages and cause issues. Multi-token signs then are converted to the single token equivalents so that everything else can work on a one token basis.

```

\_siunitx_number_parse_replace:
\_siunitx_number_parse_replace_aux:nN
\_siunitx_number_parse_replace_sign:
\_c_siunitx_number_parse_sign_replacement_tl
296 \cs_new_protected:Npn \_siunitx_number_parse_replace:
297 {
298   \_siunitx_number_parse_replace_minus:
299   \exp_last_unbraced:NV \_siunitx_number_parse_replace_aux:nN
300   \c_siunitx_number_parse_sign_replacement_tl
301   { ? } \q_recursion_tail
302   \q_recursion_stop
303 }

```

```

304 \cs_set_protected:Npn \__siunitx_number_parse_replace_aux:nN #1#2
305 {
306   \quark_if_recursion_tail_stop:N #2
307   \tl_replace_all:Nnn \l__siunitx_number_arg_tl {#1} {#2}
308   \__siunitx_number_parse_replace_aux:nN
309 }
310 \tl_const:Nn \c__siunitx_number_parse_sign_replacement_tl
311 {
312   { -+ } \mp
313   { +- } \pm
314   { << } \ll
315   { <= } \le
316   { >> } \gg
317   { >= } \ge
318 }
319 \group_begin:
320   \char_set_catcode_active:N \-
321   \cs_new_protected:Npx \__siunitx_number_parse_replace_minus:
322   {
323     \tl_replace_all:Nnn \exp_not:N \l__siunitx_number_arg_tl
324     { \exp_not:N - } { \token_to_str:N - }
325   }
326 \group_end:

```

(End definition for __siunitx_number_parse_exponent: and others.)

__siunitx_number_parse_finalise: Combine all of the bits of a number together: both the real and imaginary parts contain all of the data.

```

\__siunitx_number_parse_finalise_aux:N
\__siunitx_number_parse_finalise_aux:Nw
327 \cs_new_protected:Npn \__siunitx_number_parse_finalise:
328 {
329   \__siunitx_number_parse_finalise_aux:N \l__siunitx_number_real_tl
330   \__siunitx_number_parse_finalise_aux:N \l__siunitx_number_imaginary_tl
331 }
332 \cs_new_protected:Npn \__siunitx_number_parse_finalise_aux:N #1
333 {
334   \tl_if_empty:NF #1
335   {
336     \tl_set:Nx #1
337     {
338       { \exp_not:V \l__siunitx_number_comparator_tl }
339       \exp_not:V #1
340       \exp_after:wN \__siunitx_number_parse_finalise_aux:Nw
341       \l__siunitx_number_exponent_tl \q_stop
342     }
343   }
344 }
345 \cs_new:Npn \__siunitx_number_parse_finalise_aux:Nw #1#2 \q_stop
346 {
347   \exp_not:N #1
348   { \exp_not:n {#2} }
349 }

```

(End definition for __siunitx_number_parse_finalise:, __siunitx_number_parse_finalise_aux:N, and __siunitx_number_parse_finalise_aux:Nw.)

```

    \_siunitx_number_parse_loop:
    \_siunitx_number_parse_loop_first:N
    \_siunitx_number_parse_loop_main:NNNNN
    \_siunitx_number_parse_loop_main_end:NN
    \_siunitx_number_parse_loop_main_digit:NNNNN
    \_siunitx_number_parse_loop_main_decimal:NN
    \_siunitx_number_parse_loop_main_uncert:NNN
    \_siunitx_number_parse_loop_main_complex:N
    \_siunitx_number_parse_loop_main_sign:NNN
    \_siunitx_number_parse_loop_main_store:NNN
    \_siunitx_number_parse_loop_after_decimal:NNN
    \_siunitx_number_parse_loop_uncert:NNNNN
    \_siunitx_number_parse_loop_after_uncert:NNN
    \_siunitx_number_parse_loop_root_swap:NNwNN
    \_siunitx_number_parse_loop_complex_cleanup:wN
    \_siunitx_number_parse_loop_break:wN

```

At this stage, the partial input `\l__siunitx_number_arg_tl` will contain any mantissa, which may contain an uncertainty or complex part. Parsing this and allowing for all of the different formats possible is best done using a token-by-token approach. However, as at each stage only a subset of tokens are valid, the approach take is to use a set of semi-dedicated functions to parse different components along with switches to allow a sensible amount of code sharing.

```

350 \cs_new_protected:Npn \_siunitx_number_parse_loop:
351 {
352   \tl_clear:N \l__siunitx_number_partial_tl
353   \exp_after:wN \_siunitx_number_parse_loop_first:NNN
354   \exp_after:wN \l__siunitx_number_real_tl \exp_after:wN \c_true_bool
355   \l__siunitx_number_arg_tl
356   \q_recursion_tail \q_recursion_stop
357 }

```

The very first token of the input is handled with a dedicated function. Valid cases here are

- Entirely blank if the original input was for example `+e10`: simply clean up if in the integer part of issue an error if in a second part (complex number, *etc.*).
- An integer part digit: pass through to the main collection routine.
- A decimal marker: store an empty integer part and move to the main collection routine for a decimal part.
- A complex root token: shuffle to the end of the input.

Anything else is invalid and sends the code to the abort function.

```

358 \cs_new_protected:Npn \_siunitx_number_parse_loop_first:NNN #1#2#3
359 {
360   \quark_if_recursion_tail_stop_do:Nn #3
361   {
362     \bool_if:NTF #2
363     { \tl_put_right:Nn #1 { { 1 } { } { } } }
364     { \_siunitx_number_parse_loop_break:wN \q_recursion_stop }
365   }
366   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#3}
367   {
368     \_siunitx_number_parse_loop_main:NNNNN
369     #1 \c_true_bool \c_false_bool #2 #3
370   }
371   {
372     \tl_if_in:NnTF \l__siunitx_number_input_decimal_tl {#3}
373     {
374       \tl_put_right:Nn #1 { { 0 } }
375       \_siunitx_number_parse_loop_after_decimal:NNN #1 #2
376     }
377     {
378       \tl_if_in:NnTF \l__siunitx_number_input_complex_tl {#3}
379       { \_siunitx_number_parse_loop_root_swap:NNwNN #1 #3 }
380       { \_siunitx_number_parse_loop_break:wN }
381     }
382   }
383 }

```

A single function is used to cover the “main” part of numbers: finding real, complex or separated uncertainty parts and covering both the integer and decimal components. This works because these elements share a lot of concepts: a small number of switches can be used to differentiate between them. To keep the code at least somewhat readable, this main function deals with the validity testing but hands off other tasks to dedicated auxiliaries for each case.

The possibilities are

- The number terminates, meaning that some digits were collected and everything is simply tidied up (as far as the loop is concerned).
- A digit is found: this is the common case and leads to a storage auxiliary (which handles non-significant zeros).
- A decimal marker is found: only valid in the integer part and there leading to a store-and-switch situation.
- An open-uncertainty token: switch to the dedicated collector for uncertainties.
- A complex root token: store the current number as an imaginary part and terminate the loop.
- A sign token (if allowed): stop collecting this number and restart collection for the second part.

```

384 \cs_new_protected:Npn \__siunitx_number_parse_loop_main:NNNNN #1#2#3#4#5
385 {
386   \quark_if_recursion_tail_stop_do:Nn #5
387   { \__siunitx_number_parse_loop_main_end:NN #1#2 }
388   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#5}
389   { \__siunitx_number_parse_loop_main_digit:NNNNN #1#2#3#4#5 }
390   {
391     \tl_if_in:NnTF \l__siunitx_number_input_decimal_tl {#5}
392     {
393       \bool_if:NTF #2
394       { \__siunitx_number_parse_loop_main_decimal:NN #1 #4 }
395       { \__siunitx_number_parse_loop_break:wN }
396     }
397     {
398       \tl_if_in:NnTF \l__siunitx_number_input_uncert_open_tl {#5}
399       { \__siunitx_number_parse_loop_main_uncert:NNN #1#2 #4 }
400       {
401         \tl_if_in:NnTF \l__siunitx_number_input_complex_tl {#5}
402         {
403           \__siunitx_number_parse_loop_main_store:NNN
404           #1 #2 \c_true_bool
405           \__siunitx_number_parse_loop_main_complex:N #1
406         }
407         {
408           \bool_if:NTF #4
409           {
410             \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#5}
411             {
412               \__siunitx_number_parse_loop_main_sign:NNN
413               #1#2 #5

```

```

414         }
415         { \_siunitx_number_parse_loop_break:wN }
416     }
417     { \_siunitx_number_parse_loop_break:wN }
418 }
419 }
420 }
421 }
422 }

```

If the main loop finds the end marker then there is a tidy up phase. The current partial number is stored either as the integer or decimal, depending on the setting for the indicator switch. For the integer part, if no number has been collected then one or more non-significant zeros have been dropped. Exactly one zero is therefore needed to make sure the parsed result is correct.

```

423 \cs_new_protected:Npn \_siunitx_number_parse_loop_main_end:NN #1#2
424 {
425     \bool_lazy_and:nnT
426     {#2} { \tl_if_empty_p:N \l__siunitx_number_partial_tl }
427     { \tl_set:Nn \l__siunitx_number_partial_tl { 0 } }
428     \tl_put_right:Nx #1
429     {
430         { \exp_not:V \l__siunitx_number_partial_tl }
431         \bool_if:NT #2 { { } }
432         { }
433     }
434 }

```

The most common case for the main loop collector is to find a digit. Here, in the integer part it is possible that zeros are non-significant: that is handled using a combination of a switch and a string test. Other than that, the situation here is simple: store the input and loop.

```

435 \cs_new_protected:Npn \_siunitx_number_parse_loop_main_digit:NNNN #1#2#3#4#5
436 {
437     \bool_lazy_or:nnTF
438     {#3} { ! \str_if_eq_p:nn {#5} { 0 } }
439     {
440         \tl_put_right:Nn \l__siunitx_number_partial_tl {#5}
441         \_siunitx_number_parse_loop_main:NNNN #1 #2 \c_true_bool #4
442     }
443     { \_siunitx_number_parse_loop_main:NNNN #1 #2 \c_false_bool #4 }
444 }

```

When a decimal marker was found, move the integer part to the store and then go back to the loop with the flags set correctly. There is the case of non-significant zeros to cover before that, of course.

```

445 \cs_new_protected:Npn \_siunitx_number_parse_loop_main_decimal:NN #1#2
446 {
447     \_siunitx_number_parse_loop_main_store:NNN #1 \c_false_bool \c_false_bool
448     \_siunitx_number_parse_loop_after_decimal:NNN #1 #2
449 }

```

Starting an uncertainty part means storing the number to date as in other cases, with the possibility of a blank decimal part allowed for. The uncertainty itself is collected by a dedicated function as it is extremely restricted.

```

450 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_uncert:NNN #1#2#3
451 {
452   \__siunitx_number_parse_loop_main_store:NNN #1 #2 \c_false_bool
453   \__siunitx_number_parse_loop_uncert:NNNNN
454   #1 \c_true_bool \c_false_bool #3
455 }

```

A complex root token has to be at the end of the input (leading ones are dealt with specially). Thus after moving the data to the correct place there is a hand-off to a cleanup function. The case where only the complex root token was given is covered by __siunitx_number_parse_loop_root_swap:NNwNN.

```

456 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_complex:N #1
457 {
458   \tl_set_eq:NN \l__siunitx_number_imaginary_tl #1
459   \tl_clear:N #1
460   \__siunitx_number_parse_loop_complex_cleanup:wN
461 }

```

If a sign is found, terminate the current number, store the sign as the first token of the second part and go back to do the dedicated first-token function.

```

462 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_sign:NNN #1#2#3
463 {
464   \__siunitx_number_parse_loop_main_store:NNN #1 #2 \c_true_bool
465   \tl_set:Nn \l__siunitx_number_flex_tl {#3}
466   \__siunitx_number_parse_loop_first:NNN
467   \l__siunitx_number_flex_tl \c_false_bool
468 }

```

A common auxiliary for the various non-digit token functions: tidy up the integer and decimal parts of a number. Here, the two flags are used to indicate if empty decimal and uncertainty parts should be included in the storage cycle.

```

469 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_store:NNN #1#2#3
470 {
471   \tl_if_empty:NT \l__siunitx_number_partial_tl
472   { \tl_set:Nn \l__siunitx_number_partial_tl { 0 } }
473   \tl_put_right:Nx #1
474   {
475     { \exp_not:V \l__siunitx_number_partial_tl }
476     \bool_if:NT #2 { { } }
477     \bool_if:NT #3 { { } }
478   }
479   \tl_clear:N \l__siunitx_number_partial_tl
480 }

```

After a decimal marker there has to be a digit if there wasn't one before it. That is handled by using a dedicated function, which checks for an empty integer part first then either simply hands off or looks for a digit.

```

481 \cs_new_protected:Npn \__siunitx_number_parse_loop_after_decimal:NNN #1#2#3
482 {
483   \tl_if_blank:fTF { \exp_after:wN \use_none:n #1 }
484   {
485     \quark_if_recursion_tail_stop_do:Nn #3
486     { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
487     \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#1}
488     {

```

```

489         \tl_put_right:Nn \l__siunitx_number_partial_tl {#3}
490         \__siunitx_number_parse_loop_main:NNNNN
491         #1 \c_false_bool \c_true_bool #2
492     }
493     { \__siunitx_number_parse_loop_break:wN }
494 }
495 {
496     \__siunitx_number_parse_loop_main:NNNNN
497     #1 \c_false_bool \c_true_bool #2 #3
498 }
499 }

```

Inside the brackets for an uncertainty the range of valid choices is very limited. Either the token is a digit, in which case there is a test to look for non-significant zeros, or it is a closing bracket. The latter is not valid for the very first token, which is handled using a switch (it's a simple enough difference).

```

500 \cs_new_protected:Npn \__siunitx_number_parse_loop_uncert:NNNNN #1#2#3#4#5
501 {
502     \quark_if_recursion_tail_stop_do:Nn #5
503     { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
504     \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#5}
505     {
506         \bool_lazy_or:nnTF
507         {#3} { ! \str_if_eq_p:nn {#5} { 0 } }
508         {
509             \tl_put_right:Nn \l__siunitx_number_partial_tl {#5}
510             \__siunitx_number_parse_loop_uncert:NNNNN
511             #1 \c_false_bool \c_true_bool #4
512         }
513         {
514             \__siunitx_number_parse_loop_uncert:NNNNN
515             #1 \c_false_bool \c_false_bool #4
516         }
517     }
518     {
519         \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#5}
520         {
521             \bool_if:NTF #2
522             { \__siunitx_number_parse_loop_break:wN }
523             {
524                 \__siunitx_number_parse_loop_main_store:NNN #1
525                 \c_false_bool \c_false_bool
526                 \__siunitx_number_parse_loop_after_uncert:NNN #1 #3
527             }
528         }
529         { \__siunitx_number_parse_loop_break:wN }
530     }
531 }

```

After a bracketed uncertainty there are only a very small number of valid choices. The number can end, there can be a complex root token or there can be a sign. The latter is only allowed if the part being parsed at the moment was the first part of the number. The case where there is no root symbol but there should have been is cleared up after the loop code, so at this stage there is no check.


```

532 \cs_new_protected:Npn \__siunitx_number_parse_loop_after_uncert:NNN #1#2#3
533 {
534   \quark_if_recursion_tail_stop:N #3
535   \tl_if_in:NnTF \l__siunitx_number_input_complex_tl {#3}
536   { \__siunitx_number_parse_loop_main_complex:N #1 }
537   {
538     \bool_if:NTF #2
539     {
540       \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#3}
541       {
542         \tl_set:Nn \l__siunitx_number_flex_tl {#3}
543         \__siunitx_number_parse_loop_first:NNN
544         \l__siunitx_number_flex_tl \c_false_bool
545       }
546       { \__siunitx_number_parse_loop_break:wN }
547     }
548     { \__siunitx_number_parse_loop_break:wN }
549   }
550 }

```

When the complex root symbol comes at the start of the number rather than at the end, the easiest approach is to shuffle it to the “normal” position. As the exponent has already been removed, this must be the last token of the input and any duplication will be picked up. The case where just a complex root token has to be covered: in that situation, there is an implicit 1 to store after which the loop stops.

```

551 \cs_new_protected:Npn \__siunitx_number_parse_loop_root_swap:NNwNN #1#2#3
552 \q_recursion_tail \q_recursion_stop
553 {
554   \tl_if_blank:nTF {#3}
555   {
556     \tl_set:Nx \l__siunitx_number_imaginary_tl
557     {
558       \exp_not:V #1
559       { 1 } { } { }
560     }
561     \tl_clear:N #1
562   }
563   {
564     \use:x
565     {
566       \tl_clear:N \exp_not:N #1
567       \tl_set:Nn \exp_not:N \l__siunitx_number_flex_tl { \exp_not:V #1 }
568     }
569     \__siunitx_number_parse_loop_first:NNN
570     \l__siunitx_number_flex_tl \c_false_bool
571     #3 #2 \q_recursion_tail \q_recursion_stop
572   }
573 }

```

Nothing is allowed after a complex root token: check and if there is kill the parsing.

```

574 \cs_new_protected:Npn \__siunitx_number_parse_loop_complex_cleanup:wN
575 #1 \q_recursion_tail \q_recursion_stop
576 {
577   \tl_if_blank:nF {#1}
578   { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }

```

579 }

Something is not right: remove all of the remaining tokens from the number and clear the storage areas as a signal for the next part of the code.

```
580 \cs_new_protected:Npn \__siunitx_number_parse_loop_break:wN
581   #1 \q_recursion_stop
582   {
583     \tl_clear:N \l__siunitx_number_imaginary_tl
584     \tl_clear:N \l__siunitx_number_flex_tl
585     \tl_clear:N \l__siunitx_number_real_tl
586   }
```

(End definition for __siunitx_number_parse_loop: and others.)

__siunitx_number_parse_sign:
__siunitx_number_parse_sign_aux:Nw

The first token of a number after a comparator could be a sign. A quick check is made and if found stored; if there is no sign then the internal format requires that + is used. For the number to be valid it has to be more than just a sign, so the next part of the chain is only called if that is the case.

```
587 \cs_new_protected:Npn \__siunitx_number_parse_sign:
588   {
589     \exp_after:wN \__siunitx_number_parse_sign_aux:Nw
590     \l__siunitx_number_arg_tl \q_stop
591   }
592 \cs_new_protected:Npn \__siunitx_number_parse_sign_aux:Nw #1#2 \q_stop
593   {
594     \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#1}
595     {
596       \tl_set:Nn \l__siunitx_number_arg_tl {#2}
597       \tl_set:Nn \l__siunitx_number_real_tl {#1}
598     }
599     { \tl_set:Nn \l__siunitx_number_real_tl { + } }
600     \tl_if_empty:NTF \l__siunitx_number_arg_tl
601     { \tl_clear:N \l__siunitx_number_real_tl }
602     { \__siunitx_number_parse_exponent: }
603   }
```

(End definition for __siunitx_number_parse_sign: and __siunitx_number_parse_sign_aux:Nw.)

1.4 Processing numbers

\l__siunitx_number_round_half_up_bool
\l__siunitx_number_round_min_tl
\l__siunitx_number_round_mode_tl
\l__siunitx_number_round_precision_int

```
604 \keys_define:nn { siunitx }
605   {
606     round-half .choice: ,
607     round-half / even .code:n =
608       { \bool_set_false:N \l__siunitx_number_round_half_up_bool } ,
609     round-half / up .code:n =
610       { \bool_set_true:N \l__siunitx_number_round_half_up_bool } ,
611     round-minimum .tl_set:N =
612       \l__siunitx_number_round_min_tl ,
613     round-mode .choice: ,
614     round-mode / figures .code:n =
615       { \tl_set:Nn \l__siunitx_number_round_mode_tl { figures } } ,
616     round-mode / none .code:n =
```

```

617     { \tl_set:Nn \__siunitx_number_round_mode_tl { none } } ,
618     round-mode / places .code:n =
619     { \tl_set:Nn \__siunitx_number_round_mode_tl { places } } ,
620     round-mode / uncertainty .code:n =
621     { \tl_set:Nn \__siunitx_number_round_mode_tl { uncertainty } } ,
622     round-precision .int_set:N =
623     \l__siunitx_number_round_precision_int ,
624   }
625 \bool_new:N \l__siunitx_number_round_half_up_bool
626 \tl_new:N \__siunitx_number_round_mode_tl

```

(End definition for \l__siunitx_number_round_half_up_bool and others.)

```

\__siunitx_number_round:
\__siunitx_number_round:N 627 \cs_new_protected:Npn \__siunitx_number_round:
\__siunitx_number_round_none:nNnnnNn 628 {
629   \__siunitx_number_round:N \l__siunitx_number_real_tl
630   \__siunitx_number_round:N \l__siunitx_number_imaginary_tl
631 }
632 \cs_new_protected:Npn \__siunitx_number_round:N #1
633 {
634   \tl_if_empty:NF #1
635   {
636     \tl_set:Nx #1
637     {
638       \cs:w
639       __siunitx_number_round_ \__siunitx_number_round_mode_tl :nNnnnNn
640       \exp_after:wN
641       \cs_end: #1
642     }
643   }
644 }
645 \cs_new:Npn \__siunitx_number_round_none:nNnnnNn #1#2#3#4#5#6#7
646 { \exp_not:n { {#1} #2 {#3} {#4} {#5} #6 {#7} } }

(End definition for \__siunitx_number_round:, \__siunitx_number_round:N, and \__siunitx_number_
round_none:nNnnnNn.)

```

__siunitx_number_round_figures:nNnnnNn

```

647 \cs_new:Npn \__siunitx_number_round_figures:nNnnnNn #1#2#3#4#5#6#7
648 {
649   \tl_if_blank:nTF {#5}
650   {
651     \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
652     { }
653     { { } + { 0 } { } { } + { 0 } }
654   }
655   { \exp_not:n { {#1} #2 {#3} {#4} {#5} #6 {#7} } }
656 }

```

(End definition for __siunitx_number_round_figures:nNnnnNn.)

__siunitx_number_round_places:nNnnnNn
__siunitx_number_round_places_decimal:nNnnNn
__siunitx_number_round_places_integer:nNnnNn

The first step when rounding to a fixed number of places is to establish if this is in the decimal or integer parts. The two require different calculations for how many digits to drop from the input.

```

657 \cs_new:Npn \__siunitx_number_round_places:nNnnNn #1#2#3#4#5#6#7
658 {
659   \tl_if_blank:nTF {#5}
660   {
661     \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
662     { \__siunitx_number_round_places_decimal:nNnnNn }
663     { \__siunitx_number_round_places_integer:nNnnNn }
664     {#1} #2 {#3} {#4} #6 {#7}
665   }
666   { \exp_not:n { {#1} #2 {#3} {#4} {#5} #6 {#7} } }
667 }
668 \cs_new:Npn \__siunitx_number_round_places_decimal:nNnnNn #1#2#3#4#5#6
669 { }
670 \cs_new:Npn \__siunitx_number_round_places_integer:nNnnNn #1#2#3#4#5#6
671 { }

```

(End definition for `__siunitx_number_round_places:nNnnNn`, `__siunitx_number_round_places_decimal:nNnnNn`, and `__siunitx_number_round_places_integer:nNnnNn`.)

`__siunitx_number_round_uncertainty:nNnnNn`

```

672 \cs_new:Npn \__siunitx_number_round_uncertainty:nNnnNn #1#2#3#4#5#6#7
673 {
674   \tl_if_blank:nTF {#5}
675   { \exp_not:n { {#1} #2 {#3} {#4} { } #6 {#7} } }
676   { }
677 }

```

(End definition for `__siunitx_number_round_uncertainty:nNnnNn`.)

1.5 Formatting parsed numbers

Keys producing tokens in the output.

```

\l__siunitx_number_bracket_negative_bool
\l__siunitx_number_bracket_close_tl
\l__siunitx_number_explicit_plus_bool
\l__siunitx_number_exponent_base_tl
\l__siunitx_number_exponent_product_tl
\l__siunitx_number_group_decimal_bool
\l__siunitx_number_group_integer_bool
\l__siunitx_number_group_minimum_int
\l__siunitx_number_group_separator_tl
\l__siunitx_number_negative_color_tl
\l__siunitx_number_bracket_open_tl
\l__siunitx_number_output_uncert_close_tl
\l__siunitx_number_output_complex_tl
\l__siunitx_number_output_decimal_tl
\l__siunitx_number_output_uncert_open_tl
\l__siunitx_number_uncert_separate_bool
\l__siunitx_number_tight_bool
\l__siunitx_number_unity_mantissa_bool
\l__siunitx_number_zero_exponent_bool

678 \keys_define:nn { siunitx }
679 {
680   bracket-negative .bool_set:N =
681     \l__siunitx_number_bracket_negative_bool ,
682   explicit-plus .bool_set:N =
683     \l__siunitx_number_explicit_plus_bool ,
684   exponent-base .tl_set:N =
685     \l__siunitx_number_exponent_base_tl ,
686   exponent-product .tl_set:N =
687     \l__siunitx_number_exponent_product_tl ,
688   group-digits .choice: ,
689   group-digits / all .code:n =
690     {
691       \bool_set_true:N \l__siunitx_number_group_decimal_bool
692       \bool_set_true:N \l__siunitx_number_group_integer_bool
693     } ,
694   group-digits / decimal .code:n =
695     {
696       \bool_set_true:N \l__siunitx_number_group_decimal_bool
697       \bool_set_false:N \l__siunitx_number_group_integer_bool
698     } ,
699   group-digits / integer .code:n =

```

```

700     {
701         \bool_set_false:N \l__siunitx_number_group_decimal_bool
702         \bool_set_true:N \l__siunitx_number_group_integer_bool
703     } ,
704     group-digits / none .code:n =
705     {
706         \bool_set_false:N \l__siunitx_number_group_decimal_bool
707         \bool_set_false:N \l__siunitx_number_group_integer_bool
708     } ,
709     group-digits .default:n = all ,
710     group-minimum-digits .int_set:N =
711         \l__siunitx_number_group_minimum_int ,
712     group-separator .tl_set:N =
713         \l__siunitx_number_group_separator_tl ,
714     negative-color .tl_set:N =
715         \l__siunitx_number_negative_color_tl ,
716     number-close-bracket .tl_set:N =
717         \l__siunitx_number_bracket_close_tl ,
718     number-open-bracket .tl_set:N =
719         \l__siunitx_number_bracket_open_tl ,
720     output-close-uncertainty .tl_set:N =
721         \l__siunitx_number_output_uncert_close_tl ,
722     output-complex-root .tl_set:N =
723         \l__siunitx_number_output_complex_tl ,
724     output-decimal-marker .tl_set:N =
725         \l__siunitx_number_output_decimal_tl ,
726     output-open-uncertainty .tl_set:N =
727         \l__siunitx_number_output_uncert_open_tl ,
728     separate-uncertainty .bool_set:N =
729         \l__siunitx_number_uncert_separate_bool ,
730     tight-spacing .bool_set:N =
731         \l__siunitx_number_tight_bool ,
732     unity-mantissa .bool_set:N =
733         \l__siunitx_number_unity_mantissa_bool ,
734     zero-exponent .bool_set:N =
735         \l__siunitx_number_zero_exponent_bool ,
736 }
737 \bool_new:N \l__siunitx_number_group_decimal_bool
738 \bool_new:N \l__siunitx_number_group_integer_bool

```

(End definition for \l__siunitx_number_bracket_negative_bool and others.)

__siunitx_number_format:

```

739 \cs_new_protected:Npn \__siunitx_number_format:
740 {
741     \tl_set:Nx \l__siunitx_number_formatted_tl
742     {
743         \tl_if_empty:NTF \l__siunitx_number_real_tl
744         {
745             \tl_if_empty:NF \l__siunitx_number_imaginary_tl
746             { \__siunitx_number_format:N \l__siunitx_number_imaginary_tl }
747         }
748         {
749             \tl_if_empty:NTF \l__siunitx_number_imaginary_tl

```

```

750             { \__siunitx_number_format:N \l__siunitx_number_real_tl }
751             { ??? }
752         }
753     }
754 }

```

(End definition for __siunitx_number_format:.)

```

\__siunitx_number_format:N
  \__siunitx_number_format:nNnnNn
  \__siunitx_number_format_comparator:n
  \__siunitx_number_format_sign:N
  \__siunitx_number_format_sign_aux:N
  \__siunitx_number_format_sign_color:w
  \__siunitx_number_format_sign_brackets:w
  \__siunitx_number_format_integer:nnn
  \__siunitx_number_format_decimal:n
  \__siunitx_number_format_decimal:f
  \__siunitx_number_format_digits:nn
  \__siunitx_number_format_integer_aux:n
  \__siunitx_number_format_integer_aux_0:n
  \__siunitx_number_format_integer_aux_1:n
  \__siunitx_number_format_integer_aux_2:n
  \__siunitx_number_format_decimal_aux:n
  \__siunitx_number_format_decimal_loop:NNNN
  \__siunitx_number_format_integer_first:nnNN
  \__siunitx_number_format_integer_loop:NNNN
  \__siunitx_number_format_uncertainty:nn
  \__siunitx_number_format_uncertainty_unaligned:
  \__siunitx_number_format_uncertainty_aux:nn
  \__siunitx_number_format_uncertainty_aux:fn
  \__siunitx_number_format_uncertainty:nw
  \__siunitx_number_format_uncertainty:fnw
  \__siunitx_number_format_uncertainty:nw
  \__siunitx_number_format_exponent:Nnn
  \__siunitx_number_format_end:

```

The approach to formatting a single number is to split into the constituent parts. All of the parts are assembled including inserting tabular alignment markers (which may be empty) for each separate unit.

```

755 \cs_new:Npn \__siunitx_number_format:N #1
756 { \exp_after:wN \__siunitx_number_format:nNnnNn #1 }
757 \cs_new:Npn \__siunitx_number_format:nNnnNn #1#2#3#4#5#6#7
758 {
759   \__siunitx_number_format_comparator:n {#1}
760   \__siunitx_number_format_sign:N #2
761   \__siunitx_number_format_integer:nnn {#3} {#4} {#7}
762   \__siunitx_number_format_decimal:n {#4}
763   \__siunitx_number_format_uncertainty:nn {#5} {#4}
764   \__siunitx_number_format_exponent:Nnn #6 {#7} { #3 . #4 }
765   \__siunitx_number_format_end:
766 }

```

To get the spacing correct this needs to be an ordinary math character.

```

767 \cs_new:Npn \__siunitx_number_format_comparator:n #1
768 {
769   \tl_if_blank:nF {#1}
770   { \exp_not:n { \mathord {#1} } }
771   \exp_not:V \l__siunitx_number_tab_tl
772 }

```

Formatting signs has to deal with some additional formatting requirements for negative numbers. Both making such numbers a fixed color and bracketing them needs some rearrangement of the order of tokens, which is set up in the main formatting macro by the dedicated do-nothing end function.

```

773 \cs_new:Npn \__siunitx_number_format_sign:N #1
774 {
775   \str_if_eq:nnTF {#1} { + }
776   {
777     \bool_if:NT \l__siunitx_number_explicit_plus_bool
778     { \__siunitx_number_format_sign_aux:N #1 }
779   }
780   {
781     \str_if_eq:nnTF {#1} { - }
782     {
783       \tl_if_empty:NF \l__siunitx_number_negative_color_tl
784       { \__siunitx_number_format_sign_color:w }
785       \bool_if:NTF \l__siunitx_number_bracket_negative_bool
786       { \__siunitx_number_format_sign_brackets:w }
787       { \__siunitx_number_format_sign_aux:N #1 }
788     }
789     { \__siunitx_number_format_sign_aux:N #1 }
790   }
791 }

```

```

792 \cs_new:Npn \__siunitx_number_format_sign_aux:N #1
793 {
794   \bool_if:NTF \l__siunitx_number_tight_bool
795   { \exp_not:n { \mathord {#1} } }
796   { \exp_not:n {#1} }
797 }
798 \cs_new:Npn
799   \__siunitx_number_format_sign_color:w #1 \__siunitx_number_format_end:
800 {
801   \exp_not:N \textcolor { \exp_not:V \l__siunitx_number_negative_color_tl }
802   {
803     #1
804     \__siunitx_number_format_end:
805   }
806 }
807 \cs_new:Npn
808   \__siunitx_number_format_sign_brackets:w #1 \__siunitx_number_format_end:
809 {
810   \exp_not:V \l__siunitx_number_bracket_open_tl
811   #1
812   \exp_not:V \l__siunitx_number_bracket_close_tl
813   \__siunitx_number_format_end:
814 }

```

Digit formatting leads off with separate functions to allow for a few “up front” items before using a common set of tests for some common cases. The code then splits again as the two types of grouping need different strategies.

```

815 \cs_new:Npn \__siunitx_number_format_integer:nnn #1#2#3
816 {
817   \bool_lazy_all:nF
818   {
819     { \str_if_eq_p:nn {#1} { 1 } }
820     { \tl_if_blank_p:n {#2} }
821     { ! \str_if_eq_p:nn {#3} { 0 } }
822     { ! \l__siunitx_number_unity_mantissa_bool }
823   }
824   { \__siunitx_number_format_digits:nn { integer } {#1} }
825 }
826 \cs_new:Npn \__siunitx_number_format_decimal:n #1
827 {
828   \exp_not:V \l__siunitx_number_tab_tl
829   \tl_if_blank:nF {#1}
830   { \exp_not:V \l__siunitx_number_output_decimal_tl }
831   \exp_not:V \l__siunitx_number_tab_tl
832   \__siunitx_number_format_digits:nn { decimal } {#1}
833 }
834 \cs_generate_variant:Nn \__siunitx_number_format_decimal:n { f }
835 \cs_new:Npn \__siunitx_number_format_digits:nn #1#2
836 {
837   \bool_if:cTF { l__siunitx_number_group_ #1 _ bool }
838   {
839     \int_compare:nNnTF
840     { \tl_count:n {#2} } < \l__siunitx_number_group_minimum_int
841     { \exp_not:n {#2} }

```

```

842         { \use:c { __siunitx_number_format_ #1 _aux:n } {#2} }
843     }
844     { \exp_not:n {#2} }
845 }

```

For integers, we need to know how many digits there are to allow for the correct insertion of separators. That is done using a two-part set up such that there is no separator on the first pass.

```

846 \cs_new:Npn \__siunitx_number_format_integer_aux:n #1
847 {
848     \use:c
849     {
850         __siunitx_number_format_integer_aux_
851         \int_eval:n { \int_mod:nn { \tl_count:n {#1} } { 3 } }
852         :n
853     } {#1}
854 }
855 \cs_new:cpn { __siunitx_number_format_integer_aux_0:n } #1
856 { \__siunitx_number_format_integer_first:nnNN #1 \q_nil }
857 \cs_new:cpn { __siunitx_number_format_integer_aux_1:n } #1
858 { \__siunitx_number_format_integer_first:nnNN { } { } #1 \q_nil }
859 \cs_new:cpn { __siunitx_number_format_integer_aux_2:n } #1
860 { \__siunitx_number_format_integer_first:nnNN { } #1 \q_nil }
861 \cs_new:Npn \__siunitx_number_format_integer_first:nnNN #1#2#3#4
862 {
863     \exp_not:n {#1#2#3}
864     \quark_if_nil:NF #4
865     { \__siunitx_number_format_integer_loop:NNNN #4 }
866 }
867 \cs_new:Npn \__siunitx_number_format_integer_loop:NNNN #1#2#3#4
868 {
869     \exp_not:V \l__siunitx_number_group_separator_tl
870     \exp_not:n {#1#2#3}
871     \quark_if_nil:NF #4
872     { \__siunitx_number_format_integer_loop:NNNN #4 }
873 }

```

For decimals, no need to do any counting, just loop using enough markers to find the end of the list. By passing the decimal marker, it is possible not to have to use a check on the content of the rest of the number. The `\use_none:n(n)` mop up the remaining `\q_nil` tokens.

```

874 \cs_new:Npn \__siunitx_number_format_decimal_aux:n #1
875 {
876     \__siunitx_number_format_decimal_loop:NNNN \c_empty_tl
877     #1 \q_nil \q_nil \q_nil
878 }
879 \cs_new:Npn \__siunitx_number_format_decimal_loop:NNNN #1#2#3#4
880 {
881     \quark_if_nil:NF #2
882     {
883         \exp_not:V #1
884         \exp_not:n {#2}
885         \quark_if_nil:NTF #3
886         { \use_none:n }

```



```

887     {
888         \exp_not:n {#3}
889         \quark_if_nil:NTF #4
890         { \use_none:nn }
891         {
892             \exp_not:n {#4}
893             \__siunitx_number_format_decimal_loop:NNNN
894             \l__siunitx_number_group_separator_tl
895         }
896     }
897 }
898 }

```

Uncertainties which are directly attached are easy to deal with. For those that are separated, the first step is to find if they are entirely contained within the decimal part, and to pad if they are. For the case where the boundary is crossed to the integer part, the correct number of digit tokens need to be removed from the start of the uncertainty and the split result sent to the appropriate auxiliaries.

```

899 \cs_new:Npn \__siunitx_number_format_uncertainty:nn #1#2
900 {
901     \tl_if_blank:nTF {#1}
902     { \__siunitx_number_format_uncertainty_unaligned: }
903     {
904         \bool_if:NTF \l__siunitx_number_uncert_separate_bool
905         {
906             \exp_not:V \l__siunitx_number_tab_tl
907             \__siunitx_number_format_sign_aux:N \pm
908             \exp_not:V \l__siunitx_number_tab_tl
909             \__siunitx_number_format_uncertainty_aux:fn
910             { \int_eval:n { \tl_count:n {#1} - \tl_count:n {#2} } }
911             {#1}
912         }
913         {
914             \exp_not:V \l__siunitx_number_output_uncert_open_tl
915             \exp_not:n {#1}
916             \exp_not:V \l__siunitx_number_output_uncert_close_tl
917             \__siunitx_number_format_uncertainty_unaligned:
918         }
919     }
920 }
921 \cs_new:Npn \__siunitx_number_format_uncertainty_unaligned:
922 {
923     \exp_not:V \l__siunitx_number_tab_tl
924     \exp_not:V \l__siunitx_number_tab_tl
925     \exp_not:V \l__siunitx_number_tab_tl
926     \exp_not:V \l__siunitx_number_tab_tl
927 }
928 \cs_new:Npn \__siunitx_number_format_uncertainty_aux:nn #1#2
929 {
930     \int_compare:nNnTF {#1} > 0
931     {
932         \__siunitx_number_format_uncertainty_aux:fnw
933         { \int_eval:n { #1 - 1 } }
934         { }

```

```

935     #2 \q_nil
936   }
937   {
938     0
939     \__siunitx_number_format_decimal:f
940     {
941       \prg_replicate:nn { \int_abs:n {#1} } { 0 }
942       #2
943     }
944   }
945 }
946 \cs_generate_variant:Nn \__siunitx_number_format_uncertainty_aux:nn { f }
947 \cs_new:Npn \__siunitx_number_format_uncertainty_aux:nnw #1#2#3
948 {
949   \quark_if_nil:NF #3
950   {
951     \int_compare:nNnTF {#1} = 0
952     { \__siunitx_number_format_uncertainty_aux:nw {#2#3} }
953     {
954       \__siunitx_number_format_uncertainty_aux:fnw
955       { \int_eval:n { #1 - 1 } }
956       {#2#3}
957     }
958   }
959 }
960 \cs_generate_variant:Nn \__siunitx_number_format_uncertainty_aux:nnw { f }
961 \cs_new:Npn \__siunitx_number_format_uncertainty_aux:nw #1#2 \q_nil
962 {
963   \__siunitx_number_format_digits:nn { integer } {#1}
964   \__siunitx_number_format_decimal:n {#2}
965 }

```

Setting the exponent part requires some information about the mantissa: was it there or not. This means that whilst only the sign and value for the exponent are typeset here, there is a need to also have access to the combined mantissa part (with a decimal marker). The rest of the work is about picking up the various options and getting the combinations right. For signs, the auxiliary from the main sign routine can be used, but not the main function: negative exponents don't have special handling.

```

966 \cs_new:Npn \__siunitx_number_format_exponent:Nnn #1#2#3
967 {
968   \exp_not:V \l__siunitx_number_tab_tl
969   \bool_lazy_or:nnTF
970   { \l__siunitx_number_zero_exponent_bool }
971   { ! \str_if_eq_p:nn {#2} { 0 } }
972   {
973     \bool_lazy_and:nnTF
974     { \str_if_eq_p:nn {#3} { 1. } }
975     { ! \l__siunitx_number_unity_mantissa_bool }
976     { \exp_not:V \l__siunitx_number_tab_tl }
977     {
978       \bool_if:NTF \l__siunitx_number_tight_bool
979       {
980         \exp_not:N \mathord
981         { \exp_not:V \l__siunitx_number_exponent_product_tl }

```

```

982     }
983     { \exp_not:V \l__siunitx_number_exponent_product_tl }
984     \exp_not:V \l__siunitx_number_tab_tl
985   }
986   \exp_not:V \l__siunitx_number_exponent_base_tl
987   ~
988   {
989     \bool_lazy_or:nnT
990     { \l__siunitx_number_explicit_plus_bool }
991     { ! \str_if_eq_p:nn {#1} { + } }
992     { \l__siunitx_number_format_sign_aux:N #1 }
993     \l__siunitx_number_format_digits:nn { integer } {#2}
994   }
995 }
996 { \exp_not:V \l__siunitx_number_tab_tl }
997 }

```

A do-nothing marker used to allow shuffling of the output and so expandable operations for formatting.

```
998 \cs_new:Npn \l__siunitx_number_format_end: { }
```

(End definition for \l__siunitx_number_format:N and others.)

1.6 Miscellaneous tools

`\l__siunitx_number_valid_tl` The list of valid tokens.

```
999 \tl_new:N \l__siunitx_number_valid_tl
```

(End definition for \l__siunitx_number_valid_tl.)

`\siunitx_if_number:nTF` Test if an entire number is valid: this means parsing the number but not returning anything.

```

1000 \prg_new_protected_conditional:Npnn \siunitx_if_number:n #1
1001 { T , F , TF }
1002 {
1003   \group_begin:
1004     \bool_set_true:N \l__siunitx_number_validate_bool
1005     \l__siunitx_number_parse:n {#1}
1006     \bool_lazy_and:nnTF
1007     { \tl_if_empty_p:N \l__siunitx_number_real_tl }
1008     { \tl_if_empty_p:N \l__siunitx_number_imaginary_tl }
1009     {
1010       \group_end:
1011       \prg_return_false:
1012     }
1013     {
1014       \group_end:
1015       \prg_return_true:
1016     }
1017 }

```

(End definition for \siunitx_if_number:nTF. This function is documented on page 7.)

`\siunitx_if_number_token:N \underline{TF}` A simple conditional to answer the question of whether a specific token is possibly valid in a number.

```

1018 \prg_new_protected_conditional:Npnn \siunitx_number_token:N #1
1019 { T , F , TF }
1020 {
1021   \tl_set:Nx \l__siunitx_number_valid_tl
1022   {
1023     \exp_not:V \l__siunitx_number_input_uncert_close_tl
1024     \exp_not:V \l__siunitx_number_input_complex_tl
1025     \exp_not:V \l__siunitx_number_input_comparator_tl
1026     \exp_not:V \l__siunitx_number_input_decimal_tl
1027     \exp_not:V \l__siunitx_number_input_digit_tl
1028     \exp_not:V \l__siunitx_number_input_exponent_tl
1029     \exp_not:V \l__siunitx_number_input_ignore_tl
1030     \exp_not:V \l__siunitx_number_input_uncert_open_tl
1031     \exp_not:V \l__siunitx_number_input_sign_tl
1032     \exp_not:V \l__siunitx_number_input_uncert_sign_tl
1033   }
1034   \tl_if_in:VnTF \l__siunitx_number_valid_tl {#1}
1035   { \prg_return_true: }
1036   { \prg_return_false: }
1037 }

```

(End definition for `\siunitx` if number token:NTF. This function is documented on page 7.)

1.7 Messages

```

1038 \msg_new:nnnn { siunitx } { number / invalid-input }
1039 { Invalid-number~'#1'. }
1040 {
1041   The-input~'#1'~could-not-be-parsed-as-a-number-following-the-
1042   format-defined-in-module-documentation.
1043 }

```

1.8 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.

```

1044 \keys_set:nn { siunitx }
1045 {
1046     bracket-negative          = false
1047     evaluate-expression       = false
1048     explicit-plus             = false
1049     exponent-base             = 10
1050     exponent-product          = \times
1051     expression                = #1
1052     group-digits              = all
1053     group-minimum-digits     = 4
1054     group-separator           = \,
1055     input-close-uncertainty   = )
1056     input-complex-roots       = ij
1057     input-comparators         = { <=>\approx\ge\geq\gg\le\leq\ll\sim }
1058     input-decimal-markers     = { ., }
1059     input-digits              = 0123456789

```

```

1060     input-exponent-markers = dDeE ,
1061     input-ignore           = \, ,
1062     input-open-uncertainty = ( , % )
1063     input-signs            = +-\mp\pm ,
1064     input-uncertainty-signs = \pm ,
1065     negative-color         = , % (
1066     number-close-bracket   = ) ,
1067     number-open-bracket    = ( , % )
1068     output-close-uncertainty = ) ,
1069     output-complex-root    = \mathrm { i } ,
1070     output-decimal-marker  = . ,
1071     output-open-uncertainty = ( , % )
1072     round-half             = up ,
1073     round-minimum          = 0 ,
1074     round-mode              = none ,
1075     round-precision        = 2 ,
1076     separate-uncertainty   = false ,
1077     tight-spacing          = false ,
1078     unity-mantissa         = false ,
1079     zero-exponent          = false ,
1080 }
1081 \</package>

```

Part IV

siunitx-print – Printing material with font control

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

The printing routines assume that normal L^AT_EX 2_ε font selection commands are available, in particular `\bfsries`, `\mathrm`, `\mathversion`, `\mdseries`, `\rmfamily` and `\upshape`. It also requires the standard L^AT_EX 2_ε kernel commands `\ensuremath`, `\textsubscript` and `\textsuperscript` for printing in text mode. The following packages are also required to provide the functionality detailed.

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

`\siunitx_print:nn`
`\siunitx_print:nV`

`\siunitx_print:nn {<type>}{<material>}`

Prints the *<material>* according to the prevailing settings for the submodule as applicable to the *<type>* of content: the latter should be either **number** or **unit**. The *<material>* should comprise normal L^AT_EX mark-up for numbers or units. In particular, units will typically use `\mathrm` to indicate material to be printed in the current upright roman font, and `^` and `_` will typically be used to indicate super- and subscripts, respectively. These elements will be correctly handled when printing for example using `\mathsf` in math mode, or using only text fonts.

`\l_siunitx_print_series_prop`

This properly list contains mappings from text mode font weights to math mode font versions. The standard settings here cover

- `m` Standard (mid) weight, mapping to math version **normal**.
- `bx` Bold exanded weight, mapping to math version **bold**.
- `lt` Light weight, mapping to math version **light**.
- `l` Light weight, mapping to math version **light**.

0.1 Key–value options

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

<hr/> <hr/> color	color = $\langle color \rangle$
	Color to apply to printed output: the latter should be a named color defined for use with <code>\textcolor</code> . The standard setting is empty (no color).
<hr/> <hr/> mode	mode = $\langle choice \rangle$
	Selects which mode (math or text) the output is printed in: a choice from the options <code>match</code> , <code>math</code> or <code>text</code> . The option <code>match</code> matches the mode prevailing at the point <code>\siunitx_print:nn</code> is called. The <code>math</code> and <code>text</code> options choose the relevant T _E X mode for printing. The standard setting is <code>math</code> .
<hr/> <hr/> number-color	number-color = $\langle color \rangle$
	Color to apply to numbers in output: the latter should be a named color defined for use with <code>\textcolor</code> . The standard setting is empty (no color).
<hr/> <hr/> number-mode	number-mode = $\langle choice \rangle$
	Selects which mode (math or text) the numbers are printed in: a choice from the options <code>match</code> , <code>math</code> or <code>text</code> . The option <code>match</code> matches the mode prevailing at the point <code>\siunitx_print:nn</code> is called. The <code>math</code> and <code>text</code> options choose the relevant T _E X mode for printing. The standard setting is <code>math</code> .
<hr/> <hr/> propagate-math-font	propagate-math-font = true false
	Switch to determine if the currently-active math font is applied within printed output. This is relevant only when <code>\siunitx_print:nn</code> is called from within math mode: in text mode there is not active math font. When not active, math mode material will be typeset using standard math mode fonts without any changes being made to the supplied argument. The standard setting is <code>false</code> .
<hr/> <hr/> reset-math-version	reset-math-version = true false
	Switch to determine whether the active <code>\mathversion</code> is reset to <code>normal</code> when printing in math mode. Note that math version is typically used to select <code>\boldmath</code> , though it is also be used by <i>e.g.</i> <code>sansmath</code> . The standard setting is <code>true</code> .
<hr/> <hr/> reset-text-family	reset-text-family = true false
	Switch to determine whether the active math family is reset to <code>\rmfamily</code> when printing in text mode. The standard setting is <code>true</code> .
<hr/> <hr/> reset-text-series	reset-text-series = true false
	Switch to determine whether the active math series is reset to <code>\mdseries</code> when printing in text mode. The standard setting is <code>true</code> .
<hr/> <hr/> reset-text-shape	reset-text-shape = true false
	Switch to determine whether the active math shape is reset to <code>\upshape</code> when printing in text mode. The standard setting is <code>true</code> .

<hr/> <hr/> text-family-to-math	text-family-to-math = true false
	Switch to determine if the family of the current text font should be applied (where possible) to printing in math mode. The standard setting is false .
<hr/> <hr/> text-weight-to-math	text-weight-to-math = true false
	Switch to determine if the weight of the current text font should be applied (where possible) to printing in math mode. This is achieved by setting the <code>\mathversion</code> , and so will override <code>reset-math-version</code> . The mappings between text and math weight are stored in <code>\l_siunitx_print_series_prop</code> . The standard setting is false .
<hr/> <hr/> unit-color	unit-color = $\langle color \rangle$
	Color to apply to units in output: the latter should be a named color defined for use with <code>\textcolor</code> . The standard setting is empty (no color).
<hr/> <hr/> unit-mode	unit-mode = $\langle choice \rangle$
	Selects which mode (math or text) units are printed in: a choice from the options match , math or text . The option match matches the mode prevailing at the point <code>\siunitx_print:nn</code> is called. The math and text options choose the relevant T _E X mode for printing. The standard setting is math .

1 siunitx-print implementation

Start the DocStrip guards.

```
1  $\langle *package \rangle$ 
```

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2  $\langle @@=siunitx_print \rangle$ 
```

1.1 Initial set up

The printing routines depend on `amstext` for text mode working.

```
3 \RequirePackage { amstext }
```

Color support is always required.

```
4 \RequirePackage { color }
```

For a sensible `\textminus` we load `textcomp` if `fontspec` is not in use.

```
5 \AtBeginDocument
```

```
6 {
```

```
7     \ifpackageloaded { fontspec }
```

```
8     { }
```

```
9     { \RequirePackage { textcomp } }
```

```
10 }
```

`\l__siunitx_print_tmp_box` Scratch space.

```
11 \box_new:N \l__siunitx_print_tmp_box
```

(End definition for `\l__siunitx_print_tmp_box`.)


```

\document
\__siunitx_print_store_fam:n
\c__siunitx_print_mathrm_int
\c__siunitx_print_mathsf_int
\c__siunitx_print_mathtt_int
12 \tl_put_right:Nn \document
13 {
14   \__siunitx_print_store_fam:n { rm }
15   \__siunitx_print_store_fam:n { sf }
16   \__siunitx_print_store_fam:n { tt }
17   \ignorespaces
18 }
19 \cs_new_protected:Npn \__siunitx_print_store_fam:n #1
20 {
21   \group_begin:
22     \hbox_set:Nn \l__siunitx_print_tmp_box
23     {
24       \ensuremath
25       {
26         \use:c { math #1 }
27         { \int_const:cn { c__siunitx_print_math #1 _int } { \fam } }
28       }
29     }
30   \group_end:
31 }

```

(End definition for `\document` and others. This function is documented on page ??.)

1.2 Printing routines

```

\l_siunitx_print_number_color_tl
\l_siunitx_print_number_mode_tl
\l_siunitx_print_unit_color_tl
\l_siunitx_print_unit_mode_tl
\l_siunitx_print_math_font_bool
\l_siunitx_print_math_version_bool
\l_siunitx_print_math_family_bool
\l_siunitx_print_math_weight_bool
\l_siunitx_print_text_family_tl
\l_siunitx_print_text_series_tl
\l_siunitx_print_text_shape_tl
238 \tl_new:N \l__siunitx_print_number_mode_tl
239 \tl_new:N \l__siunitx_print_unit_mode_tl
240 \keys_define:nn { siunitx }
241 {
242   color .meta:n =
243     { number-color = #1 , unit-color = #1 } ,
244   mode .meta:n =
245     { number-mode = #1 , unit-mode = #1 } ,
246   number-color .tl_set:N =
247     \l__siunitx_print_number_color_tl ,
248   number-mode .choices:nn =
249     { match , math , text }
250     {
251       \tl_set_eq:NN
252         \l__siunitx_print_number_mode_tl \l_keys_choice_tl
253     } ,
254   propagate-math-font .bool_set:N =
255     \l_siunitx_print_math_font_bool ,
256   reset-math-version .bool_set:N =
257     \l_siunitx_print_math_version_bool ,
258   reset-text-family .bool_set:N =
259     \l_siunitx_print_text_family_bool ,

```

```

54     reset-text-series .bool_set:N =
55         \l__siunitx_print_text_series_bool ,
56     reset-text-shape .bool_set:N =
57         \l__siunitx_print_text_shape_bool ,
58     text-family-to-math .bool_set:N =
59         \l__siunitx_print_math_family_bool ,
60     text-weight-to-math .bool_set:N =
61         \l__siunitx_print_math_weight_bool ,
62     unit-color .tl_set:N =
63         \l__siunitx_print_unit_color_tl ,
64     unit-mode .choices:nn =
65         { match , math , text }
66         {
67             \tl_set_eq:NN
68             \l__siunitx_print_unit_mode_tl \l_keys_choice_tl
69         }
70 }

```

(End definition for `\l__siunitx_print_number_color_tl` and others.)

`\siunitx_print:nn` The main printing function doesn't actually need to do very much: just set the color and
`\siunitx_print:nV` select the correct sub-function.

```

71 \cs_new_protected:Npn \siunitx_print:nn #1#2
72 {
73     \tl_if_empty:cTF { l__siunitx_print_ #1 _color_tl }
74     { \use:n }
75     { \exp_args:Nv \textcolor { l__siunitx_print_ #1 _color_tl } }
76     {
77         \use:c
78         {
79             __siunitx_print_
80             \tl_use:c { l__siunitx_print_ #1 _mode_tl } :n
81         }
82         {#2}
83     }
84 }
85 \cs_generate_variant:Nn \siunitx_print:nn { nV }

```

(End definition for `\siunitx_print:nn`. This function is documented on page 35.)

`__siunitx_print_match:n` When the *output* mode should match the input, a simple selection of route can be made.

```

86 \cs_new_protected:Npn \__siunitx_print_match:n #1
87 {
88     \mode_if_math:TF
89     { \__siunitx_print_math:n {#1} }
90     { \__siunitx_print_text:n {#1} }
91 }

```

(End definition for `__siunitx_print_match:n`.)

`\l_siunitx_print_series_prop` Mapping data to relate text series to math version.

```

92 \prop_new:N \l_siunitx_print_series_prop
93 \prop_put:Nnn \l_siunitx_print_series_prop { m } { normal }
94 \prop_put:Nnn \l_siunitx_print_series_prop { bx } { bold }
95 \prop_put:Nnn \l_siunitx_print_series_prop { l } { light }
96 \prop_put:Nnn \l_siunitx_print_series_prop { lt } { light }

```

(End definition for `\l_siunitx_print_series_prop`. This variable is documented on page 35.)

```

\__siunitx_print_math:n The first step in setting in math mode is to check on the math version. The starting
\__siunitx_print_math_version:nn point is the question of whether text series needs to propagate to math mode: if so, check
\__siunitx_print_math_version:Vn on the mapping, otherwise check on the current math version.
\__siunitx_print_math_auxi:n 97 \cs_new_protected:Npn \__siunitx_print_math:n #1
\__siunitx_print_math_auxii:n 98 {
\__siunitx_print_math_auxiii:n 99 \bool_if:NTF \l_siunitx_print_math_weight_bool
\__siunitx_print_math_auxiv:n 100 {
\__siunitx_print_math_auxv:n 101 \prop_get:NVNTF \l_siunitx_print_series_prop
\__siunitx_print_math_aux:Nn 102 \f@series \l__siunitx_print_tmp_tl
\__siunitx_print_math_aux:cn 103 { \__siunitx_print_math_version:Vn \l__siunitx_print_tmp_tl {#1} }
\__siunitx_print_math_sub:n 104 { \__siunitx_print_math_auxi:n {#1} }
\__siunitx_print_math_super:n 105 }
\__siunitx_print_math_script:n 106 { \__siunitx_print_math_auxi:n {#1} }
\__siunitx_print_math_text:n 107 }
108 \cs_new_protected:Npn \__siunitx_print_math_auxi:n #1
109 {
110 \bool_if:NTF \l_siunitx_print_math_version_bool
111 { \__siunitx_print_math_version:nn { normal } {#1} }
112 { \__siunitx_print_math_auxii:n {#1} }
113 }

```

Any setting which changes the math version can only be set from text mode (as it applies at the level of a formula). As such, the first test is to see if that needs to be checked if the math version has to be set: if so, switch to text mode, sort it out and switch back. That of course means that in such cases, line breaking will not be possible.

```

114 \cs_new_protected:Npn \__siunitx_print_math_version:nn #1#2
115 {
116 \str_if_eq:x:nnTF { \math@version } { #1 }
117 { \__siunitx_print_math_auxii:n {#2} }
118 {
119 \mode_if_math:TF
120 { \text }
121 { \use:n }
122 {
123 \mathversion {#1}
124 \__siunitx_print_math_auxii:n {#2}
125 }
126 }
127 }
128 \cs_generate_variant:Nn \__siunitx_print_math_version:nn { V }

```

At this point, force math mode then start dealing with setting math font based on text family. If the text family is roman, life is slightly different to if it is sanserif or monospaced. In all cases, the outcomes can be handled using the same routines as for normal math mode treatment.

```

129 \cs_new_protected:Npn \__siunitx_print_math_auxii:n #1
130 { \ensuremath { \__siunitx_print_math_auxiii:n {#1} } }
131 \cs_new_protected:Npn \__siunitx_print_math_auxiii:n #1
132 {
133 \bool_if:NTF \l_siunitx_print_math_family_bool
134 {
135 \str_case:x:nnF { \f@family }

```

```

136     {
137       { \rmdefault } { \_siunitx_print_math_auxv:n }
138       { \sfdefault } { \_siunitx_print_math_aux:Nn \mathsf }
139       { \ttdefault } { \_siunitx_print_math_aux:Nn \mathtt }
140     }
141     { \_siunitx_print_math_auxiv:n }
142   }
143   { \_siunitx_print_math_auxiv:n }
144   {#1}
145 }

```

Now we deal with the font selection in math mode. There are two possible cases. First, we are retaining the current math font, and the active one is `\mathsf` or `\mathtt`: that needs to be applied to the argument. Alternatively, if the current font is not retained, ensure that normal math mode rules are active. The parts here are split up to allow reuse when picking up the text family.

```

146 \cs_new_protected:Npn \_siunitx_print_math_auxiv:n #1
147 {
148   \bool_if:NTF \l__siunitx_print_math_font_bool
149   {
150     \int_case:nnF \fam
151     {
152       \c__siunitx_print_mathsf_int { \_siunitx_print_math_aux:Nn \mathsf }
153       \c__siunitx_print_mathtt_int { \_siunitx_print_math_aux:Nn \mathtt }
154     }
155     { \use:n }
156   }
157   { \_siunitx_print_math_auxv:n }
158   {#1}
159 }
160 \cs_new_protected:Npn \_siunitx_print_math_auxv:n #1
161 {
162   \bool_lazy_or:nnTF
163   { \int_compare_p:nNn \fam = { -1 } }
164   { \int_compare_p:nNn \fam = \c__siunitx_print_mathrm_int }
165   { \use:n }
166   { \mathrm }
167   {#1}
168 }

```

Search-and-replace fun: deal with any `\mathrm` in the argument and also inside sub/superscripts.

```

169 \cs_new_protected:Npx \_siunitx_print_math_aux:Nn #1#2
170 {
171   \group_begin:
172   \tl_set:Nn \exp_not:N \l__siunitx_print_tmp_tl {#2}
173   \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
174   { \exp_not:N \mathrm } { \exp_not:N \use:n }
175   \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
176   { \char_generate:nn { \_ } { 8 } }
177   { \exp_not:N \_siunitx_print_math_sub:n }
178   \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
179   { ^ }
180   { \exp_not:N \_siunitx_print_math_super:n }

```

```

181     #1 { \exp_not:N \tl_use:N \exp_not:N \l__siunitx_print_tmp_tl }
182   \group_end:
183 }
184 \cs_generate_variant:Nn \__siunitx_print_math_aux:Nn { c }
185 \cs_new_protected:Npx \__siunitx_print_math_sub:n #1
186 {
187   \char_generate:nn { '\_ } { 8 }
188   { \exp_not:N \__siunitx_print_math_script:n {#1} }
189 }
190 \cs_new_protected:Npn \__siunitx_print_math_super:n #1
191 { ~ { \__siunitx_print_math_script:n {#1} } }
192 \cs_new_protected:Npn \__siunitx_print_math_script:n #1
193 {
194   \group_begin:
195   \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
196   \tl_replace_all:Nnn \l__siunitx_print_tmp_tl { \mathrm } { \use:n }
197   \tl_use:N \l__siunitx_print_tmp_tl
198   \group_end:
199 }

```

To match the text mode font, there is a simple look-up of the current font family. Luckily, mappings to math mode equivalents are easy.

200

(End definition for __siunitx_print_math:n and others.)

<pre> __siunitx_print_text:n _siunitx_print_text_replace:n _siunitx_print_text_replace:N _siunitx_print_text_replace:NNn __siunitx_print_text_sub:n _siunitx_print_text_super:n _siunitx_print_text_scripts:NnN _siunitx_print_text_scripts: _siunitx_print_text_scripts_one:NnN _siunitx_print_text_scripts_two:NnNn _siunitx_print_text_scripts_two:nn _siunitx_print_text_scripts_two:n </pre>	<pre> Typesetting in text mode is easy in font control terms but more tricky in the manipulation of the input. The easy part comes first. 201 \cs_new_protected:Npn __siunitx_print_text:n #1 202 { 203 \text 204 { 205 \bool_if:NT \l__siunitx_print_text_family_bool 206 { \rmfamily } 207 \bool_if:NT \l__siunitx_print_text_series_bool 208 { \mdseries } 209 \bool_if:NT \l__siunitx_print_text_shape_bool 210 { \upshape } 211 __siunitx_print_text_replace:n {#1} 212 } 213 } </pre>
---	---

To get math mode material to print in text mode, various search-and-replace steps are needed. The `\mathrm` command is also disabled as it is invalid in text mode.

```

214 \cs_new_protected:Npn \__siunitx_print_text_replace:n #1
215 {
216   \group_begin:
217   \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
218   \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
219   \tl_use:N \l__siunitx_print_tmp_tl
220   \group_end:
221 }
222 \cs_new_protected:Npx \__siunitx_print_text_replace:N #1
223 {
224   \exp_not:N \__siunitx_print_text_replace:NNn #1

```

```

225 \exp_not:N \mathrm
226 { \exp_not:N \use:n }
227 \exp_not:N \pm
228 { \exp_not:N \textpm }
229 \exp_not:N \mp
230 { \exp_not:n { \ensuremath { \mp } } } }
231 -
232 { \exp_not:N \textminus }
233 \char_generate:nn { '\_ } { 8 }
234 { \exp_not:N \__siunitx_print_text_sub:n }
235 ^
236 { \exp_not:N \__siunitx_print_text_super:n }
237 \exp_not:N \q_recursion_tail
238 { ? }
239 \exp_not:N \q_recursion_stop
240 }
241 \cs_new_protected:Npn \__siunitx_print_text_replace:NNn #1#2#3
242 {
243   \quark_if_recursion_tail_stop:N #2
244   \tl_replace_all:Nnn #1 {#2} {#3}
245   \__siunitx_print_text_replace:NNn #1
246 }

```

Sub- and superscripts can be in any order in the source. The first step of handling them is therefore to do a look-ahead to work out whether only one or both are present.

```

247 \cs_new_protected:Npn \__siunitx_print_text_sub:n #1
248 {
249   \__siunitx_print_text_scripts:NnN
250   \textsubscript {#1} \__siunitx_print_text_super:n
251 }
252 \cs_new_protected:Npn \__siunitx_print_text_super:n #1
253 {
254   \__siunitx_print_text_scripts:NnN
255   \textsuperscript {#1} \__siunitx_print_text_sub:n
256 }
257 \cs_new_protected:Npn \__siunitx_print_text_scripts:NnN #1#2#3
258 {
259   \cs_set_protected:Npn \__siunitx_print_text_scripts:
260   {
261     \if_meaning:w \l_peek_token #3
262       \exp_after:wN \__siunitx_print_text_scripts_two:NnNn
263     \else:
264       \exp_after:wN \__siunitx_print_text_scripts_one:Nn
265     \fi:
266     #1 {#2}
267   }
268   \peek_after:Nw \__siunitx_print_text_scripts:
269 }
270 \cs_new_protected:Npn \__siunitx_print_text_scripts: { }
271 \cs_new_protected:Npn \__siunitx_print_text_scripts_one:Nn #1#2
272 {
273   \group_begin:

```

```

274     \tl_set:Nn \l__siunitx_print_tmp_tl {#2}
275     \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
276     \exp_args:NNV \group_end:
277     #1 \l__siunitx_print_tmp_tl
278 }

```

For the two scripts case, we cannot use `\textsubscript/\textsuperscript` as they don't stack directly. Instead, we sort out the ordering then use an implementation for both parts that is the same as the kernel text scripts.

```

279 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:NnNn #1#2#3#4
280 {
281   \cs_if_eq:NNTF #1 \textsubscript
282   { \__siunitx_print_text_scripts_two:nn {#4} {#2} }
283   { \__siunitx_print_text_scripts_two:nn {#2} {#4} }
284 }
285 \cs_new_protected:Npx \__siunitx_print_text_scripts_two:nn #1#2
286 {
287   \group_begin:
288   \exp_not:N \m@th
289   \exp_not:N \ensuremath
290   {
291     ^ { \exp_not:N \__siunitx_print_text_scripts_two:n {#1} }
292     \char_generate:nn { '\_ } { 8 }
293     { \exp_not:N \__siunitx_print_text_scripts_two:n {#2} }
294   }
295   \group_end:
296 }
297 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:n #1
298 {
299   \mbox
300   {
301     \fontsize \sf@size \z@ \selectfont
302     \__siunitx_print_text_scripts_one:Nn \use:n {#1}
303   }
304 }

```

(End definition for `__siunitx_print_text:n` and others.)

1.3 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.

```

305 \keys_set:nn { siunitx }
306 {
307   color           = ,
308   mode            = math ,
309   number-color    = ,
310   number-mode     = math ,
311   propagate-math-font = false ,
312   reset-math-version = true ,
313   reset-text-shape = true ,
314   reset-text-series = true ,
315   reset-text-family = true ,
316   text-family-to-math = false ,

```

```

317     text-weight-to-math = false ,
318     unit-color          =         ,
319     unit-mode            = math
320   }
321 </package>

```


Part V

siunitx-table – Formatting numbers in tables

1 siunitx-table implementation

Start the DocStrip guards.

```
1 \*package
```

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2 \@@=siunitx_table
```

```
\l__siunitx_table_tmp_tl
```

 Scratch space.

```
3 \tl_new:N \l__siunitx_table_tmp_tl
```

(End definition for `\l__siunitx_table_tmp_tl`.)

1.1 Interface functions

```
\l__siunitx_table_parse_bool
```

 One top-level setting applies to all number cells.

```
4 \keys_define:nn { siunitx / table }
```

```
5 {
```

```
6   parse .bool_set:N = \l__siunitx_table_parse_bool
```

```
7 }
```

(End definition for `\l__siunitx_table_parse_bool`.)

```
\siunitx_cell_begin:
```

```
\siunitx_cell_end:
```

```
8 \cs_new_protected:Npn \siunitx_cell_begin:
```

```
9 {
```

```
10   \bool_if:NTF \l__siunitx_table_parse_bool
```

```
11     { \__siunitx_table_collect_begin:N }
```

```
12     { \__siunitx_table_direct_begin:N }
```

```
13 }
```

```
14 \cs_new_protected:Npn \siunitx_cell_end:
```

```
15 {
```

```
16   \__siunitx_table_collect_end:
```

```
17 }
```

(End definition for `\siunitx_cell_begin:` and `\siunitx_cell_end:`. These functions are documented on page ??.)

1.2 Collecting tokens

`\l__siunitx_table_collect_tl` Space for tokens.

```
18 \tl_new:N \l__siunitx_table_collect_tl
```

(End definition for `\l__siunitx_table_collect_tl`.)

`__siunitx_table_collect_begin:N` Collecting a tabular cell means doing a token-by-token collection. In previous versions of `siunitx` that was done along with picking out the numerical part, but the code flow ends up very tricky. Here, therefore, we just collect up the unchanged tokens first. The `#1` collects the leading `\ignorespaces` from the template: that has to go to avoid issues with the peek-ahead code. The definition of `\cr` is used to allow collection of any tokens inserted after the main content when dealing with the last cell of a row: the “group” around it is needed to avoid issues with the underlying `\halign`. (The approach is based on that in `colcell`.) Notice that as each cell forms a group there is no need to reset the definition of `\cr`.

```
19 \cs_new_protected:Npn \__siunitx_table_collect_begin:N #1
20 {
21   \tl_clear:N \l__siunitx_table_collect_tl
22   \if_false: { \fi:
23     \cs_set_protected:Npn \cr
24     {
25       \__siunitx_table_collect_loop:
26       \tex_cr:D
27     }
28   \if_false: } \fi:
29   \__siunitx_table_collect_loop:
30 }
```

(End definition for `__siunitx_table_collect_begin:N`.)

`__siunitx_table_collect_loop:` Collecting up the cell content needs a loop: this is done using a `peek` approach as it’s most natural. (A slower approach is possible using something like the `\tl_lower_case:n` loop code.) The set of possible tokens is somewhat limited compared to an arbitrary cell (*cf.* the approach in `colcell`): the special cases are pulled out for manual handling. The flexible lookup approach is more-or-less the same idea as in the kernel `case` functions. The `\relax` special case covers the case where `\` has been expanded in an empty cell.

```
31 \cs_new_protected:Npn \__siunitx_table_collect_loop:
32 {
33   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
34   { \__siunitx_table_collect_group:n }
35   { \__siunitx_table_collect_token:N }
36 }
37 \cs_new_protected:Npn \__siunitx_table_collect_group:n #1
38 {
39   \tl_put_right:Nn \l__siunitx_table_collect_tl { {#1} }
40   \__siunitx_table_collect_loop:
41 }
42 \cs_new_protected:Npn \__siunitx_table_collect_token:N #1
43 {
44   \__siunitx_table_collect_search:NnF #1
45   {
46     \unskip          { \__siunitx_table_collect_loop: }
```

```

47         \end                { \tabularnewline \end }
48         \relax              { \relax }
49         \tabularnewline     { \tabularnewline }
50         \siunitx_cell_end: { \siunitx_cell_end: }
51     }
52     {
53         \tl_put_right:Nn \l__siunitx_table_collect_tl {#1}
54         \__siunitx_table_collect_loop:
55     }
56 }
57 \AtBeginDocument
58 {
59     \@ifpackageloaded { mdwtab }
60     {
61         \cs_set_protected:Npn \__siunitx_table_collect_token:N #1
62         {
63             \__siunitx_table_collect_search:NnF #1
64             {
65                 \@maybe@unskip    { \__siunitx_table_collect_loop: }
66                 \tab@setcr         { \__siunitx_table_collect_loop: }
67                 \unskip            { \__siunitx_table_collect_loop: }
68                 \end                { \tabularnewline \end }
69                 \relax             { \relax }
70                 \tabularnewline    { \tabularnewline }
71                 \siunitx_cell_end: { \siunitx_cell_end: }
72             }
73             {
74                 \tl_put_right:Nn \l__siunitx_table_collect_tl {#1}
75                 \__siunitx_table_collect_loop:
76             }
77         }
78     }
79     { }
80 }
81 \cs_new_protected:Npn \__siunitx_table_collect_search:NnF #1#2#3
82 {
83     \__siunitx_table_collect_search_aux:NNn #1
84     #2
85     #1 {#3}
86     \q_stop
87 }
88 \cs_new_protected:Npn \__siunitx_table_collect_search_aux:NNn #1#2#3
89 {
90     \token_if_eq_meaning:NNTF #1 #2
91     { \use_i_delimit_by_q_stop:nw {#3} }
92     { \__siunitx_table_collect_search_aux:NNn #1 }
93 }

```

(End definition for `__siunitx_table_collect_loop:` and others.)

1.3 Separating collected material

The input needs to be divided into numerical tokens and those which appear before and after them. This needs a second loop and validation.

```

\l__siunitx_table_pre_tl Space for tokens.
\l__siunitx_table_number_tl 94 \tl_new:N \l__siunitx_table_pre_tl
\l__siunitx_table_post_tl 95 \tl_new:N \l__siunitx_table_number_tl
96 \tl_new:N \l__siunitx_table_post_tl

(End definition for \l__siunitx_table_pre_tl, \l__siunitx_table_number_tl, and \l__siunitx_
table_post_tl.)

\__siunitx_table_collect_end: At the end of the cell, expand all of the content as far as possible then split it up into
numerical and non-numerical parts.

97 \cs_new_protected:Npn \__siunitx_table_collect_end:
98 {
99   \protected@edef \l__siunitx_table_collect_tl
100   { \l__siunitx_table_collect_tl }
101   \tl_clear:N \l__siunitx_table_pre_tl
102   \tl_clear:N \l__siunitx_table_number_tl
103   \tl_clear:N \l__siunitx_table_post_tl
104   \exp_after:wN \__siunitx_table_split_loop: \l__siunitx_table_collect_tl
105   \q_recursion_tail \q_recursion_stop
106   \__siunitx_table_split_tidy:N \l__siunitx_table_pre_tl
107   \__siunitx_table_split_tidy:N \l__siunitx_table_post_tl
108   \tl_if_empty:NTF \l__siunitx_table_number_tl
109   { \__siunitx_table_print_text:V \l__siunitx_table_pre_tl }
110   {
111     \__siunitx_table_print:VVV
112     \l__siunitx_table_pre_tl
113     \l__siunitx_table_number_tl
114     \l__siunitx_table_post_tl
115   }
116 }

(End definition for \__siunitx_table_collect_end:.)

\__siunitx_table_split_loop: Splitting into parts uses the fact that numbers cannot contain groups and that we can
\__siunitx_table_split_group:n track where we are up to based on the content of the token lists.
\__siunitx_table_split_token:N 117 \cs_new_protected:Npn \__siunitx_table_split_loop:
118 {
119   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
120   { \__siunitx_table_split_group:n }
121   { \__siunitx_table_split_token:N }
122 }
123 \cs_new_protected:Npn \__siunitx_table_split_group:n #1
124 {
125   \tl_if_empty:NTF \l__siunitx_table_number_tl
126   { \tl_put_right:Nn \l__siunitx_table_pre_tl { {#1} } }
127   { \tl_put_right:Nn \l__siunitx_table_post_tl { {#1} } }
128   \__siunitx_table_split_loop:
129 }
130 \cs_new_protected:Npn \__siunitx_table_split_token:N #1
131 {
132   \quark_if_recursion_tail_stop:N #1
133   \tl_if_empty:NTF \l__siunitx_table_post_tl
134   {
135     \siunitx_if_number_token:NTF #1

```

```

136         { \tl_put_right:Nn \l__siunitx_table_number_tl {#1} }
137     {
138         \tl_if_empty:NTF \l__siunitx_table_number_tl
139             { \tl_put_right \l__siunitx_table_pre_tl {#1} }
140             { \tl_put_right \l__siunitx_table_post_tl {#1} }
141     }
142 }
143 { \tl_put_right:Nn \l__siunitx_table_post_tl {#1} }
144 \__siunitx_table_split_loop:
145 }

```

(End definition for `__siunitx_table_split_loop:`, `__siunitx_table_split_group:n`, and `__siunitx_table_split_token:N`.)

`__siunitx_table_split_tidy:N` A quick test for the entire content being surrounded by a set of braces: rather than look explicitly, use the fact that a string comparison can detect the same thing. The auxiliary is needed to avoid having to go *via* a :D function (for the expansion behaviour).

```

146 \cs_new_protected:Npn \__siunitx_table_split_tidy:N #1
147 {
148     \tl_if_empty:NF #1
149     { \__siunitx_table_split_tidy:NV #1 #1 }
150 }
151 \cs_new_protected:Npn \__siunitx_table_split_tidy:Nn #1#2
152 {
153     \str_if_eq_x:nnT
154         { \exp_not:n {#2} }
155         { { \exp_not:o { \use:n #2 } } }
156         { \tl_set:Nn #1 { \use:n #2 } }
157 }
158 \cs_generate_variant:Nn \__siunitx_table_split_tidy:Nn { NV }

```

(End definition for `__siunitx_table_split_tidy:N` and `__siunitx_table_split_tidy:Nn`.)

1.4 Printing numbers in cells: spacing

Getting the general alignment correct in tables is made more complex than one would like by the `colortbl` package. In the original $\text{\LaTeX} 2_{\epsilon}$ definition, cell material is centred by a construction of the (primitive) form

```

\hfil
#
\hfil

```

which only uses `fil` stretch. That is altered by `colortbl` to broadly

```

\hskip 0pt plus 0.5fill
\kern 0pt
#
\hskip 0pt plus 0.5fill

```

which means there is `fill` stretch to worry about and the `kern` as well.

`__siunitx_table_skip:n` To prevent combination of skips, a kern is inserted after each one. This is best handled as a short auxiliary.

```

159 \cs_new_protected:Npn \__siunitx_table_skip:n #1
160 {
161   \skip_horizontal:n {#1}
162   \tex_kern:D \c_zero_skip
163 }

```

(End definition for `__siunitx_table_skip:n`.)

`\l_siunitx_table_column_width_dim` Settings which apply to aligned columns in general.

```

\l_siunitx_table_fixed_width_bool
164 \keys_define:nn { siunitx / table }
165 {
166   column-width .dim_set:N =
167     \l_siunitx_table_column_width_dim ,
168   fixed-width .bool_set:N =
169     \l_siunitx_table_fixed_width_bool
170 }

```

(End definition for `\l_siunitx_table_column_width_dim` and `\l_siunitx_table_fixed_width_bool`.)

`_siunitx_table_align_center:n` The beginning and end of each table cell have to adjust the position of the content using glue. When `colortbl` is loaded the glue is done in two parts: one for our positioning and one to explicitly override that from the package. Using a two-step auxiliary chain avoids needing to repeat any code and the impact of the extra expansion should be trivial.

```

\_siunitx_table_align_left:n
\_siunitx_table_align_right:n
\_siunitx_table_align_auxi:nn
\_siunitx_table_align_auxii:nn
171 \cs_new_protected:Npn \_siunitx_table_align_center:n #1
172 { \_siunitx_table_align_auxi:nn {#1} { Opt plus 0.5fill } {#1} }
173 \cs_new_protected:Npn \_siunitx_table_align_left:n #1
174 { \_siunitx_table_align_auxi:nn {#1} { Opt } {#1} }
175 \cs_new_protected:Npn \_siunitx_table_align_right:n #1
176 { \_siunitx_table_align_auxi:nn {#1} { Opt plus 1fill } {#1} }
177 \cs_new_protected:Npn \_siunitx_table_align_auxi:nn #1#2
178 {
179   \bool_if:NTF \l_siunitx_table_fixed_width_bool
180     { \hbox_to_wd:nn \l_siunitx_table_column_width_dim }
181     { \use:n }
182     {
183       \_siunitx_table_skip:n {#2}
184       #1
185       \_siunitx_table_skip:n { Opt plus 1fill - #2 }
186     }
187 }
188 \AtBeginDocument
189 {
190   \ifpackageloaded { colortbl }
191   {
192     \cs_new_eq:NN
193       \_siunitx_table_align_auxii:nn
194       \_siunitx_table_align_auxi:nn
195     \cs_set_protected:Npn \_siunitx_table_align_auxi:nn #1#2
196     {
197       \_siunitx_table_skip:n{ Opt plus -0.5fill }
198       \_siunitx_table_align_auxii:nn {#1} {#2}
199       \_siunitx_table_skip:n { Opt plus -0.5fill }

```

```

200         }
201     }
202     { }
203 }

```

(End definition for `_siunitx_table_align_center:n` and others.)

1.5 Printing just text

In cases where there is no numerical part, siunitx allows alignment of the “escaped” text independent of the underlying column type.

`\l_siunitx_table_align_text_tl` Alignment is handled using a `tl` as this allows a fast lookup at the point of use.

```

204 \keys_define:nn { siunitx / table }
205 {
206     text-alignment .choices:nn =
207     { center , left , right }
208     { \tl_set:Nn \l__siunitx_table_align_text_tl {#1} } ,
209 }
210 \tl_new:N \l__siunitx_table_align_text_tl

```

(End definition for `\l__siunitx_table_align_text_tl`.)

`_siunitx_table_print_text:n` Printing escaped text is easy: just place it in correctly in the column.

```

\_siunitx_table_print_text:V 211 \cs_new_protected:Npn \_siunitx_table_print_text:n #1
212 {
213     \use:c { __siunitx_table_align_ \l__siunitx_table_align_text_tl :n } {#1}
214 }
215 \cs_generate_variant:Nn \_siunitx_table_print_text:n { V }

```

(End definition for `_siunitx_table_print_text:n`.)

1.6 Reserving space: the table format

`\l__siunitx_table_format_tl`

```

216 \keys_define:nn { siunitx / table }
217 {
218     format .tl_set:N = \l__siunitx_table_format_tl
219 }

```

(End definition for `\l__siunitx_table_format_tl`.)

1.7 Directly printing without collection

Collecting the number allows for various effects but is not as fast as simply aligning on the first token that is a decimal marker. The strategy here is that used by `dcolum`.

`_siunitx_table_direct_begin:N` After removing the `\ignorespaces` at the start of the cell, check to see if there is a {
`_siunitx_table_direct_begin_aux:` and branch as appropriate.

```

220 \cs_new_protected:Npn \_siunitx_table_direct_begin: #1
221 {
222     \peek_catcode_ignore_spaces:NTF \c_group_begin_token
223     { \_siunitx_table_print_text:n }
224     {

```

```

225         \m@th
226         \__siunitx_table_direct_begin_aux:
227     }
228 }
229 \cs_new_protected:Npn \__siunitx_table_direct_begin_aux:
230 {
231 }

```

(End definition for `__siunitx_table_direct_begin:N` and `__siunitx_table_direct_begin_aux:.`)

1.8 Printing numbers in cells: main functions

Alignment is handled using a `tl` as this allows a fast lookup at the point of use.

```

\l_siunitx_table_align_comparator_bool
\l_siunitx_table_align_exponent_bool
\l_siunitx_table_align_uncertainty_bool
\l_siunitx_table_alignment_tl
\l_siunitx_table_parse_only_bool
232 \keys_define:nn { siunitx / table }
233 {
234     align-comparator .bool_set:N =
235         \l_siunitx_table_align_comparator_bool ,
236     align-exponent .bool_set:N =
237         \l_siunitx_table_align_exponent_bool ,
238     align-uncertainty .bool_set:N =
239         \l_siunitx_table_align_uncertainty_bool ,
240     alignment .choices:nn =
241         { center , left , right }
242         { \tl_set:Nn \l_siunitx_table_alignment_tl {#1} } ,
243     parse-only .bool_set:N =
244         \l_siunitx_table_parse_only_bool
245 }
246 \tl_new:N \l_siunitx_table_alignment_tl

```

(End definition for `\l_siunitx_table_align_comparator_bool` and others.)

```

\__siunitx_table_print:nnn
\__siunitx_table_print:VVV
\__siunitx_table_print_non_aligned:nnn
\__siunitx_table_print_aligned:nnn
247 \cs_new_protected:Npn \__siunitx_table_print:nnn #1#2#3
248 {
249     \use:c { __siunitx_table_align_ \l_siunitx_table_alignment_tl :n }
250     {
251         \bool_if:NTF \l_siunitx_table_parse_only_bool
252             { \__siunitx_table_print_non_aligned:nnn }
253             { \__siunitx_table_print_aligned:nnn }
254             {#1} {#2} {#3}
255     }
256 }
257 \cs_generate_variant:Nn \__siunitx_table_print:nnn { VVV }
258 \cs_new_protected:Npn \__siunitx_table_print_non_aligned:nnn #1#2#3
259 {
260     #1
261     \siunitx_number_format:nN {#2} \l_siunitx_table_tmp_tl
262     \siunitx_print:nV { number } \l_siunitx_table_tmp_tl
263     #3
264 }
265 \cs_new_protected:Npn \__siunitx_table_print_aligned:nnn #1#2#3
266 {
267 }

```

(End definition for `__siunitx_table_print:nnn`, `__siunitx_table_print_non_aligned:nnn`, and `__siunitx_table_print_aligned:nnn`.)

1.9 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.

```
268 \keys_set:nn { siunitx / table }
269 {
270   align-comparator = false ,
271   align-exponent   = false ,
272   align-uncertainty = false ,
273   alignment        = center ,
274   column-width     = 0pt   ,
275   fixed-width      = false ,
276   format           =      ,
277   parse            = true  ,
278   parse-only       = false ,
279   text-alignment   = center
280 }
281 \endpackage
```

Part VI

siunitx-unit – Parsing and formatting units

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^AT_EX 2_ε math mode commands are assumed to be available as part of the formatted output. The `\mathchoice` command (normally the T_EX primitive) is needed when using `per-mode = symbol-or-fraction`. The command `\mathrm` is used for wrapping the text (letter) part of units. The commands `\frac`, `\mbox`, `_` and `\,` are used by the standard module settings, and `\ensuremath`, `\hbar`, `\mathit` and `\mathrm` in some standard unit definitions (for atomic and natural units). 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: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}
```

<code>\siunitx_unit_format:nNN</code>	<code>\siunitx_unit_format:nNN {<units>} <tl var> <fp var></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: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.

2 Defining symbolic units

<code>\siunitx_declare_prefix:Nnn</code>	<code>\siunitx_declare_prefix:Nnn <prefix> {<symbol>} {<power>}</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 <prefix> {<symbol>}</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 <pre-power> <post-power> {<value>}</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 <qualifier> {<meaning>}</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.

<hr/> <code>\siunitx_declare_unit:Nn</code>	<code>\siunitx_declare_unit:Nn <unit> {<meaning>}</code>
<code>\siunitx_declare_unit:Nx</code> <hr/>	Defines a symbolic $\langle unit \rangle$ (which should be a control sequence such as <code>\kilogram</code>) to be converted by the parser to the $\langle meaning \rangle$. The latter may consist of literal content (e.g. <code>kg</code>), other symbolic unit commands (e.g. <code>\kilo\gram</code>) or a mixture of the two. In literal mode the $\langle meaning \rangle$ will be typeset directly.

`\l_siunitx_unit_symbolic_seq`

This sequence contains all of the symbolic $\langle unit \rangle$ names defined : these will be in the form of control sequences such as `\kilogram`. The order of the sequence is unimportant.

3 Pre-defined symbolic unit components

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

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

<hr/> <code>\kilogram</code>	The base units as defined in Section 2.1 of the SI Brochure [2]. 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> <hr/>	

<hr/> <code>\gram</code> <hr/>	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> .
--------------------------------	--

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

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 [4].

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

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

The defined SI units with defined names and symbols, as given in Section 2.2.2 of the SI Brochure [3]. Notice that the names of the units are lower case with the exception of `\degreeCelsius`, and that this unit name includes “degree”.

`\day`
`\hectare`
`\hour`
`\litre`
`\liter`
`\minute`
`\tonne`

Units accepted for use with the SI: here `\minute` is a unit of time not of plane angle. These units are taken from Table 4.1 of the SI Brochure [6].

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

<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 4.1 of the SI Brochure [6].
<code>\arcsecond</code>	
<code>\degree</code>	
<hr/>	
<code>\astronomicalunit</code>	Non-SI where values must be determined experimentally. These units are taken from Table 7 of the SI Brochure [7]. Where no better name is given for the unit in the SI Brochure, the prefixes <code>nu</code> (natural unit) and <code>au</code> (atomic unit) are used.
<code>\atomicmassunit</code>	
<code>\auaction</code>	Note that the value of the natural unit of speed (the speed of light) is used to define the second and is thus not determined by experiment: it is however included in this set of units.
<code>\aucharge</code>	
<code>\auenergy</code>	
<code>\aulength</code>	
<code>\aumass</code>	
<code>\autime</code>	
<code>\bohr</code>	
<code>\dalton</code>	
<code>\electronvolt</code>	
<code>\hartree</code>	
<code>\nuaction</code>	
<code>\numass</code>	
<code>\nuspeed</code>	
<code>\nutime</code>	
<hr/>	
<code>\angstrom</code>	Non-SI units accepted for use with the SI. These units are taken from Table 8 of the SI Brochure [8].
<code>\bar</code>	
<code>\barn</code>	
<code>\bel</code>	
<code>\decibel</code>	
<code>\knot</code>	
<code>\millimetremercury</code>	
<code>\nauticalmile</code>	
<code>\neper</code>	
<hr/>	
<code>\dyne</code>	Non-SI units associated with the CGS and the CGS-Gaussian system of units. These units are taken from Table 9 of the SI Brochure [9].
<code>\erg</code>	
<code>\gal</code>	
<code>\gauss</code>	
<code>\maxwell</code>	
<code>\oersted</code>	
<code>\phot</code>	
<code>\poise</code>	
<code>\stilb</code>	
<code>\stokes</code>	
<hr/>	
<code>\percent</code>	The mathematical concept of percent, usable with the SI as detailed in Section 5.3.7 of the SI Brochure [5].
<hr/>	
<code>\square</code>	<code>\square</code> $\langle prefix \rangle \langle unit \rangle$
<code>\cubic</code>	<code>\cubic</code> $\langle prefix \rangle \langle unit \rangle$
<hr/>	
	Pre-defined unit powers which apply to the next $\langle prefix \rangle / \langle unit \rangle$ combination.

<hr/> <code>\squared</code>	<code>\langle prefix \rangle \langle unit \rangle \backslash squared</code>
<code>\cubed</code>	<code>\langle prefix \rangle \langle unit \rangle \backslash cubed</code>

Pre-defined unit powers which apply to the preceding $\langle prefix \rangle / \langle unit \rangle$ combination.

<hr/> <code>\per</code>	<code>\per \langle prefix \rangle \langle unit \rangle \langle power \rangle</code>
-------------------------	---

Indicates that the next $\langle prefix \rangle / \langle unit \rangle / \langle power \rangle$ 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 \langle prefix \rangle \langle unit \rangle \langle power \rangle</code>
----------------------------	--

Indicates that the next $\langle prefix \rangle / \langle unit \rangle / \langle power \rangle$ 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 { \langle color \rangle } \langle prefix \rangle \langle unit \rangle \langle power \rangle</code>
-------------------------------	---

Indicates that the next $\langle prefix \rangle / \langle unit \rangle / \langle power \rangle$ combination should be highlighted in the specified $\langle color \rangle$. 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>\langle prefix \rangle \langle unit \rangle \langle power \rangle \backslash of { \langle qualifier \rangle }</code>
------------------------	--

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

<hr/> <code>\raiseto</code>	<code>\raiseto { \langle power \rangle } \langle prefix \rangle \langle unit \rangle</code>
<code>\tothe</code>	<code>\langle prefix \rangle \langle unit \rangle \backslash tothe { \langle power \rangle }</code>

Indicates that the $\langle power \rangle$ applies to the current $\langle prefix \rangle / \langle unit \rangle$ combination. As shown, `\raiseto` applies to the next $\langle unit \rangle$ 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.

3.1 Key-value options

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

<hr/> <code>bracket-denominator</code>	<code>bracket-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/> fraction-command	<p><code>fraction-command = \langlecommand\rangle</code></p> <p>Command used to create fractional output when <code>per-mode</code> is set to <code>fraction</code>. The standard setting is <code>\frac</code>.</p>
<hr/> <hr/> parse-units	<p><code>parse-units = true false</code></p> <p>Determines whether parsing of unit symbols is attempted or literal mode is used directly. The standard setting is <code>true</code>.</p>
<hr/> <hr/> per-mode	<p><code>per-mode = \langlechoice\rangle</code></p> <p>Selects how the negative powers (<code>\per</code>) are formatted: a choice from the options <code>fraction</code>, <code>power</code>, <code>power-positive-first</code>, <code>repeated-symbol</code>, <code>symbol</code> and <code>symbol-or-fraction</code>. The option <code>fraction</code> generates fractional output when appropriate using the command specified by the <code>fraction-command</code> option. The setting <code>power</code> uses reciprocal powers leaving the units in the order of input, while <code>power-positive-first</code> uses the same display format but sorts units such that the positive powers come before negative ones. The <code>symbol</code> setting uses a symbol (specified by <code>per-symbol</code>) between positive and negative powers, while <code>repeated-symbol</code> 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, <code>symbol-or-fraction</code> acts like <code>symbol</code> for inline output and like <code>fraction</code> when the output is used in a display math environment. The standard setting is <code>power</code>.</p>
<hr/> <hr/> per-symbol	<p><code>per-symbol = \langlesymbol\rangle</code></p> <p>Specifies the symbol to be used to denote negative powers when the option <code>per-mode</code> is set to <code>repeated-symbol</code>, <code>symbol</code> or <code>symbol-or-fraction</code>. The standard setting is <code>/</code>.</p>
<hr/> <hr/> qualifier-mode	<p><code>qualifier-mode = \langlechoice\rangle</code></p> <p>Selects how qualifiers are formatted: a choice from the options <code>brackets</code>, <code>combine</code>, <code>phrase</code> and <code>subscript</code>. The option <code>bracket</code> wraps the qualifier in parenthesis, <code>combine</code> joins the qualifier with the unit directly, <code>phrase</code> inserts the content stored by the option <code>qualifier-phrase</code> between the unit and qualifier, and <code>subscript</code> formats the qualifier as a subscript. The standard setting is <code>subscript</code>.</p>
<hr/> <hr/> qualifier-phrase	<p><code>qualifier-phrase = \langlechoice\rangle</code></p> <p>Defines the text to be inserted between a unit and qualifier when <code>qualifier-mode</code> is set to <code>phrase</code>. This material is inserted without any font control and so if text mode is required it should be included in the setting, for example <code>_ \mbox{of} _</code>. The standard setting is a full width space (<code>_</code>).</p>
<hr/> <hr/> sticky-per	<p><code>sticky-per = true false</code></p> <p>Used to determine whether <code>\per</code> should be applied one a unit-by-unit basis (when <code>false</code>) or should apply to all following units (when <code>true</code>). The latter mode is somewhat akin conceptually to the T_EX <code>\over</code> primitive. The standard setting is <code>false</code>.</p>
<hr/> <hr/> unit-close-bracket	<p><code>unit-close-bracket = \langlesymbol\rangle</code></p> <p>Bracket symbol used to close a matched pair around units when once is required to maintain mathematical logic. The standard setting is <code>)</code>.</p>

<hr/> <hr/> unit-open-bracket	<code>unit-open-bracket = $\langle symbol \rangle$</code> Bracket symbol used to open a matched pair around units when once is required to maintain mathematical logic. The standard setting is (.
<hr/> <hr/> unit-product	<code>unit-product = $\langle separator \rangle$</code> Inserted between unit combinations in parsed mode, and used to replace . and ~ in literal mode. The standard setting is \,.

4 siunitx-unit implementation

Start the DocStrip guards.

```
1  $\langle *package \rangle$ 
```

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2  $\langle @@=siunitx_{unit} \rangle$ 
```

4.1 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:nn { \_ } { 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. If an `f`-type definition then yields nothing at all then the assumption is that the input is symbolic. (We use `f`-type expansion since it will turn the symbolic unit names into nothing at all but doesn’t require them to be expandable.)

```

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     \tl_set:Nf \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`.)

4.2 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 57.)

`_siunitx_unit_set_symbolic:Nnn`
`_siunitx_unit_set_symbolic:Npnn`
`_siunitx_unit_set_symbolic:NNpnn` 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.

```

22 \cs_new_protected:Npn \_siunitx_unit_set_symbolic:Nnn #1
23   { \_siunitx_unit_set_symbolic:NNnnn \cs_set:cpn #1 { } }
24 \cs_new_protected:Npn \_siunitx_unit_set_symbolic:Npnn #1#2#
25   { \_siunitx_unit_set_symbolic:NNnnn \cs_set:cpn #1 {#2} }
26 \cs_new_protected:Npn \_siunitx_unit_set_symbolic:NNnnn #1#2#3#4#5

```

```

27 {
28   \seq_put_right:Nn \l_siunitx_unit_symbolic_seq {#2}
29   #1 { units ~ > ~ \token_to_str:N #2 } #3
30   {
31     \bool_if:NF \l__siunitx_unit_test_bool
32     {
33       \bool_if:NTF \l__siunitx_unit_parsing_bool
34       {#5}
35       {#4}
36     }
37   }
38 }

```

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

`\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.

```

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

```

(End definition for `\siunitx_declare_power:Nnn`. This function is documented on page 56.)

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

```

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

```

(End definition for `\siunitx_declare_prefix:Nn` and others. These functions are documented on page 56.)

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

```

62 \cs_new_protected:Npn \siunitx_declare_qualifier:Nn #1#2
63 {
64   \__siunitx_unit_set_symbolic:Nnn #1
65   { \ ( #2 ) }
66   { \__siunitx_unit_parse_qualifier:nn {#1} {#2} }
67 }

```

(End definition for `\siunitx_declare_qualifier:Nn`. This function is documented on page 56.)

`\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`

```

68 \cs_new_protected:Npn \siunitx_declare_unit:Nn #1#2
69 {
70   \__siunitx_unit_set_symbolic:Nnn #1
71   {#2}
72   {
73     \__siunitx_unit_if_symbolic:nTF {#2}
74     {#2}
75     { \__siunitx_unit_parse_unit:Nn #1 {#2} }
76   }
77 }
78 \cs_generate_variant:Nn \siunitx_declare_unit:Nn { Nx }

```

(End definition for `\siunitx_declare_unit:Nn`. This function is documented on page 57.)

4.3 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!

```

79 \__siunitx_unit_set_symbolic:Nnn \per
80 { / }
81 { \__siunitx_unit_parse_per: }

```

(End definition for `\per`. This function is documented on page 60.)

`\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.

```

82 \__siunitx_unit_set_symbolic:Npnn \cancel
83 { \__siunitx_unit_literal_special:nN { \cancel } }
84 { \__siunitx_unit_parse_special:n { \cancel } }
85 \__siunitx_unit_set_symbolic:Npnn \highlight #1
86 { \__siunitx_unit_literal_special:nN { \textcolor {#1} } }
87 { \__siunitx_unit_parse_special:n { \textcolor {#1} } }

```

(End definition for `\cancel` and `\highlight`. These functions are documented on page 60.)

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

```
88 \__siunitx_unit_set_symbolic:Npnn \of #1
89   { \ ( #1 ) }
90   { \__siunitx_unit_parse_qualifier:nn { \of {#1} } {#1} }
```

(End definition for `\of`. This function is documented on page 60.)

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

\tothe

```
91 \__siunitx_unit_set_symbolic:Npnn \raiseto #1
92   { \__siunitx_unit_literal_power:nN {#1} }
93   { \__siunitx_unit_parse_power:nnN { \raiseto {#1} } {#1} \c_true_bool }
94 \__siunitx_unit_set_symbolic:Npnn \tothe #1
95   { ^ {#1} }
96   { \__siunitx_unit_parse_power:nnN { \tothe {#1} } {#1} \c_false_bool }
```

(End definition for `\raiseto` and `\tothe`. These functions are documented on page 60.)

4.4 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_product_tl` Options which apply to the main formatting routine, and so are not tied to either symbolic or literal input.

```
97 \keys_define:nn { siunitx }
98   {
99     unit-product .tl_set:N = \l__siunitx_unit_product_tl
100   }
```

(End definition for `\l__siunitx_unit_product_tl`.)

`\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.

```
101 \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 into a power of ten. This variation is handled using two separate functions: as this submodule does not really deal with numbers, formatting the numeral part here would be tricky and it is better therefore to have a mechanism to return a simple numerical power. At the same time, most uses will no want this more complex return format and so a version of the code which does not do this is also provided.

\siunitx_unit_format:nNN

`__siunitx_unit_format:nNN`

`__siunitx_unit_format_aux:`

The main unit formatting routine groups all of the parsing/formatting, so that the 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 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.

```

102 \cs_new_protected:Npn \siunitx_unit_format:nN #1#2
103 {
104   \bool_set_false:N \l__siunitx_unit_prefix_power_bool
105   \__siunitx_unit_format:nNN {#1} #2 \l__siunitx_unit_tmp_fp
106 }
107 \cs_new_protected:Npn \siunitx_unit_format:nNN #1#2#3
108 {
109   \bool_set_true:N \l__siunitx_unit_prefix_power_bool
110   \__siunitx_unit_format:nNN {#1} #2 #3
111 }
112 \cs_new_protected:Npn \__siunitx_unit_format:nNN #1#2#3
113 {
114   \group_begin:
115   \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
116   { \cs_set_eq:Nc ##1 { units ~ > ~ \token_to_str:N ##1 } }
117   \tl_clear:N \l__siunitx_unit_formatted_tl
118   \fp_zero:N \l__siunitx_unit_prefix_fp
119   \bool_if:NTF \l__siunitx_unit_parse_bool
120   {
121     \__siunitx_unit_if_symbolic:nTF {#1}
122     {
123       \__siunitx_unit_parse:n {#1}
124       \prop_if_empty:NF \l__siunitx_unit_parsed_prop
125       { \__siunitx_unit_format_parsed: }
126     }
127     { \__siunitx_unit_format_literal:n {#1} }
128   }
129   { \__siunitx_unit_format_literal:n {#1} }
130   \cs_set_protected:Npx \__siunitx_unit_format_aux:
131   {
132     \tl_set:Nn \exp_not:N #2
133     { \exp_not:N \l__siunitx_unit_formatted_tl }
134     \fp_set:Nn \exp_not:N #3
135     { \fp_use:N \l__siunitx_unit_prefix_fp }
136   }
137   \exp_after:wN \group_end:
138   \__siunitx_unit_format_aux:
139 }
140 \cs_new_protected:Npn \__siunitx_unit_format_aux: { }

```

(End definition for `\siunitx_unit_format:nN` and others. These functions are documented on page 55.)

4.5 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.

```
141 \cs_new_protected: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.

```
142 \cs_new_protected: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.

As with other code dealing with user input, `\protected@edef` is used here rather than `\tl_set:Nx` as L^AT_EX 2_ε robust commands may be present.

```
143 \group_begin:
144   \char_set_catcode_active:n { '~ }
145   \cs_new_protected:Npx \_siunitx_unit_format_literal:n #1
146   {
147     \group_begin:
148     \exp_not:n { \bool_set_false:N \l__siunitx_unit_parsing_bool }
149     \tl_set:Nn \exp_not:N \l__siunitx_unit_tmp_tl {#1}
150     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
151     { \exp_not:N ~ } { . }
152     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
153     { \token_to_str:N ^ } { ^ }
154     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
155     { \token_to_str:N _ } { \c__siunitx_unit_math_subscript_tl }
156     \exp_not:n
157     {
158       \protected@edef \l__siunitx_unit_tmp_tl
159       { \l__siunitx_unit_tmp_tl }
160       \tl_clear:N \l__siunitx_unit_formatted_tl
161       \tl_if_empty:NF \l__siunitx_unit_tmp_tl
162       {
163         \exp_after:wN \_siunitx_unit_format_literal_auxi:w
164         \l__siunitx_unit_tmp_tl .
165         \q_recursion_tail . \q_recursion_stop
166       }
167       \exp_args:NNNV \group_end:
168       \tl_set:Nn \l__siunitx_unit_formatted_tl
169       \l__siunitx_unit_formatted_tl
170     }
171   }
172 \group_end:
```

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.

```

173 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxi:w #1 .
174 {
175   \quark_if_recursion_tail_stop:n {#1}
176   \__siunitx_unit_format_literal_auxii:n {#1}
177   \tl_set_eq:NN \l__siunitx_unit_separator_tl \l__siunitx_unit_product_tl
178   \__siunitx_unit_format_literal_auxi:w
179 }
180 \cs_set_protected:Npn \__siunitx_unit_format_literal_auxii:n #1
181 {
182   \__siunitx_unit_format_literal_auxiii:w
183   #1 / \q_recursion_tail / \q_recursion_stop
184 }
185 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxiii:w #1 /
186 {
187   \quark_if_recursion_tail_stop:n {#1}
188   \__siunitx_unit_format_literal_auxiv:w #1 ^ ^ \q_stop
189   \tl_set:Nn \l__siunitx_unit_separator_tl { / }
190   \__siunitx_unit_format_literal_auxiii:w
191 }

```

Within each unit any sub- and superscript parts need to be separated out: wrapping everything in the font command is incorrect. The superscript part is relatively easy as there is no catcode issue or font command to add, while the subscript part needs a bit more work. As the user might have the two parts in either order, picking up the subscript needs to look in two places. We assume that only one is given: odd input here is simply accepted.

```

192 \use:x
193 {
194   \cs_new_protected:Npn \exp_not:N \__siunitx_unit_format_literal_auxiv:w
195     ##1 ^ ##2 ^ ##3 \exp_not:N \q_stop
196   {
197     \exp_not:N \__siunitx_unit_format_literal_auxv:w
198     ##1
199     \c__siunitx_unit_math_subscript_tl
200     \c__siunitx_unit_math_subscript_tl
201     \exp_not:N \q_mark
202     ##2
203     \c__siunitx_unit_math_subscript_tl
204     \c__siunitx_unit_math_subscript_tl
205     \exp_not:N \q_stop
206   }
207   \cs_new_protected:Npn \exp_not:N \__siunitx_unit_format_literal_auxv:w
208     ##1 \c__siunitx_unit_math_subscript_tl
209     ##2 \c__siunitx_unit_math_subscript_tl ##3
210     \exp_not:N \q_mark
211     ##4 \c__siunitx_unit_math_subscript_tl
212     ##5 \c__siunitx_unit_math_subscript_tl ##6
213     \exp_not:N \q_stop
214   {
215     \tl_set:Nx \exp_not:N \l__siunitx_unit_formatted_tl
216     {
217       \exp_not:N \exp_not:N
218       \exp_not:N \l__siunitx_unit_formatted_tl
219       \exp_not:N \tl_if_empty:NF

```



```

220         \exp_not:N \l__siunitx_unit_formatted_tl
221     {
222         \exp_not:N \exp_not:V
223         \exp_not:N \l__siunitx_unit_separator_tl
224     }
225 \exp_not:N \tl_if_blank:nF {##1}
226 {
227     \exp_not:N \exp_not:N
228     \exp_not:N \mathrm
229     { \exp_not:N \exp_not:n {##1} }
230 }
231 \exp_not:N \tl_if_blank:nF {##4}
232 { ^ { \exp_not:N \exp_not:n {##4} } }
233 \exp_not:N \tl_if_blank:nF {##2##5}
234 {
235     \c_siunitx_unit_math_subscript_tl
236     {
237         \exp_not:N \exp_not:N
238         \exp_not:N \mathrm
239         { \exp_not:N \exp_not:n {##2##5} }
240     }
241 }
242 }
243 }
244 }
245 \tl_new:N \l__siunitx_unit_separator_tl

```

(End definition for `_siunitx_unit_format_literal:n` and others.)

4.6 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.

`\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`.

```

246 \keys_define:nn { siunitx }
247   {
248     sticky-per .bool_set:N = \l__siunitx_unit_sticky_per_bool
249   }

```

(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.

```

250 \prop_new:N \l__siunitx_unit_parsed_prop
251 \bool_new:N \l__siunitx_unit_per_bool
252 \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.

```

253 \cs_new_protected:Npn \__siunitx_unit_parse:n #1
254   {
255     \prop_clear:N \l__siunitx_unit_parsed_prop
256     \bool_set_true:N \l__siunitx_unit_parsing_bool
257     \bool_set_false:N \l__siunitx_unit_per_bool
258     \bool_set_false:N \l__siunitx_unit_test_bool
259     \int_zero:N \l__siunitx_unit_position_int
260     #1
261     \int_step_inline:nnnn 1 1 \l__siunitx_unit_position_int
262     { \__siunitx_unit_parse_finalise:n {##1} }
263     \__siunitx_unit_parse_finalise:
264   }

```

(End definition for `__siunitx_unit_parse:n`.)

`_siunitx_unit_parse_add:nnnn` In all cases, storing a data item requires setting a temporary `t1` which will be used as the key, then using this to store the value. The `t1` is set using `x-type` expansion as this will expand the unit index and any additional calculations made for this.

```

265 \cs_new_protected:Npn \_siunitx_unit_parse_add:nnnn #1#2#3#4
266   {
267     \tl_set:Nx \l__siunitx_unit_tmp_tl { #1 - #2 }
268     \prop_if_in:NVTF \l__siunitx_unit_parsed_prop
269     \l__siunitx_unit_tmp_tl
270     {
271       \msg_error:nnxx { siunitx } { unit / duplicate-part }
272       { \exp_not:n {#1} } { { \token_to_str:N #3 } }
273     }
274     {
275       \prop_put:NVn \l__siunitx_unit_parsed_prop
276       \l__siunitx_unit_tmp_tl {#4}

```

```

277     }
278 }

```

(End definition for `_siunitx_unit_parse_add:nnnn`.)

<code>_siunitx_unit_parse_prefix:Nn</code> <code>_siunitx_unit_parse_power:nnN</code> <code>_siunitx_unit_parse_qualifier:nn</code> <code>_siunitx_unit_parse_special:n</code>	<p>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.</p>
---	---

```

279 \cs_new_protected:Npn \_siunitx_unit_parse_prefix:Nn #1#2
280 {
281   \int_set:Nn \l__siunitx_unit_tmp_int { \l__siunitx_unit_position_int + 1 }
282   \_siunitx_unit_parse_add:nnnn { prefix }
283   { \int_use:N \l__siunitx_unit_tmp_int } {#1} {#2}
284 }
285 \cs_new_protected:Npn \_siunitx_unit_parse_power:nnN #1#2#3
286 {
287   \tl_set:Nx \l__siunitx_unit_tmp_tl
288     { unit- \int_use:N \l__siunitx_unit_position_int }
289   \bool_lazy_or:nnTF
290     {#3}
291     {
292       \prop_if_in_p:Nv
293         \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
294     }
295     {
296       \_siunitx_unit_parse_add:nnnn { power }
297       {
298         \int_eval:n
299           { \l__siunitx_unit_position_int \bool_if:NT #3 { + 1 } }
300       }
301       {#1} {#2}
302     }
303   {
304     \msg_error:nnxx { siunitx }
305       { unit / part-before-unit } { power } { \token_to_str:N #1 }
306   }
307 }
308 \cs_new_protected:Npn \_siunitx_unit_parse_qualifier:nn #1#2
309 {
310   \tl_set:Nx \l__siunitx_unit_tmp_tl
311     { unit- \int_use:N \l__siunitx_unit_position_int }
312   \prop_if_in:NvTF \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
313     {
314       \_siunitx_unit_parse_add:nnnn { qualifier }
315       { \int_use:N \l__siunitx_unit_position_int } {#1} {#2}
316     }
317     {
318       \msg_error:nnnn { siunitx }
319       { unit / part-before-unit } { qualifier } { \token_to_str:N #1 }
320     }
321 }

```

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

```

322 \cs_new_protected:Npn \__siunitx_unit_parse_special:n #1
323 {
324   \__siunitx_unit_parse_add:nnnn { special }
325   { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
326   {#1} {#1}
327 }

```

(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.

```

328 \cs_new_protected:Npn \__siunitx_unit_parse_unit:Nn #1#2
329 {
330   \int_incr:N \l__siunitx_unit_position_int
331   \__siunitx_unit_parse_add:nnnn { unit }
332   { \int_use:N \l__siunitx_unit_position_int }
333   {#1} {#2}
334   \bool_if:NT \l__siunitx_unit_per_bool
335   {
336     \__siunitx_unit_parse_add:nnnn { per }
337     { \int_use:N \l__siunitx_unit_position_int }
338     { \per } { true }
339   }
340 }

```

(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.

```

341 \cs_new_protected:Npn \__siunitx_unit_parse_per:
342 {
343   \bool_if:NTF \l__siunitx_unit_sticky_per_bool
344   {
345     \bool_set_true:N \l__siunitx_unit_per_bool
346     \cs_set_protected:Npn \per
347     { \msg_error:nn { siunitx } { unit / duplicate-sticky-per } }
348   }
349   {
350     \__siunitx_unit_parse_add:nnnn
351     { per } { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
352     { \per } { true }
353   }
354 }

```

(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.

```

355 \cs_new_protected:Npn \_siunitx_unit_parse_finalise:n #1
356 {
357   \tl_set:Nx \l__siunitx_unit_tmp_tl { per- #1 }
358   \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
359   {
360     \prop_remove:NV \l__siunitx_unit_parsed_prop
361       \l__siunitx_unit_tmp_tl
362     \tl_set:Nx \l__siunitx_unit_tmp_tl { power- #1 }
363     \prop_get:NVNTF
364       \l__siunitx_unit_parsed_prop
365       \l__siunitx_unit_tmp_tl
366       \l__siunitx_unit_part_tl
367     {
368       \tl_set:Nx \l__siunitx_unit_part_tl
369         { \fp_eval:n { \l__siunitx_unit_part_tl * -1 } }
370       \fp_compare:nNnTF \l__siunitx_unit_part_tl = 1
371       {
372         \prop_remove:NV \l__siunitx_unit_parsed_prop
373           \l__siunitx_unit_tmp_tl
374       }
375       {
376         \prop_put:NVV \l__siunitx_unit_parsed_prop
377           \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
378       }
379     }
380     {
381       \prop_put:NVn \l__siunitx_unit_parsed_prop
382         \l__siunitx_unit_tmp_tl { -1 }
383     }
384   }
385 }

```

(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.

```

386 \cs_new_protected:Npn \_siunitx_unit_parse_finalise:
387 {
388   \clist_map_inline:nn { per , power , prefix }
389   {
390     \tl_set:Nx \l__siunitx_unit_tmp_tl
391       { ##1 - \int_eval:n { \l__siunitx_unit_position_int + 1 } }
392     \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
393     { \msg_error:nnn { siunitx } { unit / dangling-part } { ##1 } }
394   }
395 }

```

(End definition for `_siunitx_unit_parse_finalise:`.)

4.7 Formatting parsed units

Set up the options which apply to formatting.

```

\l_siunitx_unit_denominator_bracket_bool
\l_siunitx_unit_fraction_function_tl
\l_siunitx_unit_bracket_close_tl
\l_siunitx_unit_bracket_open_tl
\l__siunitx_unit_parse_bool
\l_siunitx_unit_per_symbol_tl
\l_siunitx_unit_qualifier_mode_tl
\l_siunitx_unit_qualifier_phrase_tl
396 \keys_define:nn { siunitx }
397 {
398   bracket-denominator .bool_set:N =
399     \l_siunitx_unit_denominator_bracket_bool ,
400   fraction-command .tl_set:N =
401     \l_siunitx_unit_fraction_function_tl ,
402   parse-units .bool_set:N =
403     \l_siunitx_unit_parse_bool ,
404   per-mode .choice: ,
405   per-mode / fraction .code:n =
406     {
407       \bool_set_false:N \l_siunitx_unit_autofrac_bool
408       \bool_set_false:N \l_siunitx_unit_per_symbol_bool
409       \bool_set_true:N \l_siunitx_unit_powers_positive_bool
410       \bool_set_true:N \l_siunitx_unit_two_part_bool
411     } ,
412   per-mode / power .code:n =
413     {
414       \bool_set_false:N \l_siunitx_unit_autofrac_bool
415       \bool_set_false:N \l_siunitx_unit_per_symbol_bool
416       \bool_set_false:N \l_siunitx_unit_powers_positive_bool
417       \bool_set_false:N \l_siunitx_unit_two_part_bool
418     } ,
419   per-mode / power-positive-first .code:n =
420     {
421       \bool_set_false:N \l_siunitx_unit_autofrac_bool
422       \bool_set_false:N \l_siunitx_unit_per_symbol_bool
423       \bool_set_false:N \l_siunitx_unit_powers_positive_bool
424       \bool_set_true:N \l_siunitx_unit_two_part_bool
425     } ,
426   per-mode / repeated-symbol .code:n =
427     {
428       \bool_set_false:N \l_siunitx_unit_autofrac_bool
429       \bool_set_true:N \l_siunitx_unit_per_symbol_bool
430       \bool_set_true:N \l_siunitx_unit_powers_positive_bool
431       \bool_set_false:N \l_siunitx_unit_two_part_bool
432     } ,
433   per-mode / symbol .code:n =
434     {
435       \bool_set_false:N \l_siunitx_unit_autofrac_bool
436       \bool_set_true:N \l_siunitx_unit_per_symbol_bool
437       \bool_set_true:N \l_siunitx_unit_powers_positive_bool
438       \bool_set_true:N \l_siunitx_unit_two_part_bool
439     } ,
440   per-mode / symbol-or-fraction .code:n =
441     {
442       \bool_set_true:N \l_siunitx_unit_autofrac_bool
443       \bool_set_true:N \l_siunitx_unit_per_symbol_bool
444       \bool_set_true:N \l_siunitx_unit_powers_positive_bool
445       \bool_set_true:N \l_siunitx_unit_two_part_bool
446     } ,

```

```

447   per-symbol .tl_set:N =
448     \l__siunitx_unit_per_symbol_tl ,
449   qualifier-mode .choice: ,
450   qualifier-mode / bracket .code:n =
451     { \tl_set:Nn \l__siunitx_unit_qualifier_mode_tl { bracket } } ,
452   qualifier-mode / combine .code:n =
453     { \tl_set:Nn \l__siunitx_unit_qualifier_mode_tl { combine } } ,
454   qualifier-mode / phrase .code:n =
455     { \tl_set:Nn \l__siunitx_unit_qualifier_mode_tl { phrase } } ,
456   qualifier-mode / subscript .code:n =
457     { \tl_set:Nn \l__siunitx_unit_qualifier_mode_tl { subscript } } ,
458   qualifier-phrase .tl_set:N =
459     \l__siunitx_unit_qualifier_phrase_tl ,
460   unit-close-bracket .tl_set:N =
461     \l__siunitx_unit_bracket_close_tl ,
462   unit-open-bracket .tl_set:N =
463     \l__siunitx_unit_bracket_open_tl
464 }

```

(End definition for `\l__siunitx_unit_denominator_bracket_bool` and others.)

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

```
465 \bool_new:N \l__siunitx_unit_bracket_bool
```

(End definition for `\l__siunitx_unit_bracket_bool`.)

`\l__siunitx_unit_font_bool` A flag to control when font wrapping is applied to the output.

```
466 \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`

`\l__siunitx_unit_per_symbol_bool`

`\l__siunitx_unit_two_part_bool`

```

467 \bool_new:N \l__siunitx_unit_autofrac_bool
468 \bool_new:N \l__siunitx_unit_per_symbol_bool
469 \bool_new:N \l__siunitx_unit_powers_positive_bool
470 \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).

```
471 \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.

```
472 \tl_new:N \l__siunitx_unit_qualifier_mode_tl
```

(End definition for `\l__siunitx_unit_qualifier_mode_tl`.)

`\l__siunitx_unit_prefix_power_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).

```
473 \bool_new:N \l__siunitx_unit_prefix_power_bool
```

(End definition for \l__siunitx_unit_prefix_power_bool.)

\l__siunitx_unit_prefix_fp When converting prefixes to powers, the calculations are done as an fp.

474 \fp_new:N \l__siunitx_unit_prefix_fp

(End definition for \l__siunitx_unit_prefix_fp.)

\l__siunitx_unit_current_tl Building up the (partial) formatted unit requires some token list storage. Each part of
\l__siunitx_unit_part_tl 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.

475 \tl_new:N \l__siunitx_unit_current_tl

476 \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).

477 \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.

478 \int_new:N \l__siunitx_unit_total_int

(End definition for \l__siunitx_unit_total_int.)

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

479 \cs_new_protected:Npn _siunitx_unit_format_parsed:

480 {

481 \int_set_eq:NN \l__siunitx_unit_total_int \l__siunitx_unit_position_int

482 \tl_clear:N \l__siunitx_unit_denominator_tl

483 \tl_clear:N \l__siunitx_unit_formatted_tl

484 \fp_zero:N \l__siunitx_unit_prefix_fp

485 \int_zero:N \l__siunitx_unit_position_int

486 \int_do_while:nNnn

487 \l__siunitx_unit_position_int < \l__siunitx_unit_total_int

488 {

489 \bool_set_false:N \l__siunitx_unit_bracket_bool

490 \tl_clear:N \l__siunitx_unit_current_tl

491 \bool_set_false:N \l__siunitx_unit_font_bool

492 \bool_set_true:N \l__siunitx_unit_numerator_bool

493 \int_incr:N \l__siunitx_unit_position_int

494 \clist_map_inline:nn { prefix , unit , qualifier , power , special }

495 { _siunitx_unit_format_parsed_aux:n {##1} }

496 _siunitx_unit_format_output:

497 }

498 _siunitx_unit_format_finalise:

499 }

500 \cs_new_protected:Npn _siunitx_unit_format_parsed_aux:n #1

501 {

502 \tl_set:Nx \l__siunitx_unit_tmp_tl

503 { #1 - \int_use:N \l__siunitx_unit_position_int }


```

504 \prop_get:NVNT \l__siunitx_unit_parsed_prop
505 \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
506 { \use:c { __siunitx_unit_format_ #1 : } }
507 }

```

(End definition for `__siunitx_unit_format_parsed:` and `__siunitx_unit_format_parsed_aux:n.`)

`__siunitx_unit_format_bracket:N` A quick utility function which wraps up a token list variable in brackets if they are required.

```

508 \cs_new:Npn \__siunitx_unit_format_bracket:N #1
509 {
510   \bool_if:NTF \l__siunitx_unit_bracket_bool
511   {
512     \exp_not:V \l__siunitx_unit_bracket_open_tl
513     \exp_not:V #1
514     \exp_not:V \l__siunitx_unit_bracket_close_tl
515   }
516   { \exp_not:V #1 }
517 }

```

(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.

```

\__siunitx_unit_format_power_aux:wTF
\__siunitx_unit_format_power_positive:
\__siunitx_unit_format_power_negative:
\__siunitx_unit_format_power_negative_aux:w
\__siunitx_unit_format_power_superscript:
518 \cs_new_protected:Npn \__siunitx_unit_format_power:
519 {
520   \__siunitx_unit_format_font:
521   \exp_after:wN \__siunitx_unit_format_power_aux:wTF
522   \l__siunitx_unit_part_tl - \q_stop
523   { \__siunitx_unit_format_power_negative: }
524   { \__siunitx_unit_format_power_positive: }
525 }
526 \cs_new:Npn \__siunitx_unit_format_power_aux:wTF #1 - #2 \q_stop
527 { \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 $\neq 1$).

```

528 \cs_new_protected:Npn \__siunitx_unit_format_power_positive:
529 { \__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 `~` then ensure that the result is not the trivial power 1. Assuming all is well, addition to the current unit combination goes ahead.

```

530 \cs_new_protected:Npn \__siunitx_unit_format_power_negative:
531 {
532   \bool_set_false:N \l__siunitx_unit_numerator_bool
533   \bool_if:NTF \l__siunitx_unit_powers_positive_bool
534   {
535     \tl_set:Nx \l__siunitx_unit_part_tl
536     {
537       \exp_after:wN \__siunitx_unit_format_power_negative_aux:w
538       \l__siunitx_unit_part_tl \q_stop

```

```

539     }
540     \str_if_eq_x:nnF { \exp_not:V \l__siunitx_unit_part_tl } { 1 }
541     { \__siunitx_unit_format_power_superscript: }
542   }
543   { \__siunitx_unit_format_power_superscript: }
544 }
545 \cs_new:Npn \__siunitx_unit_format_power_negative_aux:w - #1 \q_stop
546 { \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.

```

547 \cs_new_protected:Npn \__siunitx_unit_format_power_superscript:
548 {
549   \tl_set:Nx \l__siunitx_unit_current_tl
550   {
551     \__siunitx_unit_format_bracket:N \l__siunitx_unit_current_tl
552     ^ { \exp_not:V \l__siunitx_unit_part_tl }
553   }
554   \bool_set_false:N \l__siunitx_unit_bracket_bool
555 }

```

(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.

```

556 \cs_new_protected:Npn \__siunitx_unit_format_prefix:
557 {
558   \bool_if:NTF \l__siunitx_unit_prefix_power_bool
559   { \__siunitx_unit_format_prefix_power: }
560   { \__siunitx_unit_format_prefix_symbol: }
561 }
562 \cs_new_protected:Npn \__siunitx_unit_format_prefix_power:
563 {
564   \prop_get:NVNTF \l__siunitx_unit_prefixes_forward_prop
565   \l__siunitx_unit_part_tl \l__siunitx_unit_part_tl
566   {
567     \tl_set:Nx \l__siunitx_unit_tmp_tl
568     { power- \int_use:N \l__siunitx_unit_position_int }
569     \prop_get:NVNF \l__siunitx_unit_parsed_prop
570     \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
571     { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
572     \fp_add:Nn \l__siunitx_unit_prefix_fp
573     { \l__siunitx_unit_tmp_tl * \l__siunitx_unit_part_tl }
574   }
575   { \__siunitx_unit_format_prefix_symbol: }
576 }
577 \cs_new_protected:Npn \__siunitx_unit_format_prefix_symbol:
578 { \tl_set_eq:NN \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl }

```

(End definition for `__siunitx_unit_format_prefix:`, `__siunitx_unit_format_prefix_power:`, and `__siunitx_unit_format_prefix_symbol:`.)

`_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. In the case that a linking phrase is in use, the addition of spaces means that the unit may end up ambiguous, and brackets are therefore required *if* there is a power. Notice that when the qualifier is just treated as “text”, the auxiliary is actually a no-op.

```

579 \cs_new_protected:Npn \_siunitx_unit_format_qualifier:
580 {
581   \use:c
582   {
583     \_siunitx_unit_format_qualifier_
584     \l__siunitx_unit_qualifier_mode_tl :
585   }
586   \tl_put_right:NV \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
587 }
588 \cs_new_protected:Npn \_siunitx_unit_format_qualifier_bracket:
589 {
590   \_siunitx_unit_format_font:
591   \tl_set:Nx \l__siunitx_unit_part_tl
592   {
593     \exp_not:V \l__siunitx_unit_bracket_open_tl
594     \exp_not:N \mathrm
595     { \exp_not:V \l__siunitx_unit_part_tl }
596     \exp_not:V \l__siunitx_unit_bracket_close_tl
597   }
598 }
599 \cs_new_protected:Npn \_siunitx_unit_format_qualifier_combine: { }
600 \cs_new_protected:Npn \_siunitx_unit_format_qualifier_phrase:
601 {
602   \_siunitx_unit_format_font:
603   \tl_set:Nx \l__siunitx_unit_part_tl
604   {
605     \exp_not:V \l__siunitx_unit_qualifier_phrase_tl
606     \exp_not:N \mathrm
607     { \exp_not:V \l__siunitx_unit_part_tl }
608   }
609 }
610 \cs_new_protected:Npn \_siunitx_unit_format_qualifier_subscript:
611 {
612   \_siunitx_unit_format_font:
613   \tl_set:Nx \l__siunitx_unit_part_tl
614   {
615     \c__siunitx_unit_math_subscript_tl
616     {
617       \exp_not:N \mathrm
618       { \exp_not:V \l__siunitx_unit_part_tl }
619     }
620   }
621 }

```

(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.

```

622 \cs_new_protected:Npn \_siunitx_unit_format_special:

```

```

623 {
624   \tl_set:Nx \l__siunitx_unit_current_tl
625   {
626     \exp_not:V \l__siunitx_unit_part_tl
627     { \exp_not:V \l__siunitx_unit_current_tl }
628   }
629 }

```

(End definition for `__siunitx_unit_format_special:`)

`__siunitx_unit_format_unit:` A very simple task: add the unit to the output currently being constructed.

```

630 \cs_new_protected:Npn \__siunitx_unit_format_unit:
631 {
632   \tl_put_right:NV
633   \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
634 }

```

(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:
\__siunitx_unit_format_output_denominator:
\__siunitx_unit_format_output_aux:nn
\__siunitx_unit_format_output_aux:nV
\__siunitx_unit_format_output_aux:nv
635 \cs_new_protected:Npn \__siunitx_unit_format_output:
636 {
637   \__siunitx_unit_format_font:
638   \bool_set_false:N \l__siunitx_unit_bracket_bool
639   \use:c
640   {
641     __siunitx_unit_format_output_
642     \bool_if:NTF \l__siunitx_unit_numerator_bool
643     { aux: }
644     { denominator: }
645   }
646 }
647 \cs_new_protected:Npn \__siunitx_unit_format_output_aux:
648 {
649   \__siunitx_unit_format_output_aux:nV { formatted }
650   \l__siunitx_unit_product_tl
651 }

```

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.

```

652 \cs_new_protected:Npn \__siunitx_unit_format_output_denominator:
653 {
654   \bool_if:NTF \l__siunitx_unit_two_part_bool
655   {

```

```

656     \bool_lazy_and:nnT
657       { \l__siunitx_unit_denominator_bracket_bool }
658       { ! \tl_if_empty_p:N \l__siunitx_unit_denominator_tl }
659       { \bool_set_true:N \l__siunitx_unit_bracket_bool }
660     \__siunitx_unit_format_output_aux:nV { denominator }
661     \l__siunitx_unit_product_tl
662   }
663   {
664     \bool_lazy_and:nnT
665       { \l__siunitx_unit_per_symbol_bool }
666       { \tl_if_empty_p:N \l__siunitx_unit_formatted_tl }
667       { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
668     \__siunitx_unit_format_output_aux:nv { formatted }
669     {
670       l__siunitx_unit_
671       \bool_if:NTF \l__siunitx_unit_per_symbol_bool
672         { per_symbol }
673         { product }
674       _tl
675     }
676   }
677 }
678 \cs_new_protected:Npn \__siunitx_unit_format_output_aux:nn #1#2
679 {
680   \tl_set:cx { l__siunitx_unit_ #1 _tl }
681   {
682     \exp_not:v { l__siunitx_unit_ #1 _tl }
683     \tl_if_empty:cF { l__siunitx_unit_ #1 _tl }
684     { \exp_not:n {#2} }
685     \exp_not:V \l__siunitx_unit_current_tl
686   }
687 }
688 \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.

```

689 \cs_new_protected:Npn \__siunitx_unit_format_font:
690 {
691   \bool_if:NF \l__siunitx_unit_font_bool
692   {
693     \tl_set:Nx \l__siunitx_unit_current_tl
694     {
695       \exp_not:N \mathrm
696       { \exp_not:V \l__siunitx_unit_current_tl }
697     }
698     \bool_set_true:N \l__siunitx_unit_font_bool
699   }
700 }

```

(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:

```

701 \cs_new_protected:Npn \__siunitx_unit_format_finalise:
702 {
703   \tl_if_empty:NF \l__siunitx_unit_denominator_tl
704   {
705     \bool_if:NTF \l__siunitx_unit_powers_positive_bool
706     { \__siunitx_unit_format_finalise_fractional: }
707     { \__siunitx_unit_format_finalise_power: }
708   }
709 }

```

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.

```

710 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fractional:
711 {
712   \tl_if_empty:NT \l__siunitx_unit_formatted_tl
713   { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
714   \bool_if:NTF \l__siunitx_unit_autofrac_bool
715   { \__siunitx_unit_format_finalise_autofrac: }
716   {
717     \bool_if:NTF \l__siunitx_unit_per_symbol_bool
718     { \__siunitx_unit_format_finalise_symbol: }
719     { \__siunitx_unit_format_finalise_fraction: }
720   }
721 }

```

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.

```

722 \cs_new_protected:Npn \__siunitx_unit_format_finalise_autofrac:
723 {
724   \group_begin:
725   \__siunitx_unit_format_finalise_fraction:
726   \exp_args:NNNV \group_end:
727   \tl_set:Nn \l__siunitx_unit_tmp_tl \l__siunitx_unit_formatted_tl
728   \__siunitx_unit_format_finalise_symbol:
729   \tl_set:Nx \l__siunitx_unit_formatted_tl
730   {
731     \mathchoice
732     { \exp_not:V \l__siunitx_unit_tmp_tl }
733     { \exp_not:V \l__siunitx_unit_formatted_tl }
734     { \exp_not:V \l__siunitx_unit_formatted_tl }
735     { \exp_not:V \l__siunitx_unit_formatted_tl }
736   }
737 }

```

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

```

738 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fraction:
739 {
740   \tl_set:Nx \l__siunitx_unit_formatted_tl
741   {
742     \exp_not:V \l__siunitx_unit_fraction_function_tl

```

```

743         { \exp_not:V \l__siunitx_unit_formatted_tl }
744         { \exp_not:V \l__siunitx_unit_denominator_tl }
745     }
746 }
747 \cs_new_protected:Npn \__siunitx_unit_format_finalise_symbol:
748 {
749     \tl_set:Nx \l__siunitx_unit_formatted_tl
750     {
751         \exp_not:V \l__siunitx_unit_formatted_tl
752         \exp_not:V \l__siunitx_unit_per_symbol_tl
753         \__siunitx_unit_format_bracket:N \l__siunitx_unit_denominator_tl
754     }
755 }

```

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.

```

756 \cs_new_protected:Npn \__siunitx_unit_format_finalise_power:
757 {
758     \tl_if_empty:NTF \l__siunitx_unit_formatted_tl
759     {
760         \tl_set_eq:NN
761         \l__siunitx_unit_formatted_tl
762         \l__siunitx_unit_denominator_tl
763     }
764     {
765         \tl_set:Nx \l__siunitx_unit_formatted_tl
766         {
767             \exp_not:V \l__siunitx_unit_formatted_tl
768             \exp_not:V \l__siunitx_unit_product_tl
769             \exp_not:V \l__siunitx_unit_denominator_tl
770         }
771     }
772 }

```

(End definition for __siunitx_unit_format_finalise: and others.)

4.8 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
\metre \meter.
\meter 773 \siunitx_declare_unit:Nn \kilogram { \kilo \gram }
\mole 774 \siunitx_declare_unit:Nn \metre { m }
\kelvin 775 \siunitx_declare_unit:Nn \meter { \metre }
\candela 776 \siunitx_declare_unit:Nn \mole { mol }
\second 777 \siunitx_declare_unit:Nn \second { s }
\ampere 778 \siunitx_declare_unit:Nn \ampere { A }
779 \siunitx_declare_unit:Nn \kelvin { K }
780 \siunitx_declare_unit:Nn \candela { cd }

```

(End definition for \kilogram and others. These functions are documented on page 57.)

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

```
781 \siunitx_declare_unit:Nn \gram { g }
```

(End definition for \gram. This function is documented on page 57.)

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

```
782 \siunitx_declare_prefix:Nnn \yocto { y } { -24 }
783 \siunitx_declare_prefix:Nnn \zepto { z } { -21 }
784 \siunitx_declare_prefix:Nnn \atto { a } { -18 }
785 \siunitx_declare_prefix:Nnn \femto { f } { -15 }
786 \siunitx_declare_prefix:Nnn \pico { p } { -12 }
787 \siunitx_declare_prefix:Nnn \nano { n } { -9 }
788 \siunitx_declare_prefix:Nnn \micro { [micro] } { -6 }
789 \siunitx_declare_prefix:Nnn \milli { m } { -3 }
790 \siunitx_declare_prefix:Nnn \centi { c } { -2 }
791 \siunitx_declare_prefix:Nnn \deci { d } { -1 }
```

(End definition for \yocto and others. These functions are documented on page 58.)

\deca Now the large ones.

```
792 \siunitx_declare_prefix:Nnn \deca { da } { 1 }
793 \siunitx_declare_prefix:Nnn \deka { da } { 1 }
794 \siunitx_declare_prefix:Nnn \hecto { h } { 2 }
795 \siunitx_declare_prefix:Nnn \kilo { k } { 3 }
796 \siunitx_declare_prefix:Nnn \mega { M } { 6 }
797 \siunitx_declare_prefix:Nnn \giga { G } { 9 }
798 \siunitx_declare_prefix:Nnn \tera { T } { 12 }
799 \siunitx_declare_prefix:Nnn \peta { P } { 15 }
800 \siunitx_declare_prefix:Nnn \exa { E } { 18 }
801 \siunitx_declare_prefix:Nnn \zetta { Z } { 21 }
802 \siunitx_declare_prefix:Nnn \yotta { Y } { 24 }
```

(End definition for \deca and others. These functions are documented on page 58.)

\becquerel Named derived units: first half of alphabet.

```
803 \siunitx_declare_unit:Nn \becquerel { Bq }
804 \siunitx_declare_unit:Nn \degreeCelsius
805 { \ensuremath { { } ^ { \circ } } } \kern -\scriptspace C }
806 \siunitx_declare_unit:Nn \coulomb { C }
807 \siunitx_declare_unit:Nn \farad { F }
808 \siunitx_declare_unit:Nn \gray { Gy }
809 \siunitx_declare_unit:Nn \hertz { Hz }
810 \siunitx_declare_unit:Nn \henry { H }
811 \siunitx_declare_unit:Nn \joule { J }
812 \siunitx_declare_unit:Nn \katal { kat }
813 \siunitx_declare_unit:Nn \lumen { lm }
814 \siunitx_declare_unit:Nn \lux { lx }
```

(End definition for \becquerel and others. These functions are documented on page 58.)

\newton Named derived units: second half of alphabet.

```
815 \siunitx_declare_unit:Nn \newton { N }
816 \siunitx_declare_unit:Nn \ohm { \ensuremath { \Omega } }
817 \siunitx_declare_unit:Nn \pascal { Pa }
818 \siunitx_declare_unit:Nn \radian { rad }
\siemens
\sievert
\steradian
\tesla
\volt
\watt
\weber
```



```

819 \siunitx_declare_unit:Nn \siemens { S }
820 \siunitx_declare_unit:Nn \sievert { Sv }
821 \siunitx_declare_unit:Nn \steradian { sr }
822 \siunitx_declare_unit:Nn \tesla { T }
823 \siunitx_declare_unit:Nn \volt { V }
824 \siunitx_declare_unit:Nn \watt { W }
825 \siunitx_declare_unit:Nn \weber { Wb }

```

(End definition for `\newton` and others. These functions are documented on page 58.)

```

\day Non-SI, but accepted for general use. Once again there are two spellings, here for litre
\hectare and with different output in this case.
\hour 826 \siunitx_declare_unit:Nn \day { d }
\litre 827 \siunitx_declare_unit:Nn \hectare { ha }
\liter 828 \siunitx_declare_unit:Nn \hour { h }
\minute 829 \siunitx_declare_unit:Nn \litre { L }
\tonne 830 \siunitx_declare_unit:Nn \liter { \litre }
831 \siunitx_declare_unit:Nn \minute { min }
832 \siunitx_declare_unit:Nn \tonne { t }

```

(End definition for `\day` and others. These functions are documented on page 58.)

```

\arcminute Arc units: again, non-SI, but accepted for general use.
\arcsecond 833 \siunitx_declare_unit:Nn \arcminute { \ensuremath { ^ { \circ } } }
\degree 834 \siunitx_declare_unit:Nn \arcsecond { \ensuremath { ^ { \circ } } }
835 \siunitx_declare_unit:Nn \degree { \ensuremath { ^ { \circ } } }

```

(End definition for `\arcminute`, `\arcsecond`, and `\degree`. These functions are documented on page 59.)

```

\astronomicalunit A few units based on physical measurements exist: these ones are accepted for use with
\atomicmassunit the International System.
\dalton 836 \siunitx_declare_unit:Nn \astronomicalunit { au }
\electronvolt 837 \siunitx_declare_unit:Nn \atomicmassunit { u }
838 \siunitx_declare_unit:Nn \dalton { Da }
839 \siunitx_declare_unit:Nn \electronvolt { eV }

```

(End definition for `\astronomicalunit` and others. These functions are documented on page 59.)

```

\nuaction Natural units based on physical constants.
\numass 840 \siunitx_declare_unit:Nn \nuaction { \mathit { \hbar } }
\nuspeed 841 \siunitx_declare_unit:Nx \numass
\nutime 842 {
843   \exp_not:N \ensuremath
844   {
845     \exp_not:N \mathit { m }
846     \c__siunitx_unit_math_subscript_tl { \exp_not:N \mathrm { e } }
847   }
848 }
849 \siunitx_declare_unit:Nx \nuspeed
850 {
851   \exp_not:N \ensuremath
852   { \exp_not:N \mathit { c } \c__siunitx_unit_math_subscript_tl { 0 } }
853 }
854 \siunitx_declare_unit:Nn \nutime
855 { \numass \per \numass \per \nuspeed \squared }

```

(End definition for `\nuaction` and others. These functions are documented on page 59.)

```

\auaction Atomic units based on physical constants.
\aucharge 856 \siunitx_declare_unit:Nn \auaction { \ensuremath { \mathit { \hbar } } } }
\auenergy 857 \siunitx_declare_unit:Nn \aucharge { \ensuremath { \mathit { e } } } }
\aulength 858 \siunitx_declare_unit:Nx \auenergy
\aumass    859 {
\autime    860   \exp_not:N \ensuremath
\bohr      861   {
\hartree   862     \exp_not:N \mathit { E }
            863     \c__siunitx_unit_math_subscript_tl { \exp_not:N \mathrm { h } } }
            864   }
            865 }
            866 \siunitx_declare_unit:Nx \aulength
            867 {
            868   \exp_not:N \ensuremath
            869   { \exp_not:N \mathit { a } } \c__siunitx_unit_math_subscript_tl { 0 } }
            870 }
            871 \siunitx_declare_unit:Nx \aumass
            872 {
            873   \exp_not:N \ensuremath
            874   {
            875     \exp_not:N \mathit { m }
            876     \c__siunitx_unit_math_subscript_tl { \exp_not:N \mathrm { e } } }
            877   }
            878 }
            879 \siunitx_declare_unit:Nn \autime { \auaction \per \auenergy }
            880 \siunitx_declare_unit:Nn \bohr   { \aulength }
            881 \siunitx_declare_unit:Nn \hartree { \auenergy }

```

(End definition for `\auaction` and others. These functions are documented on page 59.)

```

\angstrom There are then some day-to-day units which are accepted for use with SI, but are not
\bar       part of the official specification.
\barn      882 \siunitx_declare_unit:Nn \angstrom { \mbox { \AA } }
\bel       883 \siunitx_declare_unit:Nn \bar { bar }
\decibel   884 \siunitx_declare_unit:Nn \barn { b }
\knot      885 \siunitx_declare_unit:Nn \bel { B }
\millimetremercury 886 \siunitx_declare_unit:Nn \decibel { \deci \bel }
\nauticalmile 887 \siunitx_declare_unit:Nn \knot { kn }
\neper     888 \siunitx_declare_unit:Nn \millimetremercury { mmHg }
           889 \siunitx_declare_unit:Nn \nauticalmile { M }
           890 \siunitx_declare_unit:Nn \neper { Np }

```

(End definition for `\angstrom` and others. These functions are documented on page 59.)

```

\dyne CGS units: similar to the set immediately above, these may be used for specific applica-
\erg   tions.
\gal   891 \siunitx_declare_unit:Nn \dyne { dyn }
\gauss 892 \siunitx_declare_unit:Nn \erg { erg }
\maxwell 893 \siunitx_declare_unit:Nn \gal { Gal }
\oersted 894 \siunitx_declare_unit:Nn \gauss { G }
\phot    895 \siunitx_declare_unit:Nn \maxwell { Mx }
\poise   896 \siunitx_declare_unit:Nn \oersted { Oe }
\stilb
\stokes

```

```

897 \siunitx_declare_unit:Nn \phot    { ph }
898 \siunitx_declare_unit:Nn \poise   { P }
899 \siunitx_declare_unit:Nn \stilb   { sb }
900 \siunitx_declare_unit:Nn \stokes  { St }

```

(End definition for `\dyne` and others. These functions are documented on page 59.)

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

```

901 \siunitx_declare_unit:Nn \percent { \char "25 ~ }

```

(End definition for `\percent`. This function is documented on page 59.)

\square Basic powers.

```

\squared 902 \siunitx_declare_power:NNn \square \squared { 2 }
\cubic    903 \siunitx_declare_power:NNn \cubic  \cubed  { 3 }
\cubed

```

(End definition for `\square` and others. These functions are documented on page 59.)

4.9 Messages

```

904 \msg_new:nnnn { siunitx } { unit / dangling-part }
905 { Found~#1~part~with~no~unit. }
906 {
907   Each~#1~part~must~be~associated~with~a~unit:~a~#1~part~was~found~
908   but~no~following~unit~was~given.
909 }
910 \msg_new:nnnn { siunitx } { unit / duplicate-part }
911 { Duplicate~#1~part:~#2. }
912 {
913   Each~unit~may~have~only~one~#1:\\
914   the~additional~#1~part~'~#2'~will~be~ignored.
915 }
916 \msg_new:nnnn { siunitx } { unit / duplicate-sticky-per }
917 { Duplicate~\token_to_str:N \per. }
918 {
919   When~the~'sticky-per'~option~is~active,~only~one~
920   \token_to_str:N \per \ may~appear~in~a~unit.
921 }
922 \msg_new:nnnn { siunitx } { unit / part-before-unit }
923 { Found~#1~part~before~first~unit:~#2. }
924 {
925   The~#1~part~'~#2'~must~follow~after~a~unit:~
926   it~cannot~appear~before~any~units~and~will~therefore~be~ignored.
927 }

```

4.10 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.

```

928 \keys_set:nn { siunitx }
929 {
930   bracket-denominator = true      ,
931   fraction-command     = \frac    ,
932   parse-units          = true      ,
933   per-mode             = power     ,

```

```

934     per-symbol          = / ,
935     qualifier-mode      = subscript ,
936     qualifier-phrase    = \ ,
937     sticky-per          = false ,
938     unit-close-bracket  = ) , % (
939     unit-open-bracket   = ( , % )
940     unit-product        = \,
941 }
942 </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/publications/si-brochure/section2-1.html>.
- [3] *Units with special names and symbols; units that incorporate special names and symbols*, <https://www.bipm.org/en/publications/si-brochure/section2-2-2.html>.
- [4] *SI Prefixes*, <https://www.bipm.org/en/publications/si-brochure/chapter3.html>.
- [5] *Stating values of dimensionless quantities, or quantities of dimension one*, <https://www.bipm.org/en/publications/si-brochure/section5-3-7.html>.
- [6] *Non-SI units accepted for use with the International System of Units*, <https://www.bipm.org/en/publications/si-brochure/table6.html>.
- [7] *Non-SI units whose values in SI units must be obtained experimentally*, <https://www.bipm.org/en/publications/si-brochure/table7.html>.
- [8] *Other non-SI units*, <https://www.bipm.org/en/publications/si-brochure/table8.html>.
- [9] *Non-SI units associated with the CGS and the CGS-Gaussian system of units*, <https://www.bipm.org/en/publications/si-brochure/table9.html>.

Part VII

siunitx-v1 – Version 1 compatibility

1 siunitx-v1 implementation

Start the DocStrip guards.

¹ `*cfg`

Identify the internal prefix (L^AT_EX3 DocStrip convention). In contrast to other parts of the bundle, the functions here may need to redefine those from various submodules.

² `\@@=siunitx`

³ `\./cfg`

Part VIII

siunitx-v2 – Version 2 compatibility

1 siunitx-v2 implementation

Start the DocStrip guards.

```
1 < *cfg >
```

Identify the internal prefix (L^AT_EX3 DocStrip convention). In contrast to other parts of the bundle, the functions here may need to redefine those from various submodules.

```
2 < @@=siunitx >
```

The old `s` column type is handled by using the functionality of `collcell`.

```
3 \RequirePackage { collcell }
4 \AtBeginDocument
5 {
6   \__siunitx_declare_column:Nnn s
7     { \collectcell \unit }
8     { \endcollectcell }
9 }
10 < /cfg >
```

Index

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

Symbols	
\,	55, 63, 940, 1054, 1061
\-	320
\\	913
_	8, 176, 187, 233, 292
\~	144
\sq	65, 89, 920, 936
A	
\AA	882
\ampere	57, <u>773</u>
\angstrom	59, <u>882</u>
\approx	1057
\arcminute	59, <u>833</u>
\arcsecond	59, <u>833</u>
\array	105
\arraycolsep	103
\astronomicalunit	59, <u>836</u>
\AtBeginDocument 4, 4, 5, 33, 57, 89, 133, 145, 188
\atomicmassunit	59, <u>836</u>
\atto	58, <u>782</u>
\auaction	59, <u>856</u>
\aucharge	59, <u>856</u>
\auenergy	59, <u>856</u>
\aulength	59, <u>856</u>
\aumass	59, <u>856</u>
\autime	59, <u>856</u>
B	
\bar	59, <u>882</u>
\barn	59, <u>882</u>
\bcolumn	94, 108
\becquerel	58, <u>803</u>
\bel	59, <u>882</u>
\bfsries	35
\bohr	59, <u>856</u>
\boldmath	36
bool commands:	
\bool_if:NTF	10, 31, 33, 79, 99, 110, 116, 119, 133, 148, 179, 205, 207, 209, 251, 299, 334, 343, 362, 393, 408, 431, 476, 477, 510, 521, 533, 538, 558, 642, 654, 671, 691, 705, 714, 717, 777, 785, 794, 837, 904, 978
\bool_lazy_all:NTF	817
\bool_lazy_and:nnTF	93, 112, 175, 425, 656, 664, 973, 1006
\bool_lazy_or:nnTF	162, 289, 437, 506, 969, 989
\bool_new:N	9, 10, 71, 251, 465, 466, 467, 468, 469, 470, 471, 473, 625, 737, 738
\bool_set_false:N	104, 148, 257, 258, 407, 408, 414, 415, 416, 417, 421, 422, 423, 428, 431, 435, 489, 491, 532, 554, 608, 638, 697, 701, 706, 707
\bool_set_true:N	14, 109, 256, 345, 409, 410, 424, 429, 430, 436, 437, 438, 442, 443, 444, 445, 492, 610, 659, 691, 692, 696, 698, 702, 1004
\c_false_bool	46, 96, 369, 443, 447, 452, 454, 467, 491, 497, 511, 515, 525, 544, 570
\c_true_bool	43, 93, 354, 369, 404, 441, 454, 464, 491, 497, 511
box commands:	
\box_new:N	11
bracket-denominator	60
C	
\cancel	55, 60, 65, <u>82</u>
\candela	57, <u>773</u>
\centi	58, <u>782</u>
\char	901
char commands:	
\char_generate:nn	8, 176, 187, 233, 292
\char_set_catcode_active:N	320
\char_set_catcode_active:n	144
\circ	805, 835
clist commands:	
\clist_map_function:nN	30, 35
\clist_map_inline:nn	388, 494
\collectcell	7
color	36
\coulomb	58, <u>803</u>
\cr	23, <u>47</u>
cs commands:	
\cs:w	638
\cs_end:	641
\cs_generate_variant:Nn	3, 3, 4, 5, 14, 78, 85, 128, 151, 158, 165, 184, 215, 257, 688, 834, 946, 960
\cs_if_eq:NNTF	281

<code>\cs_new:Npn</code>	62, 187, 194, 345, 508, 526, 545, 645, 647, 657, 668, 670, 672, 755, 757, 767, 773, 792, 798, 807, 815, 826, 835, 846, 855, 857, 859, 861, 867, 874, 879, 899, 921, 928, 947, 961, 966, 998
<code>\cs_new_eq:NN</code>	192
<code>\cs_new_protected:Npn</code>	8, 9, 14, 15, 19, 19, 20, 20, 22, 24, 26, 31, 37, 39, 42, 48, 54, 62, 68, 71, 73, 81, 86, 88, 89, 97, 97, 102, 107, 108, 112, 114, 117, 117, 123, 124, 129, 130, 130, 131, 140, 141, 142, 144, 146, 146, 151, 152, 159, 160, 167, 171, 173, 173, 175, 177, 185, 190, 192, 194, 197, 201, 202, 207, 211, 213, 214, 220, 229, 241, 243, 244, 247, 247, 252, 253, 257, 257, 258, 265, 265, 265, 270, 271, 275, 279, 279, 283, 285, 293, 296, 297, 308, 322, 327, 328, 332, 341, 350, 355, 358, 384, 386, 423, 435, 445, 450, 456, 462, 469, 479, 481, 500, 500, 518, 528, 530, 532, 547, 551, 556, 562, 574, 577, 579, 580, 587, 588, 592, 599, 600, 610, 622, 627, 630, 632, 635, 647, 652, 678, 689, 701, 710, 722, 738, 739, 747, 756
<code>\cs_new_protected:Npx</code>	145, 169, 185, 222, 285, 321
<code>\cs_set:Npn</code>	23, 25, 33, 95
<code>\cs_set_eq:NN</code>	116
<code>\cs_set_protected:Npn</code>	23, 61, 120, 137, 180, 195, 227, 259, 304, 346
<code>\cs_set_protected:Npx</code>	130
<code>\cs_undefine:N</code>	98
<code>\cubed</code>	60, <u>902</u>
<code>\cubic</code>	59, <u>902</u>
D	
<code>\dalton</code>	59, <u>836</u>
<code>\day</code>	58, <u>826</u>
<code>\deca</code>	58, <u>792</u>
<code>\deci</code>	58, <u>782</u> , 886
<code>\decibel</code>	59, <u>882</u>
<code>\DeclareSIpower</code>	<u>40</u>
<code>\DeclareSIPrefix</code>	<u>40</u>
<code>\DeclareSIQualifier</code>	<u>40</u>
<code>\DeclareSIUnit</code>	<u>40</u>
<code>\def</code>	101
<code>\degree</code>	59, <u>833</u>
<code>\degreeCelsius</code>	58, <u>803</u>
<code>\deka</code>	58, <u>792</u>
<code>\document</code>	12, <u>38</u>
<code>\dyne</code>	59, <u>891</u>
E	
<code>\ecolumn</code>	94, 108
<code>\electronvolt</code>	59, <u>836</u>
else commands:	
<code>\else:</code>	263
<code>\end</code>	47, 68
<code>\endcollectcell</code>	8
<code>\endinput</code>	13
<code>\ensuremath</code>	24, 35, 55, 130, 230, 289, 805, 816, 833, 834, 835, 840, 843, 851, 856, 857, 860, 868, 873
<code>\erg</code>	59, <u>891</u>
<code>\exa</code>	58, <u>792</u>
exp commands:	
<code>\exp_after:wN</code> 96, 97, 100, 104, 104, 122, 126, 128, 137, 163, 199, 262, 264, 340, 353, 354, 483, 521, 537, 589, 640, 756
<code>\exp_args:NNc</code>	123
<code>\exp_args:NNNV</code>	25, 167, 726
<code>\exp_args:NNV</code>	16, 276
<code>\exp_args:Nv</code>	75
<code>\exp_last_unbraced:Nn</code>	299
<code>\exp_not:N</code> 132, 134, 149, 150, 151, 152, 154, 172, 173, 174, 175, 177, 178, 180, 181, 188, 194, 195, 197, 201, 205, 207, 210, 213, 215, 217, 218, 219, 220, 222, 223, 224, 225, 225, 226, 227, 227, 228, 228, 228, 229, 229, 231, 231, 232, 232, 233, 234, 236, 237, 237, 238, 238, 239, 239, 239, 288, 289, 291, 293, 323, 324, 347, 566, 567, 594, 606, 617, 695, 801, 843, 845, 846, 851, 852, 860, 862, 863, 868, 869, 873, 875, 876, 980
<code>\exp_not:n</code>	119, 133, 148, 154, 155, 156, 173, 179, 196, 217, 222, 229, 229, 230, 230, 232, 237, 238, 239, 239, 272, 338, 339, 348, 430, 475, 512, 513, 514, 516, 540, 546, 552, 558, 567, 593, 595, 596, 605, 607, 618, 626, 627, 646, 655, 666, 675, 682, 684, 685, 696, 732, 733, 734, 735, 742, 743, 744, 751, 752, 767, 768, 769, 770, 771, 795, 796, 801, 810, 812, 828, 830, 831, 841, 844, 863, 869, 870, 883, 884, 888, 892, 906, 908, 914, 915, 916, 923, 924, 925, 926, 968, 976, 981, 983, 984, 986, 996, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032

`\expandafter` 125

F

`\fam` 27, 38, 150, 163, 164

`\farad` 58, 803

`\femto` 58, 782

`\fi` 111

fi commands:

`\fi:` 22, 28, 265

`\fontsize` 301

fp commands:

`\fp_add:Nn` 572

`\fp_compare:nNnTF` 370

`\fp_eval:n` 80, 369

`\fp_new:N` 4, 474

`\fp_set:Nn` 134

`\fp_use:N` 135

`\fp_zero:N` 118, 484

`\l_tmpa_fp` 56

`\frac` 55, 931

`\fraction-command` 61

G

`\gal` 59, 891

`\gauss` 59, 891

`\ge` 317, 1057

`\geq` 1057

`\gg` 316, 1057

`\giga` 58, 792

`\gram` 57, 773, 781

`\gray` 58, 803

group commands:

`\group_begin:` 13,
21, 22, 22, 59, 71, 80, 114, 143, 147,
171, 194, 216, 273, 287, 319, 724, 1003

`\c_group_begin_token` ... 33, 119, 222

`\group_end:` 16, 25, 25, 28,
30, 66, 75, 84, 137, 167, 172, 182,
198, 220, 276, 295, 326, 726, 1010, 1014

H

`\hartree` 59, 856

`\hbar` 55, 840, 856

hbox commands:

`\hbox_set:Nn` 22

`\hbox_to_wd:nn` 180

`\hectare` 58, 826

`\hecto` 58, 792

`\henry` 58, 803

`\hertz` 58, 803

`\highlight` 60, 65, 82

`\hour` 58, 826

`\hskip` 103

I

if commands:

`\if_false:` 22, 28

`\if_meaning:w` 261

`\ignorespaces` 17

int commands:

`\int_abs:n` 148, 941

`\int_case:nnTF` 150

`\int_compare:nNnTF`
..... 133, 155, 651, 661, 839, 930, 951

`\int_compare_p:nNn` 163, 164

`\int_const:Nn` 27

`\int_do_while:nNnn` 486

`\int_eval:n` 140,
298, 325, 351, 391, 851, 910, 933, 955

`\int_incr:N` 330, 493

`\int_mod:nn` 851

`\int_new:N` 5, 252, 478

`\int_set:Nn` 281

`\int_set_eq:NN` 481

`\int_step_inline:nnnn` 261

`\int_use:N`
283, 288, 311, 315, 332, 337, 503, 568

`\int_zero:N` 259, 485

J

`\joule` 58, 803

K

`\katal` 58, 803

`\kelvin` 57, 773

`\kern` 805

keys commands:

`\l_keys_choice_tl` 46, 68

`\keys_define:nn` 4, 28, 34, 97,
164, 204, 216, 232, 246, 396, 604, 678

`\keys_set:nn` 60,
72, 81, 87, 149, 268, 305, 928, 1044

`\kilo` 58, 70, 773, 792

`\kilogram` 57, 57, 58, 773

`\knot` 59, 882

L

`\le` 315, 1057

`\leavevmode` 58, 70, 79

`\leq` 1057

`\let` 104

`\liter` 58, 826

`\litre` 58, 826

`\ll` 314, 1057

`\lumen` 58, 803

`\lux` 58, 803

M

`\mathchoice` 55, 83, 731

<code>\quark_if_recursion_tail_stop:N</code> .	<code>\siunitx_declare_qualifier:Nn</code> . . .
. 132, 189, 243, 285, 306, 534 50, 56, 62
<code>\quark_if_recursion_tail_stop:n</code> .	<code>\siunitx_declare_unit:Nn</code> 54,
. 175, 187	57, 68, 773, 774, 775, 776, 777, 778,
<code>\quark_if_recursion_tail_stop-</code>	779, 780, 781, 803, 804, 806, 807,
do:Nn 277, 360, 386, 485, 502	808, 809, 810, 811, 812, 813, 814,
<code>\q_recursion_stop</code> . . 105, 165, 183,	815, 816, 817, 818, 819, 820, 821,
183, 195, 239, 272, 294, 302, 356,	822, 823, 824, 825, 826, 827, 828,
364, 486, 503, 552, 571, 575, 578, 581	829, 830, 831, 832, 833, 834, 835,
<code>\q_recursion_tail</code> 105, 165, 183, 183,	836, 837, 838, 839, 840, 841, 849,
195, 237, 272, 301, 356, 552, 571, 575	854, 856, 857, 858, 866, 871, 879,
<code>\q_stop</code> . . 86, 95, 97, 188, 195, 200,	880, 881, 882, 883, 884, 885, 886,
202, 205, 213, 231, 239, 252, 257,	887, 888, 889, 890, 891, 892, 893,
341, 345, 522, 526, 538, 545, 590, 592	894, 895, 896, 897, 898, 899, 900, 901
R	
<code>\radian</code> 58, 815	<code>\siunitx_if_number:nTF</code> 7, 1000
<code>\raiseto</code> 60, 91	<code>\siunitx_if_number_token:NTF</code> . . .
<code>\relax</code> 48, 69 7, 7, 135, 1018
<code>\renewcommand</code> 123	<code>\siunitx_number_format:nN</code>
<code>\RequirePackage</code> 3, 3, 3, 4, 9, 39, 88 7, 9, 61, 73, 261
<code>reset-math-version</code> 36	<code>\siunitx_number_format:nNN</code> 7, 9
<code>reset-text-family</code> 36	<code>\siunitx_print:nn</code> 35,
<code>reset-text-series</code> 36	36, 36, 36, 37, 62, 65, 71, 74, 83, 262
<code>reset-text-shape</code> 36	<code>\l_siunitx_print_series_prop</code> . . .
<code>\rmdefault</code> 137 35, 37, 92, 101
<code>\rmfamily</code> 35, 36, 206	<code>\siunitx_unit_format:nN</code>
S	
<code>\scriptspace</code> 805 55, 55, 56, 64, 67, 82, 102
<code>\second</code> 57, 59, 773	<code>\siunitx_unit_format:nNN</code> 56, 102
<code>\selectfont</code> 301	<code>\siunitx_unit_power_set:NnN</code> 66
seq commands:	<code>\l_siunitx_unit_symbolic_seq</code> . . .
<code>\seq_map_inline:Nn</code> 115 21, 28, 57, 115
<code>\seq_new:N</code> 21	siunitx internal commands:
<code>\seq_put_right:Nn</code> 28	<code>__siunitx_declare_column:Nnn</code> 6, 117
<code>\sfdefault</code> 138	<code>__siunitx_load_check:</code> 17
<code>\siemens</code> 58, 815	<code>__siunitx_load_check:n</code> . . . 20, 32, 36
<code>\sievert</code> 58, 815	<code>\l_siunitx_number_arg_tl</code> 14, 17,
<code>\sim</code> 1057	63, 77, 83, 85, 200, 207, 210, 222,
<code>\sisetup</code> 86	237, 249, 307, 323, 355, 590, 596, 600
siunitx commands:	<code>\l__siunitx_number_bracket-</code>
<code>\siunitx_cell_begin:</code> 8, 150	close_tl 678, 812
<code>\siunitx_cell_end:</code> 8, 50, 71, 152	<code>\l__siunitx_number_bracket-</code>
<code>\siunitx_declare_power:NnN</code>	negative_bool 678, 785
. 39, 42, 56, 902, 903	<code>\l__siunitx_number_bracket_open-</code>
<code>\siunitx_declare_power:NnN</code> 56	tl 678, 810
<code>\siunitx_declare_prefix:Nn</code> 48, 56, 56	<code>\l__siunitx_number_comparator_tl</code>
<code>\siunitx_declare_prefix:Nnn</code> 64, 206, 209, 338
. 46, 48,	<code>\l__siunitx_number_explicit-</code>
56, 56, 782, 783, 784, 785, 786, 787,	plus_bool 678, 777, 990
788, 789, 790, 791, 792, 793, 794,	<code>\l__siunitx_number_exponent-</code>
795, 796, 797, 798, 799, 800, 801, 802	base_tl 678, 986
	<code>\l__siunitx_number_exponent-</code>
	product_tl 678, 981, 983
	<code>\l__siunitx_number_exponent_tl</code> . .
 65, 216, 247, 262, 267, 278, 288, 341

__siunitx_number_expression:n ..	__siunitx_number_format_-
..... 28 , 80	uncertainty_aux:nnw 932 , 947 , 954 , 960
\l_siunitx_number_expression_-	__siunitx_number_format_-
bool 28 , 79	uncertainty_aux:nw 952 , 961
\l_siunitx_number_flex_tl	__siunitx_number_format_-
..... 11 , 66 , 91 , 100 , 104 , 128 ,	uncertainty_unaligned: 755
136 , 465 , 467 , 542 , 544 , 567 , 570 , 584	\l__siunitx_number_formatted_tl .
__siunitx_number_format: ... 24 , 739 7 , 26 , 741
__siunitx_number_format:N	\l_siunitx_number_group_-
..... 746 , 750 , 755	decimal_bool 678
__siunitx_number_format:nN 9	\l_siunitx_number_group_-
__siunitx_number_format:nNnnNn 755	integer_bool 678
__siunitx_number_format_-	\l_siunitx_number_group_-
comparator:n 755	minimum_int 678 , 840
__siunitx_number_format_-	\l_siunitx_number_group_-
decimal:n 755	separator_tl 678 , 869 , 894
__siunitx_number_format_-	\l_siunitx_number_imaginary_tl .
decimal_aux:n 755 67 , 75 , 114 , 330 ,
__siunitx_number_format_-	458 , 556 , 583 , 630 , 745 , 746 , 749 , 1008
decimal_loop:NNNN 755	\l_siunitx_number_input_-
__siunitx_number_format_-	comparator_tl 28 , 204 , 1025
digits:nn 755	\l_siunitx_number_input_-
__siunitx_number_format_end: .. 755	complex_tl . 28 , 378 , 401 , 535 , 1024
__siunitx_number_format_-	\l_siunitx_number_input_-
exponent:Nnn 755	decimal_tl 28 , 372 , 391 , 1026
__siunitx_number_format_-	\l_siunitx_number_input_digit_-
integer:nnn 755	tl . 28 , 286 , 366 , 388 , 487 , 504 , 1027
__siunitx_number_format_-	\l_siunitx_number_input_-
integer_aux:n 755	exponent_tl . 28 , 215 , 219 , 220 , 1028
__siunitx_number_format_-	\l_siunitx_number_input_ignore_-
integer_aux_0:n 755	tl 28 , 1029
__siunitx_number_format_-	\l_siunitx_number_input_sign_tl
integer_aux_1:n 755 28 , 259 , 410 , 540 , 594 , 1031
__siunitx_number_format_-	\l_siunitx_number_input_tl
integer_aux_2:n 755 69 , 83 , 119
__siunitx_number_format_-	\l_siunitx_number_input_uncert_-
integer_first:nnNN 755	close_tl 28 , 519 , 1023
__siunitx_number_format_-	\l_siunitx_number_input_uncert_-
integer_loop:NNNN 755	open_tl 28 , 398 , 1030
__siunitx_number_format_sign:N 755	\l_siunitx_number_input_uncert_-
__siunitx_number_format_sign_-	sign_tl 28 , 105 , 1032
aux:N 755	\l_siunitx_number_negative_-
__siunitx_number_format_sign_-	color_tl 678 , 783 , 801
brackets:w 755	\l_siunitx_number_output_-
__siunitx_number_format_sign_-	complex_tl 678
color:w 755	\l_siunitx_number_output_-
__siunitx_number_format_-	decimal_tl 678 , 830
uncertainty:nn 755	\l_siunitx_number_output_-
__siunitx_number_format_-	uncert_close_tl 678 , 916
uncertainty:nnw 755	\l_siunitx_number_output_-
__siunitx_number_format_-	uncert_open_tl 678 , 914
uncertainty:nw 755	__siunitx_number_parse:n 23 , 72 , 1005
__siunitx_number_format_-	
uncertainty_aux:nn 755	

_siunitx_number_parse_check: ..	_siunitx_number_parse_loop_-
..... 87, 89	first:N 350
_siunitx_number_parse_combine_-	_siunitx_number_parse_loop_-
uncert: 107, 124	first:NNN .. 353, 358, 466, 543, 569
_siunitx_number_parse_combine_-	_siunitx_number_parse_loop_-
uncert_auxi:NnnNnnn 124	main:NNNNN 350
_siunitx_number_parse_combine_-	_siunitx_number_parse_loop_-
uncert_auxii:nnnn 124	main_complex:N 350
_siunitx_number_parse_combine_-	_siunitx_number_parse_loop_-
uncert_auxiii:nnnnnn 124	main_decimal:NN 350
_siunitx_number_parse_combine_-	_siunitx_number_parse_loop_-
uncert_auxiv:nnnn 124	main_digit:NNNN 350
_siunitx_number_parse_combine_-	_siunitx_number_parse_loop_-
uncert_auxv:w 124	main_end:NN 350
_siunitx_number_parse_combine_-	_siunitx_number_parse_loop_-
uncert_auxvi:w 124	main_sign:NNN 350
_siunitx_number_parse_comparator:	_siunitx_number_parse_loop_-
..... 86, 197	main_store:NNN 350
_siunitx_number_parse_comparator_-	_siunitx_number_parse_loop_-
aux:Nw 197	main_uncert:NNN 350
_siunitx_number_parse_exponent:	_siunitx_number_parse_loop_-
..... 213, 602	root_swap:NNwNN 20, 350
_siunitx_number_parse_exponent_-	_siunitx_number_parse_loop_-
aux:Nn 213	uncert:NNNNN 350
_siunitx_number_parse_exponent_-	_siunitx_number_parse_replace:
aux:nn 213 84, 296
_siunitx_number_parse_exponent_-	_siunitx_number_parse_replace_-
aux:Nw 213	aux:nN 296
_siunitx_number_parse_exponent_-	_siunitx_number_parse_replace_-
aux:w 213	minus: 298, 321
_siunitx_number_parse_exponent_-	_siunitx_number_parse_replace_-
check:N 213	sign: 296
_siunitx_number_parse_exponent_-	_siunitx_number_parse_sign: ...
cleanup:N 213 211, 587
_siunitx_number_parse_exponent_-	_siunitx_number_parse_sign_-
cleanup:wN 291, 293	aux:Nw 587
_siunitx_number_parse_exponent_-	\c_siunitx_number_parse_sign_-
zero_test:N 213	replacement_tl 296
_siunitx_number_parse_finalise:	\l_siunitx_number_partial_tl ...
..... 122, 327 70, 352, 426, 427,
_siunitx_number_parse_finalise_-	430, 440, 471, 472, 475, 479, 489, 509
aux:N 327	\l_siunitx_number_real_tl
_siunitx_number_parse_finalise_- 12, 67, 76, 96,
aux:Nw 327	108, 110, 113, 128, 135, 170, 172,
_siunitx_number_parse_loop: ...	251, 254, 263, 269, 295, 329, 354,
..... 255, 350	585, 597, 599, 601, 629, 743, 750, 1007
_siunitx_number_parse_loop_-	_siunitx_number_round: 627
after_decimal:NNN 350	_siunitx_number_round:N 627
_siunitx_number_parse_loop_-	_siunitx_number_round_figures:nNnnNn
after_uncert:NNN 350 647
_siunitx_number_parse_loop_-	\l_siunitx_number_round_half_-
break:wN 350	up_bool 604
_siunitx_number_parse_loop_-	\l_siunitx_number_round_min_tl 604
complex_cleanup:wN 350	

<code>__siunitx_number_round_mode_tl .</code>	<code>\c__siunitx_print_mathsf_int</code>	12 , 152
615 , 617 , 619 , 621 , 626 , 639	<code>\c__siunitx_print_mathtt_int</code>	12 , 153
<code>\l__siunitx_number_round_mode_tl</code>	<code>\l__siunitx_print_number_color_-</code>	
<code>__siunitx_number_round_none:nNnnnNn</code>	<code>tl</code>	32
627	<code>\l__siunitx_print_number_mode_tl</code>	32
<code>__siunitx_number_round_places:nNnnnNn</code>	<code>__siunitx_print_store_fam:n</code>	12
657	<code>__siunitx_print_text:n</code>	90 , 201
<code>__siunitx_number_round_places_-</code>	<code>\l__siunitx_print_text_family_-</code>	
<code>decimal:nNnnNn</code>	<code>bool</code>	53 , 205
657	<code>\l__siunitx_print_text_family_tl</code>	32
<code>__siunitx_number_round_places_-</code>	<code>__siunitx_print_text_replace:N</code>	201
<code>integer:nNnnNn</code>	<code>__siunitx_print_text_replace:n</code>	201
657	<code>__siunitx_print_text_replace:NnN</code>	
<code>\l__siunitx_number_round_-</code>	<code>__siunitx_print_text_scripts:</code>	201
<code>precision_int</code>	<code>__siunitx_print_text_scripts:NnN</code>	201
604 , 651 , 661	<code>__siunitx_print_text_scripts_-</code>	
<code>__siunitx_number_round_uncertainty:nNnnnNn</code>	<code>one:Nn</code>	264 , 271 , 302
672	<code>__siunitx_print_text_scripts_-</code>	
<code>\l__siunitx_number_tab_tl</code>	<code>one:NnN</code>	201
8 , 11 , 17 , 771 , 828 , 831 , 906 , 908 , 923 , 924 , 925 , 926 , 968 , 976 , 984 , 996	<code>__siunitx_print_text_scripts_-</code>	
<code>\l__siunitx_number_tight_bool</code>	<code>two:n</code>	201
678 , 794 , 978	<code>__siunitx_print_text_scripts_-</code>	
<code>\l__siunitx_number_tmp_tl</code>	<code>two:nn</code>	201
6 , 103 , 106 , 218 , 223 , 229 , 230 , 238 , 239	<code>__siunitx_print_text_scripts_-</code>	
<code>\l__siunitx_number_uncert_-</code>	<code>two:NnNn</code>	201
<code>separate_bool</code>	<code>\l__siunitx_print_text_series_-</code>	
678 , 904	<code>bool</code>	55 , 207
<code>\l__siunitx_number_unity_-</code>	<code>\l__siunitx_print_text_series_tl</code>	32
<code>mantissa_bool</code>	<code>\l__siunitx_print_text_shape_-</code>	
678 , 822 , 975	<code>bool</code>	57 , 209
<code>\l__siunitx_number_valid_tl</code>	<code>\l__siunitx_print_text_shape_tl</code>	32
999 , 1021 , 1034	<code>__siunitx_print_text_sub:n</code>	201
<code>\l__siunitx_number_validate_bool</code>	<code>__siunitx_print_text_super:n</code>	201
71 , 116 , 1004	<code>\l__siunitx_print_tmp_box</code>	11 , 22
<code>\l__siunitx_number_zero_exponent_-</code>	<code>\l__siunitx_print_tmp_tl</code>	102 , 103 , 172 , 173 , 175 , 178 , 181 , 195 , 196 , 197 , 217 , 218 , 219 , 274 , 275 , 277
<code>bool</code>	<code>\l__siunitx_print_unit_color_tl</code>	32
678 , 970	<code>\l__siunitx_print_unit_mode_tl</code>	32
<code>__siunitx_print_match:n</code>	<code>__siunitx_table_align_auxi:nn</code>	171
86	<code>__siunitx_table_align_auxii:nn</code>	171
<code>__siunitx_print_math:n</code>	<code>__siunitx_table_align_center:n</code>	171
89 , 97	<code>\l__siunitx_table_align_comparator_-</code>	
<code>__siunitx_print_math_aux:Nn</code>	<code>bool</code>	232
97	<code>\l__siunitx_table_align_exponent_-</code>	
<code>__siunitx_print_math_auxi:n</code>	<code>bool</code>	232
97	<code>__siunitx_table_align_left:n</code>	171
<code>__siunitx_print_math_auxii:n</code>	<code>__siunitx_table_align_right:n</code>	171
97	<code>\l__siunitx_table_align_text_tl</code>	204 , 213
<code>__siunitx_print_math_auxiii:n</code>	<code>\l__siunitx_table_align_uncertainty_-</code>	
97	<code>bool</code>	232
<code>__siunitx_print_math_auxiv:n</code>		
97		
<code>__siunitx_print_math_auxv:n</code>		
97		
<code>\l__siunitx_print_math_family_-</code>		
<code>bool</code>		
32 , 133		
<code>\l__siunitx_print_math_font_bool</code>		
32 , 148		
<code>__siunitx_print_math_script:n</code>		
97		
<code>__siunitx_print_math_sub:n</code>		
97		
<code>__siunitx_print_math_super:n</code>		
97		
<code>__siunitx_print_math_text:n</code>		
97		
<code>__siunitx_print_math_version:nn</code>		
97		
<code>\l__siunitx_print_math_version_-</code>		
<code>bool</code>		
32 , 110		
<code>\l__siunitx_print_math_weight_-</code>		
<code>bool</code>		
32 , 99		
<code>\c__siunitx_print_mathrm_int</code>		
12 , 164		

\l__siunitx_table_alignment_tl ..	\l__siunitx_unit_autofrac_bool ..
..... 232, 249	407, 414, 421, 428, 435, 442, 467, 714
__siunitx_table_collect_begin:N	\l__siunitx_unit_bracket_bool ...
..... 11, 19 465, 489, 510, 554, 638, 659
__siunitx_table_collect_end: 16, 97	\l__siunitx_unit_bracket_close_-
__siunitx_table_collect_group:n 31	tl 396, 514, 596
__siunitx_table_collect_loop: ..	\l__siunitx_unit_bracket_open_tl
..... 25, 29, 31 396, 512, 593
__siunitx_table_collect_-	\l__siunitx_unit_current_tl
search:NnTF 31 475, 490, 549, 551,
__siunitx_table_collect_search_-	578, 586, 624, 627, 633, 685, 693, 696
aux:NNn 31	\l__siunitx_unit_denominator_-
\l__siunitx_table_collect_tl ...	bracket_bool 396, 657
..... 18, 21, 39, 53, 74, 99, 100, 104	\l__siunitx_unit_denominator_tl .
__siunitx_table_collect_token:N 31	477, 482, 658, 703, 744, 753, 762, 769
\l__siunitx_table_column_width_-	\l__siunitx_unit_font_bool
dim 164, 180 466, 491, 691, 698
__siunitx_table_direct_begin: . 220	__siunitx_unit_format:nNN 102
__siunitx_table_direct_begin:N .	__siunitx_unit_format_aux: ... 102
..... 12, 220	__siunitx_unit_format_bracket:N
__siunitx_table_direct_begin_- 508, 551, 753
aux: 220	__siunitx_unit_format_finalise:
\l__siunitx_table_fixed_width_- 498, 701
bool 164, 179	__siunitx_unit_format_finalise_-
\l__siunitx_table_format_tl ... 216	autofrac: 701
\l__siunitx_table_number_tl	__siunitx_unit_format_finalise_-
..... 94, 102, 108, 113, 125, 136, 138	fraction: 719, 725, 738
\l__siunitx_table_parse_bool . 4, 10	__siunitx_unit_format_finalise_-
\l__siunitx_table_parse_only_-	fractional: 701
bool 232, 251	__siunitx_unit_format_finalise_-
\l__siunitx_table_post_tl	power: 701
94, 103, 107, 114, 127, 133, 140, 143	__siunitx_unit_format_finalise_-
\l__siunitx_table_pre_tl	symbol: 718, 728, 747
94, 101, 106, 109, 112, 126, 139	__siunitx_unit_format_font: ...
__siunitx_table_print:nnn . 111, 247 520, 590, 602, 612, 637, 689
__siunitx_table_print_aligned:nnn	__siunitx_unit_format_literal:n
..... 247 127, 129, 143
__siunitx_table_print_non_-	__siunitx_unit_format_literal_-
aligned:nnn 247	auxi:w 143
__siunitx_table_print_text:n ...	__siunitx_unit_format_literal_-
..... 109, 211, 223	auxii:n 176, 180
__siunitx_table_skip:n	__siunitx_unit_format_literal_-
159, 183, 185, 197, 199	auxii:w 143
__siunitx_table_split_group:n . 117	__siunitx_unit_format_literal_-
__siunitx_table_split_loop: 104, 117	auxiii:w 143
__siunitx_table_split_tidy:N ...	__siunitx_unit_format_literal_-
..... 106, 107, 146	auxiv:w 143
__siunitx_table_split_tidy:Nn . 146	__siunitx_unit_format_literal_-
__siunitx_table_split_token:N . 117	auxv:w 143
\l__siunitx_table_tmp_tl . 3, 261, 262	__siunitx_unit_format_output: ..
__siunitx_tmp:w 95, 97, 120, 122 496, 635
\l__siunitx_tmp_tl	__siunitx_unit_format_output_-
38, 61, 62, 64, 65, 73, 74, 82, 83	aux: 635

_siunitx_unit_format_output_- aux:nm	635	199, 200, 203, 204, 208, 209, 211, 212, 235, 615, 846, 852, 863, 869, 876
_siunitx_unit_format_output_- denominator:	635	\l_siunitx_unit_numerator_bool 81, 471, 492, 532, 642
_siunitx_unit_format_parsed:	125, 479	_siunitx_unit_parse:n 123, 253
_siunitx_unit_format_parsed_- aux:n	479	_siunitx_unit_parse_add:nmmn .. 265, 282, 296, 314, 324, 331, 336, 350
_siunitx_unit_format_power: ..	518	\l_siunitx_unit_parse_bool 119, 396
_siunitx_unit_format_power_- aux:wTF	518	_siunitx_unit_parse_finalise: 263, 386
_siunitx_unit_format_power_- negative:	518	_siunitx_unit_parse_finalise:n 262, 355
_siunitx_unit_format_power_- negative_aux:w	518	_siunitx_unit_parse_per: .. 81, 341
_siunitx_unit_format_power_- positive:	518	_siunitx_unit_parse_power:nmN 43, 46, 93, 96, 279
_siunitx_unit_format_power_- superscript:	518	_siunitx_unit_parse_prefix:Nm 52, 279
_siunitx_unit_format_prefix: .	556	_siunitx_unit_parse_qualifier:nm 66, 90, 279
_siunitx_unit_format_prefix_- power:	556	_siunitx_unit_parse_special:n 84, 87, 279
_siunitx_unit_format_prefix_- symbol:	556	_siunitx_unit_parse_unit:Nm 75, 328
_siunitx_unit_format_qualifier:	579	\l_siunitx_unit_parsed_prop 70, 124, 250, 255, 268, 275, 293, 312, 358, 360, 364, 372, 376, 381, 392, 504, 569
_siunitx_unit_format_qualifier_- bracket_⏟:	579	\l_siunitx_unit_parsing_bool 9, 33, 148, 256
_siunitx_unit_format_qualifier_- bracket:	588	\l_siunitx_unit_part_tl 366, 368, 369, 370, 377, 475, 505, 522, 535, 538, 540, 552, 565, 573, 578, 586, 591, 595, 603, 607, 613, 618, 626, 633
_siunitx_unit_format_qualifier_- combine:	579	\l_siunitx_unit_per_bool 73, 250, 257, 334, 345
_siunitx_unit_format_qualifier_- phrase:	579	\l_siunitx_unit_per_symbol_bool 408, 415, 422, 429, 436, 443, 467, 665, 671, 717
_siunitx_unit_format_qualifier_- subscript:	579	\l_siunitx_unit_per_symbol_tl 396, 752
_siunitx_unit_format_special: 622		\l_siunitx_unit_position_int 77, 250, 259, 261, 281, 288, 299, 311, 315, 325, 330, 332, 337, 351, 391, 481, 485, 487, 493, 503, 568
_siunitx_unit_format_unit: ..	630	\l_siunitx_unit_powers_positive_- bool 409, 416, 423, 430, 437, 444, 467, 533, 705
\l_siunitx_unit_formatted_tl 77, 101, 117, 133, 160, 168, 169, 215, 218, 220, 483, 666, 667, 712, 713, 727, 729, 733, 734, 735, 740, 743, 749, 751, 758, 761, 765, 767		\l_siunitx_unit_prefix_fp 118, 135, 474, 484, 572
\l_siunitx_unit_fraction_- function_tl	396, 742	\l_siunitx_unit_prefix_power_- bool 104, 109, 473, 558
_siunitx_unit_if_symbolic:n ...	11	\l_siunitx_unit_prefixes_- forward_prop 48, 564
_siunitx_unit_if_symbolic:nTF	11, 73, 121	\l_siunitx_unit_prefixes_- reverse_prop 48
_siunitx_unit_literal_power:nN	42, 92, 141	
_siunitx_unit_literal_special:nN	83, 86, 142	
\c_siunitx_unit_math_subscript_- tl	7, 155,	

<code>\l__siunitx_unit_product_tl</code>	<code>\@ifpackagelater</code> 4
. 97, 177, 650, 661, 768	<code>\@ifpackageloaded</code> 7, 23, 59, 91, 135, 190
<code>\l__siunitx_unit_qualifier_mode_-</code>	<code>\@maybe@unskip</code> 65
<code>tl</code> 396, 472, 584	<code>\@nil</code> 94, 108
<code>\l__siunitx_unit_qualifier_-</code>	<code>\@temptokena</code> 125, 127
<code>phrase_tl</code> 396, 605	<code>\AtBeginDocument</code> 38
<code>\l__siunitx_unit_separator_tl</code> . . 143	<code>\c@MaxMatrixCols</code> 107
<code>__siunitx_unit_set_symbolic:Nnn</code>	<code>\env@matrix</code> 101
. 22, 41, 44, 50, 64, 70, 79	<code>\f@family</code> 135
<code>__siunitx_unit_set_symbolic:NNnnn</code>	<code>\f@series</code> 102
. 23, 25, 26	<code>\ifcellspace@m</code> 100
<code>__siunitx_unit_set_symbolic:NNpnn</code>	<code>\m@th</code> 225, 288
. 22	<code>\math@version</code> 116
<code>__siunitx_unit_set_symbolic:Npnn</code>	<code>\NC@do</code> 95, 120, 121
. 22, 82, 85, 88, 91, 94	<code>\NC@find</code> 130
<code>\l__siunitx_unit_sticky_per_bool</code>	<code>\NC@find@S</code> 98
. 73, 246, 343	<code>\NC@list</code> 4, 4, 96, 97, 121, 122
<code>\l__siunitx_unit_test_bool</code>	<code>\new@ifnextchar</code> 104
. 10, 14, 31, 258	<code>\protected@edef</code> 68, 77, 99, 158
<code>\l__siunitx_unit_tmp_fp</code> 4, 105	<code>\sf@size</code> 301
<code>\l__siunitx_unit_tmp_int</code> . . 4, 281, 283	<code>\tab@setcr</code> 66
<code>\l__siunitx_unit_tmp_tl</code>	<code>\z@</code> 301
. 4, 15, 17, 149, 150, 152, 154,	tex commands:
158, 159, 161, 164, 267, 269, 276,	<code>\tex_cr:D</code> 26
287, 293, 310, 312, 357, 358, 361,	<code>\tex_kern:D</code> 162
362, 365, 373, 377, 382, 390, 392,	<code>\tex_the:D</code> 97, 122
502, 505, 567, 570, 571, 573, 727, 732	<code>\text</code> 35, 120, 203
<code>\l__siunitx_unit_total_int</code>	text-family-to-math 37
. 478, 481, 487	text-weight-to-math 37
<code>\l__siunitx_unit_two_part_bool</code> . .	<code>\textcolor</code> 35,
410, 417, 424, 431, 438, 445, 467, 654	36, 36, 37, 55, 60, 75, 86, 87, 801
skip commands:	<code>\textminus</code> 35, 37, 232
<code>\skip_horizontal:n</code> 161	<code>\textpm</code> 35, 228
<code>\c_zero_skip</code> 162	<code>\textsubscript</code> 35, 250, 281
<code>\sp</code> 79	<code>\textsuperscript</code> 35, 255
<code>\square</code> 59, 902	<code>\the</code> 127
<code>\squared</code> 60, 855, 902	<code>\times</code> 1050
<code>\steradian</code> 58, 815	tl commands:
sticky-per 61	<code>\c_empty_tl</code> 876
<code>\stilb</code> 59, 891	<code>\tl_clear:N</code>
<code>\stokes</code> 59, 891	11, 21, 75, 76, 101, 102, 103, 108,
str commands:	110, 117, 135, 136, 160, 209, 251,
<code>\str_case:x:nnTF</code> 135	263, 269, 295, 352, 459, 479, 482,
<code>\str_if_eq:nnTF</code> . . . 190, 279, 775, 781	483, 490, 561, 566, 583, 584, 585, 601
<code>\str_if_eq_p:nn</code>	<code>\tl_const:Nn</code> 7, 310
. 438, 507, 819, 821, 971, 974, 991	<code>\tl_count:n</code> . . . 133, 140, 840, 851, 910
<code>\str_if_eq_x:nnTF</code> 116, 153, 540	<code>\tl_head:n</code> 172, 219
T	<code>\tl_if_blank:nTF</code> 3,
<code>\tabularnewline</code> 47, 49, 68, 70	17, 225, 231, 233, 250, 268, 483,
<code>\tera</code> 58, 792	554, 577, 649, 659, 674, 769, 829, 901
<code>\tesla</code> 58, 815	<code>\tl_if_blank_p:n</code> 4, 95, 99, 176, 177, 820
TeX and L ^A T _E X 2 _ε commands:	<code>\tl_if_empty:NTF</code> . . 73, 85, 91, 108,
<code>\@ifnextchar</code> 104	125, 133, 138, 148, 161, 210, 215,

219, 254, 262, 334, 471, 600, 634, 683, 703, 712, 743, 745, 749, 758, 783	\token_to_str:N 29, 116, 153, 155, 272, 305, 319, 324, 917, 920
\tl_if_empty:nTF 527	\tonne 58, <u>826</u>
\tl_if_empty_p:N	\tothe 60, <u>91</u>
. . 113, 114, 426, 658, 666, 1007, 1008	\ttdefault 139
\tl_if_in:NnTF 5, 57, 105, 204, 259, 286, 366, 372, 378, 388, 391, 398, 401, 410, 487, 504, 519, 535, 540, 594	U
\tl_if_in:nnTF 1034	\unit 7, <u>68</u>
\tl_map_inline:Nn 220	unit-close-bracket 61
\tl_map_inline:nn 55	unit-color 37
\tl_new:N 3, 6, 6, 7, 8, 18, 32, 33, 38, 63, 64, 65, 66, 67, 68, 69, 70, 94, 95, 96, 101, 210, 245, 246, 472, 475, 476, 477, 626, 999	unit-mode 37
\tl_put_right 139, 140	unit-open-bracket 62
\tl_put_right:Nn 12, 39, 53, 58, 74, 126, 127, 136, 143, 288, 363, 374, 428, 440, 473, 489, 509, 586, 632	unit-product 62
\tl_replace_all:Nnn 3, 150, 152, 154, 173, 175, 178, 196, 222, 244, 307, 323	\unskip 46, 67
\tl_set:Nn 15, 17, 26, 54, 68, 83, 103, 132, 149, 156, 168, 170, 172, 189, 195, 206, 207, 208, 215, 216, 217, 218, 242, 247, 249, 267, 267, 274, 278, 287, 310, 336, 357, 362, 368, 390, 427, 451, 453, 455, 457, 465, 472, 502, 535, 542, 549, 556, 567, 567, 571, 591, 596, 597, 599, 603, 613, 615, 617, 619, 621, 624, 636, 667, 680, 693, 713, 727, 729, 740, 741, 749, 765, 1021	\upshape 35, 36, 210
\tl_set_eq:NN	use commands:
. 45, 67, 83, 177, 458, 578, 760	\use:N 26, 77, 213, 249, 506, 581, 639, 842, 848
\tl_use:N 80, 181, 197, 219	\use:n 74, 121, 155, 155, 156, 165, 174, 181, 192, 196, 225, 226, 234, 302, 564
\l_tmpa_tl 55, 56	\use_i:nnnn 104
token commands:	\use_i_delimit_by_q_stop:nw 91
\l_peek_token 261	\use_iv:nnnn 96, 100
\token_if_eq_meaning:NNTF 90	\use_none:n 483, 886
	\use_none:nn 890
	V
	\volt 58, <u>815</u>
	W
	\watt 58, <u>815</u>
	\weber 58, <u>815</u>
	Y
	\yocto 58, <u>782</u>
	\yotta 58, <u>792</u>
	Z
	\zepto 58, <u>782</u>
	\zetta 58, <u>792</u>