

siunitx – A comprehensive (SI) units package*

Joseph Wright[†]

Released 2018-07-21

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
1.5	“Glue” commands	4
1.5.1	Table column	4
II	siunitx-angle – Formatting angles	6
0.1	Key-value options	6
1	siunitx-angle implementation	7
III	siunitx-number – Parsing and formatting numbers	12
0.1	Key-value options	13
1	siunitx-number implementation	16
1.1	Initial set-up	16
1.2	Main formatting routine	16
1.3	Parsing numbers	17
1.4	Processing numbers	31
1.5	Formatting parsed numbers	46
1.6	Miscellaneous tools	53
1.7	Messages	54
1.8	Standard settings for module options	54

*This file describes v3.0.0-alpha, last revised 2018-07-21.

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

IV	siunitx-print – Printing material with font control	56
1	Printing quantities	56
1.1	Key–value options	57
2	siunitx-print implementation	58
2.1	Initial set up	58
2.2	Printing routines	59
2.3	Standard settings for module options	66
V	siunitx-font – Font-related settings	67
1	siunitx-symbol implementation	67
VI	siunitx-table – Formatting numbers in tables	71
1	siunitx-table implementation	71
1.1	Interface functions	71
1.2	Collecting tokens	72
1.3	Separating collected material	74
1.4	Printing numbers in cells: spacing	75
1.5	Printing just text	77
1.6	Number alignment: core ideas	77
1.7	Directly printing without collection	80
1.8	Printing numbers in cells: main functions	82
1.9	Standard settings for module options	89
VII	siunitx-unit – Parsing and formatting units	90
1	Formatting units	90
2	Defining symbolic units	91
3	Pre-defined symbolic unit components	92
3.1	Key–value options	95
4	siunitx-unit implementation	97
4.1	Initial set up	97
4.2	Defining symbolic unit	98
4.3	Non-standard symbolic units	100
4.4	Main formatting routine	101
4.5	Formatting literal units	103
4.6	Parsing symbolic units	105
4.7	Formatting parsed units	110
4.8	Non-Latin character support	119
4.9	Pre-defined unit components	120
4.10	Messages	124
4.11	Standard settings for module options	124

VIII	siunitx-abbreviations – Abbreviations	126
1	siunitx-abbreviation implementation	128
IX	siunitx-emulation – Emulation	132
1	siunitx-emulation implementation	132
1.1	Version 2	132
1.2	Document commands	132
1.2.1	Number options	133
1.2.2	Unit options	135
1.2.3	Table options	136
1.2.4	Loadable configuration	138
	Index	140

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}{2018-06-01}
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} {2018-07-21} {3.0.0-alpha}
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 }
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   \mode_leave_vertical:
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 ??.)

`\ang`

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

`\num`

`\unit`

```

68 \NewDocumentCommand \ang { 0 { } > { \SplitArgument { 2 } { ; } } m }
69 {
70   \mode_leave_vertical:
71   \group_begin:
72     \keys_set:nn { siunitx } {#1}
73     \__siunitx_angle:nnn #2
74   \group_end:
75 }
76 \NewDocumentCommand \num { 0 { } m }
77 {
78   \mode_leave_vertical:
79   \group_begin:
80     \keys_set:nn { siunitx } {#1}
81     \siunitx_number_format:nN {#2} \l__siunitx_tmp_tl
82     \siunitx_print:nV { number } \l__siunitx_tmp_tl
83   \group_end:
84 }
85 \NewDocumentCommand \unit { 0 { } m }
86 {
87   \mode_leave_vertical:
88   \group_begin:
89     \keys_set:nn { siunitx } {#1}
90     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
91     \siunitx_print:nV { unit } \l__siunitx_tmp_tl
92   \group_end:
93 }

```

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

`\tablenum`

Slightly odd set up at present: we have to have the `\ignorespaces`.

```

94 \NewDocumentCommand \tablenum { 0 { } m }
95 {
96   \mode_leave_vertical:
97   \group_begin:

```

```

98     \keys_set:nn { siunitx } {#1}
99     \siunitx_cell_begin:w
100     \ignorespaces #2
101     \siunitx_cell_end:
102     \group_end:
103 }

```

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

`\sisetup` A very thin wrapper.

```

104 \NewDocumentCommand \sisetup { m }
105 { \keys_set:nn { siunitx } {#1} }

```

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

1.5 “Glue” commands

`__siunitx_angle:nnn` The document level interface for `\ang` needs some “glue” to work with the code-level API.

```

106 \cs_new_protected:Npn \__siunitx_angle:nnn #1#2#3
107 {
108     \tl_if_novalue:nTF {#2}
109     { \siunitx_angle:n {#1} }
110     {
111         \tl_if_novalue:nTF {#3}
112         { \siunitx_angle:nnn {#1} {#2} { } }
113         { \siunitx_angle:nnn {#1} {#2} {#3} }
114     }
115 }

```

(End definition for `__siunitx_angle:nnn`.)

1.5.1 Table column

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

```

116 \RequirePackage { array }

```

`__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     \newcolumntype {#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 S
148     {
149         \keys_set:nn { siunitx } {#1}
150         \siunitx_cell_begin:w
151     }
152     { \siunitx_cell_end: }
153 }

```

(End definition for `__siunitx_declare_column:Nnn`.)

```

154 \endpackage

```


Part II

siunitx-angle – Formatting angles

<hr/> <code>\siunitx_angle:n</code> <hr/>	<code>\siunitx_angle:n {⟨angle⟩}</code>
<code>\siunitx_angle:nnn</code>	<code>\siunitx_angle:nnn {⟨degrees⟩} {⟨minutes⟩} {⟨seconds⟩}</code>

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

0.1 Key-value options

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

<hr/> <code>angle-mode</code> <hr/>	<code>angle-mode = ⟨choice⟩</code>
-------------------------------------	------------------------------------

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

<hr/> <code>arc-separator</code> <hr/>	<code>arc-separator = ⟨separator⟩</code>
--	--

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

<hr/> <code>arc-separator-over-decimal</code> <hr/>	<code>arc-separator-over-decimal = true false</code>
---	--

<hr/> <code>fill-arc-degrees</code> <hr/>	<code>fill-arc-degrees = true false</code>
---	--

<hr/> <code>fill-arc-minutes</code> <hr/>	<code>fill-arc-minutes = true false</code>
---	--

<hr/> <code>fill-arc-seconds</code> <hr/>	<code>fill-arc-seconds = true false</code>
---	--

<hr/> <code>number-angle-product</code> <hr/>	<code>number-angle-product = ⟨separator⟩</code>
---	---

Inserted between the value of an angle and the unit (degree, minute or second component). The standard setting is `\,`.

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>
```

```
\l__siunitx_angle_tmp_tl
```

Scratch space.

```
3 \tl_new:N \l__siunitx_angle_tmp_tl
```

(End definition for \l__siunitx_angle_tmp_tl.)

```
\l__siunitx_angle_force_arc_bool
\l__siunitx_angle_force_decimal_bool
\l__siunitx_angle_astronomy_bool
\l__siunitx_angle_separator_tl
\l__siunitx_angle_fill_degrees_bool
\l__siunitx_angle_fill_minutes_bool
\l__siunitx_angle_fill_seconds_bool
\l__siunitx_angle_product_tl

4 \keys_define:nn { siunitx }
5 {
6   angle-mode .choice: ,
7   angle-mode / arc .code:n =
8   {
9     \bool_set_true:N \l__siunitx_angle_force_arc_bool
10    \bool_set_false:N \l__siunitx_angle_force_decimal_bool
11  } ,
12  angle-mode / decimal .code:n =
13  {
14    \bool_set_false:N \l__siunitx_angle_force_arc_bool
15    \bool_set_true:N \l__siunitx_angle_force_decimal_bool
16  } ,
17  angle-mode / input .code:n =
18  {
19    \bool_set_false:N \l__siunitx_angle_force_arc_bool
20    \bool_set_false:N \l__siunitx_angle_force_decimal_bool
21  } ,
22  angle-symbol-over-decimal .bool_set:N =
23  \l__siunitx_angle_astronomy_bool ,
24  arc-separator .tl_set:N =
25  \l__siunitx_angle_separator_tl ,
26  fill-arc-degrees .bool_set:N =
27  \l__siunitx_angle_fill_degrees_bool ,
28  fill-arc-minutes .bool_set:N =
29  \l__siunitx_angle_fill_minutes_bool ,
30  fill-arc-seconds .bool_set:N =
31  \l__siunitx_angle_fill_seconds_bool ,
32  number-angle-product .tl_set:N =
33  \l__siunitx_angle_product_tl
34 }
35 \bool_new:N \l__siunitx_angle_force_arc_bool
36 \bool_new:N \l__siunitx_angle_force_decimal_bool
```

(End definition for \l__siunitx_angle_force_arc_bool and others.)

```
\siunitx_angle:n
```

```
\siunitx_angle:nmn
```

```
\_siunitx_angle_arc_convert:n
```

The first step here is to force format conversion if required. Going to a decimal is easy, going to arc format is a bit more painful: avoid repeating calculations mainly for code readability.

```

37 \cs_new_protected:Npn \siunitx_angle:n #1
38 {
39   \bool_if:NTF \l__siunitx_angle_force_arc_bool
40   {
41     \use:x
42     { \__siunitx_angle_arc_convert:n { \fp_eval:n {#1} } }
43   }
44   {
45     \siunitx_number_format:nN {#1} \l__siunitx_angle_degrees_tl
46     \__siunitx_angle_arc_print:VVV
47     \l__siunitx_angle_degrees_tl
48     \c_empty_tl
49     \c_empty_tl
50   }
51 }
52 \cs_new_protected:Npn \siunitx_angle:nnn #1#2#3
53 {
54   \bool_if:NTF \l__siunitx_angle_force_decimal_bool
55   {
56     \exp_args:Nx \siunitx_angle:n
57     { \fp_eval:n { #1 + (#2) / 60 + (#3) / 3600 } }
58   }
59   { \__siunitx_angle_arc_sign:nnn {#1} {#2} {#3} }
60 }
61 \cs_new_protected:Npn \__siunitx_angle_arc_convert:n #1
62 {
63   \use:x
64   {
65     \siunitx_angle:nnn
66     { \fp_eval:n { trunc(#1,0) } }
67     { \fp_eval:n { trunc((#1 - trunc(#1,0)) * 60,0) } }
68     {
69       \fp_eval:n
70       {
71         (
72           (#1 - trunc(#1,0)) * 60
73           - trunc((#1 - trunc(#1,0)) * 60,0)
74         )
75         * 60
76       }
77     }
78   }
79 }

```

(End definition for `\siunitx_angle:n`, `\siunitx_angle:nnn`, and `__siunitx_angle_arc_convert:n`. These functions are documented on page 6.)

```

\l__siunitx_angle_degrees_tl Space for formatting parsed numbers.
\l__siunitx_angle_minutes_tl 80 \tl_new:N \l__siunitx_angle_degrees_tl
\l__siunitx_angle_seconds_tl 81 \tl_new:N \l__siunitx_angle_minutes_tl
                             82 \tl_new:N \l__siunitx_angle_seconds_tl

```

(End definition for `\l__siunitx_angle_degrees_tl`, `\l__siunitx_angle_minutes_tl`, and `\l__siunitx_angle_seconds_tl`.)

\l__siunitx_angle_sign_tl For the “sign shuffle”.

```
83 \tl_new:N \l__siunitx_angle_sign_tl
```

(End definition for \l__siunitx_angle_sign_tl.)

__siunitx_angle_arc_sign:nn To get the sign in the right place whilst dealing with zero filling means doing some shuffling. That means doing processing of each number manually.

```
\__siunitx_angle_arc_sign:nn
\__siunitx_angle_extract_sign:nnnnnnnn
\__siunitx_angle_sign:nnnnnnnn
84 \cs_new_protected:Npn \__siunitx_angle_arc_sign:nnn #1#2#3
85 {
86   \group_begin:
87   \keys_set:nn { siunitx }
88   {
89     input-close-uncertainty = ,
90     input-exponent-markers = ,
91     input-open-uncertainty = ,
92     input-uncertainty-signs =
93   }
94   \tl_clear:N \l__siunitx_angle_sign_tl
95   \__siunitx_angle_arc_sign:nn {#1} { degrees }
96   \__siunitx_angle_arc_sign:nn {#2} { minutes }
97   \__siunitx_angle_arc_sign:nn {#3} { seconds }
98   \tl_if_empty:NF \l__siunitx_angle_sign_tl
99   {
100     \clist_map_inline:nn { degrees , minutes , seconds }
101     {
102       \tl_if_empty:cF { l__siunitx_angle_ ##1 _tl }
103       {
104         \tl_set:cx { l__siunitx_angle_ ##1 _tl }
105         {
106           { }
107           { \exp_not:V \l__siunitx_angle_sign_tl }
108           \exp_after:wN \exp_after:wN \exp_after:wN
109             \__siunitx_angle_sign:nnnnnnn
110             \cs:w l__siunitx_angle_ ##1 _tl \cs_end:
111         }
112         \clist_map_break:
113       }
114     }
115   }
116   \clist_map_inline:nn { degrees , minutes , seconds }
117   {
118     \tl_if_empty:cF { l__siunitx_angle_ ##1 _tl }
119     {
120       \tl_set:cx { l__siunitx_angle_ ##1 _tl }
121       {
122         \exp_args:Nc \siunitx_number_format:N
123           { l__siunitx_angle_ ##1 _tl }
124       }
125     }
126   }
127   \__siunitx_angle_arc_print:VVV
128   \l__siunitx_angle_degrees_tl
129   \l__siunitx_angle_minutes_tl
130   \l__siunitx_angle_seconds_tl
```

```

131 \group_end:
132 }
133 \cs_new_protected:Npn \__siunitx_angle_arc_sign:nn #1#2
134 {
135   \tl_if_blank:nTF {#1}
136   {
137     \bool_if:cTF { l__siunitx_angle_fill_ #2 _bool }
138     {
139       \tl_set:cn { l__siunitx_angle_ #2 _tl }
140       { { } { } { } { 0 } { } { } { } { 0 } }
141     }
142     { \tl_clear:c { l__siunitx_angle_ #2 _tl } }
143   }
144   {
145     \siunitx_number_parse:nN {#1} \l__siunitx_angle_tmp_tl
146     \exp_after:wN \__siunitx_angle_extract_sign:nnnnnnnn \l__siunitx_angle_tmp_tl {#2}
147   }
148 }
149 \cs_new_protected:Npn \__siunitx_angle_extract_sign:nnnnnnnn #1#2#3#4#5#6#7#8
150 {
151   \tl_if_blank:nTF {#2}
152   { \tl_set_eq:cN { l__siunitx_angle_ #8 _tl } \l__siunitx_angle_tmp_tl }
153   {
154     \tl_set:cn { l__siunitx_angle_ #8 _tl }
155     { {#1} { } {#3} {#4} {#5} {#6} {#7} }
156     \tl_set:Nn \l__siunitx_angle_sign_tl {#2}
157     \keys_set:nn { siunitx }
158     { input-comparators = , input-signs = }
159   }
160 }
161 \cs_new:Npn \__siunitx_angle_sign:nnnnnnnn #1#2#3#4#5#6#7
162 { \exp_not:n { {#3} {#4} {#5} {#6} {#7} } }

```

(End definition for __siunitx_angle_arc_sign:nnn and others.)

```

\__siunitx_angle_arc_print:nnn
\__siunitx_angle_arc_print:VVV
\__siunitx_angle_arc_print_aux:nnn

```

The final stage of printing an angle is to put together the three parts: this works even for decimal angles as they will blank arguments for the other two parts.

```

163 \cs_new_protected:Npn \__siunitx_angle_arc_print:nnn #1#2#3
164 {
165   \__siunitx_angle_arc_print_aux:nnn {#1} { \degree } {#2#3}
166   \__siunitx_angle_arc_print_aux:nnn {#2} { \arcminute } {#3}
167   \__siunitx_angle_arc_print_aux:nnn {#3} { \arcsecond } { }
168 }
169 \cs_generate_variant:Nn \__siunitx_angle_arc_print:nnn { VVV }
170 \cs_new_protected:Npn \__siunitx_angle_arc_print_aux:nnn #1#2#3
171 {
172   \tl_if_blank:nF {#1}
173   {
174     \siunitx_print:nn { number } {#1}
175     \nobreak
176     \l__siunitx_angle_product_tl
177     \siunitx_unit_format:nN {#2} \l__siunitx_angle_tmp_tl
178     \siunitx_print:nV { unit } \l__siunitx_angle_tmp_tl
179     \tl_if_blank:nF {#3}

```

```

180         {
181             \nobreak
182             \l__siunitx_angle_separator_tl
183         }
184     }
185 }

(End definition for \__siunitx_angle_arc_print:nnn and \__siunitx_angle_arc_print_aux:nnn.)

186 \keys_set:nn { siunitx }
187 {
188     angle-mode = input ,
189     angle-symbol-over-decimal = false ,
190     arc-separator = ,
191     fill-arc-degrees = false ,
192     fill-arc-minutes = false ,
193     fill-arc-seconds = false ,
194     number-angle-product =
195 }

196 </package>

```

Part III

siunitx-number – Parsing and formatting numbers

<code>\siunitx_number_format:nN</code>	<code>\siunitx_number_format:nN {<number>} <tl var></code>
<code>\siunitx_number_format:VN</code>	

<code>\siunitx_number_parse:nN</code>	★ <code>\siunitx_number_parse:nN {<number>} <tl var></code>
---------------------------------------	---

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

The structure of a valid number is:

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

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

S A single symmetrical uncertainty (*e.g.* a statistical standard uncertainty)

<code>\siunitx_number_process:NN</code>	<code>\siunitx_number_process:N <tl var1> <tl var2></code>
---	--

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

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

with the result stored in *<tl var2>*.

<code>\siunitx_number_format:N</code> ☆	<code>\siunitx_number_format:N</code> $\langle number \rangle$
<code>\siunitx_number_format:NN</code> ☆	<code>\siunitx_number_format:NN</code> $\langle number \rangle$ $\langle marker \rangle$

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

In the `NN` version, the $\langle marker \rangle$ token is inserted at each possible alignment position in the output, *viz.* `TODO`

<code>\siunitx_if_number_p:n</code> ☆	<code>\siunitx_if_number_token:NTF</code> $\{\langle tokens \rangle\}$
<code>\siunitx_if_number:nTF</code> ☆	$\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

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

<code>\siunitx_if_number_token:NTF</code>	<code>\siunitx_if_number_token:NTF</code> $\{\langle token \rangle\}$
	$\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

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

<code>\l_siunitx_number_parse_bool</code>

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

<code>\l_siunitx_number_input_decimal_tl</code>
<code>\l_siunitx_number_output_decimal_tl</code>

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

0.1 Key-value options

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

<code>bracket-negative</code>	<code>bracket-negative = true false</code>
<code>drop-exponent</code>	<code>drop-exponent = true false</code>
<code>drop-uncertainty</code>	<code>drop-uncertainty = true false</code>
<code>drop-zero-decimal</code>	<code>drop-zero-decimal = true false</code>
<code>evaluate-expression</code>	<code>evaluate-expression = true false</code>
<code>exponent-base</code>	<code>exponent-base = $\langle base \rangle$</code>
<code>exponent-mode</code>	<code>exponent-mode = engineering fixed input scientific</code>

<u>exponent-product</u>	exponent-product = $\langle symbol \rangle$
<u>expression</u>	expression = $\langle expression \rangle$
<u>fixed-exponent</u>	fixed-exponent = $\langle exponent \rangle$
<u>group-digits</u>	group-digits = all decimal integer none
<u>group-minimum-digits</u>	group-minimum-digits = $\langle value \rangle$
<u>group-separator</u>	group-separator = $\langle symbol \rangle$
<u>input-close-uncertainty</u>	input-close-uncertainty = $\langle tokens \rangle$
<u>input-comparators</u>	input-comparators = $\langle tokens \rangle$
<u>input-close-uncertainty</u>	input-close-uncertainty = $\langle tokens \rangle$
<u>input-decimal-markers</u>	input-decimal-markers = $\langle tokens \rangle$
<u>input-digits</u>	input-digits = $\langle tokens \rangle$
<u>input-exponent-markers</u>	input-exponent-markers = $\langle tokens \rangle$
<u>input-open-uncertainty</u>	input-open-uncertainty = $\langle tokens \rangle$
<u>input-signs</u>	input-signs = $\langle tokens \rangle$
<u>input-uncertainty-signs</u>	input-uncertainty-signs = $\langle tokens \rangle$
<u>minimum-decimal-digits</u>	minimum-decimal-digits = $\langle min \rangle$
<u>minimum-integer-digits</u>	minimum-integer-digits = $\langle min \rangle$
<u>negative-color</u>	negative-color = $\langle color \rangle$

<u>number-close-bracket</u>	number-close-bracket = $\langle symbol \rangle$
<u>number-open-bracket</u>	number-open-bracket = $\langle symbol \rangle$
<u>output-close-uncertainty</u>	output-close-uncertainty = $\langle symbol \rangle$
<u>output-decimal-marker</u>	output-decimal-marker = $\langle symbol \rangle$
<u>output-open-uncertainty</u>	output-open-uncertainty = $\langle symbol \rangle$
<u>parse-numbers</u>	parse-numbers = true false
<u>print-implicit-plus</u>	print-implicit-plus = true false
<u>print-unity-mantissa</u>	print-unity-mantissa = true false
<u>print-zero-exponent</u>	print-zero-exponent = true false
<u>round-half</u>	round-half = even up
<u>round-minimum</u>	round-minimum = $\langle min \rangle$
<u>round-mode</u>	round-mode = figures none places uncertainty
<u>round-pad</u>	round-pad = true false
<u>round-precision</u>	round-precision = $\langle precision \rangle$
<u>separate-uncertainty</u>	separate-uncertainty = true false
<u>track-explicit-plus</u>	track-explicit-plus = true false
<u>uncertainty-separator</u>	uncertainty-separator = $\langle separator \rangle$
<u>tight-spacing</u>	tight-spacing = true false

1 siunitx-number 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_number>
```

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

`\l__siunitx_number_tmp_tl` Scratch space.

```
7 \tl_new:N \l__siunitx_number_tmp_tl
```

(End definition for `\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.

```
8 \tl_new:N \l_siunitx_number_formatted_tl
```

(End definition for `\l_siunitx_number_formatted_tl`.)

`\l_siunitx_number_parse_bool` Tracks whether to parse numbers: public as this may affect other behaviors.

```
9 \tl_new:N \l_siunitx_number_parse_bool
```

(End definition for `\l_siunitx_number_parse_bool`. This variable is documented on page 13.)

`\l_siunitx_number_parse_bool` Top-level options.

```
10 \keys_define:nn { siunitx }
11 {
12   parse-numbers .bool_set:N = \l_siunitx_number_parse_bool
13 }
```

(End definition for `\l_siunitx_number_parse_bool`. This variable is documented on page 13.)

`\siunitx_number_format:nN`

`\siunitx_number_format:VN`

`__siunitx_number_format:nN`

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

```
15 {
```

```
16   \group_begin:
```

```
17   \bool_if:NTF \l_siunitx_number_parse_bool
```

```
18   {
```

```
19     \siunitx_number_parse:nN {#1} \l__siunitx_number_parsed_tl
```

```
20     \siunitx_number_process:NN \l__siunitx_number_parsed_tl \l__siunitx_number_parsed_tl
```

```
21     \tl_set:Nx \l_siunitx_number_formatted_tl
```

```
22     { \siunitx_number_format:N \l__siunitx_number_parsed_tl }
```

```
23   }
```

```

24      { \tl_set:Nn \l__siunitx_number_formatted_tl { \ensuremath {#1} } }
25      \exp_args:NNNV \group_end:
26      \tl_set:Nn #2 \l__siunitx_number_formatted_tl
27    }
28    \cs_generate_variant:Nn \siunitx_number_format:nN { V }

```

(End definition for `\siunitx_number_format:nN` and `__siunitx_number_format:nN`. This function is documented on page 12.)

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 T_EX’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 T_EX 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).

In this release, the structure of a valid number is:

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

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 `\integer` and `\exponent` parts.

`\l_siunitx_number_input_decimal_tl` The input decimal markers(s).

```

29 \tl_new:N \l_siunitx_number_input_decimal_tl

```

(End definition for `\l_siunitx_number_input_decimal_tl`. This variable is documented on page 13.)

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

```

\l_siunitx_number_input_uncert_close_tl
\l_siunitx_number_input_comparator_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_explicit_plus_bool
\l_siunitx_number_expression:nN

```

```

30 \keys_define:nn { siunitx }
31 {
32   evaluate-expression .bool_set:N =
33     \l_siunitx_number_expression_bool ,
34   expression .code:n =
35     \cs_set:Npn \__siunitx_number_expression:n ##1 {#1} ,
36   input-close-uncertainty .tl_set:N =
37     \l_siunitx_number_input_uncert_close_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 parse-numbers .bool_set:N =
62   \l__siunitx_number_parse_bool ,
63 track-explicit-plus .bool_set:N =
64   \l__siunitx_number_explicit_plus_bool
65 }
66 \cs_new:Npn \__siunitx_number_expression:n #1 { }
67 \tl_new:N \l__siunitx_number_input_uncert_sign_tl

```

(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.

```
68 \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.

```
69 \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.

```
70 \tl_new:N \l__siunitx_number_exponent_tl
```

(End definition for \l__siunitx_number_exponent_tl.)

`\l__siunitx_number_flex_tl` In a number with an uncertainty, the exact meaning of a second part is not fully resolved until parsing is complete. That is handled using this “flexible” store.

```
71 \tl_new:N \l__siunitx_number_flex_tl
```

(End definition for \l__siunitx_number_flex_tl.)

`\l__siunitx_number_parsed_tl` The number parsed into internal format.

```
72 \tl_new:N \l__siunitx_number_parsed_tl
```

(End definition for \l__siunitx_number_parsed_tl.)

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

```

73 \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.

```

74 \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.

```

75 \bool_new:N \l__siunitx_number_validate_bool

(End definition for \l__siunitx_number_validate_bool.)

```

`\siunitx_number_parse:nN` 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. 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.

```

76 \cs_new_protected:Npn \siunitx_number_parse:nN #1#2
77 {
78   \bool_if:NTF \l__siunitx_number_parse_bool
79     { \__siunitx_number_parse:nN {#1} #2 }
80     { \tl_clear:N #2 }
81 }
82 \cs_new_protected:Npn \__siunitx_number_parse:nN #1#2
83 {
84   \group_begin:
85     \tl_clear:N \l__siunitx_number_parsed_tl
86     \protected@edef \l__siunitx_number_arg_tl
87       {
88         \bool_if:NTF \l__siunitx_number_expression_bool
89           { \fp_eval:n { \__siunitx_number_expression:n {#1} } }
90           {#1}
91       }
92     \tl_set_eq:NN \l__siunitx_number_input_tl \l__siunitx_number_arg_tl
93     \__siunitx_number_parse_replace:
94     \tl_if_empty:NF \l__siunitx_number_arg_tl
95       { \__siunitx_number_parse_comparator: }
96     \__siunitx_number_parse_check:
97     \exp_args:NNNV \group_end:
98     \tl_set:Nn #2 \l__siunitx_number_parsed_tl
99   }

```

(End definition for `\siunitx_number_parse:nN` and `__siunitx_number_parse:nN`. This function is documented on page 12.)

`__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

```

100 \cs_new_protected:Npn \__siunitx_number_parse_check:
101 {

```

```

102 \tl_if_empty:NF \l__siunitx_number_flex_tl
103 {
104   \bool_lazy_and:nnTF
105   {
106     \tl_if_blank_p:f
107     { \exp_after:wN \use_iv:nnnn \l__siunitx_number_parsed_tl }
108   }
109   {
110     \tl_if_blank_p:f
111     { \exp_after:wN \use_iv:nnnn \l__siunitx_number_flex_tl }
112   }
113   {
114     \tl_set:Nx \l__siunitx_number_tmp_tl
115     { \exp_after:wN \use_i:nnnn \l__siunitx_number_flex_tl }
116     \tl_if_in:NVTF \l__siunitx_number_input_uncert_sign_tl
117     \l__siunitx_number_tmp_tl
118     { \__siunitx_number_parse_combine_uncert: }
119     { \tl_clear:N \l__siunitx_number_parsed_tl }
120   }
121   { \tl_clear:N \l__siunitx_number_parsed_tl }
122 }
123 \tl_if_empty:NTF \l__siunitx_number_parsed_tl
124 {
125   \bool_if:NF \l__siunitx_number_validate_bool
126   {
127     \msg_error:nnx { siunitx } { invalid-number }
128     { \exp_not:V \l__siunitx_number_input_tl }
129   }
130 }
131 { \__siunitx_number_parse_finalise: }
132 }

```

(End definition for `__siunitx_number_parse_check:.`)

```

\__siunitx_number_parse_combine_uncert:
\__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_auxv:w
\__siunitx_number_parse_combine_uncert_auxvi:w

```

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 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.

```

133 \cs_new_protected:Npn \__siunitx_number_parse_combine_uncert:
134 {
135   \exp_after:wN \exp_after:wN \exp_after:wN
136   \__siunitx_number_parse_combine_uncert_auxi:nnnnnnnn
137   \exp_after:wN \l__siunitx_number_parsed_tl \l__siunitx_number_flex_tl
138 }

```

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_parsed_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.

```

139 \cs_new_protected:Npn
140   \__siunitx_number_parse_combine_uncert_auxi:nnnnnnnn #1#2#3#4#5#6#7#8

```

```

141 {
142   \int_compare:nNnTF { \tl_count:n {#6} } > { \tl_count:n {#2} }
143   {
144     \tl_clear:N \l__siunitx_number_parsed_tl
145     \tl_clear:N \l__siunitx_number_flex_tl
146   }
147   {
148     \__siunitx_number_parse_combine_uncert_auxii:fnnnn
149     { \int_eval:n { \tl_count:n {#3} - \tl_count:n {#7} } }
150     {#2} {#3} {#6} {#7}
151   }
152 }
153 \cs_new_protected:Npn
154   \__siunitx_number_parse_combine_uncert_auxii:nnnnn #1
155   {
156     \__siunitx_number_parse_combine_uncert_auxiii:fnnnnn
157     { \prg_replicate:nn { \int_abs:n {#1} } { 0 } }
158     {#1}
159   }
160 \cs_generate_variant:Nn \__siunitx_number_parse_combine_uncert_auxii:nnnnn { f }
161 \cs_new_protected:Npn
162   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn #1#2#3#4#5#6
163   {
164     \int_compare:nNnTF {#2} > 0
165     {
166       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
167       {#3} {#4} {#5} { #6 #1 }
168     }
169     {
170       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
171       {#3} { #4 #1 } {#5} {#6}
172     }
173   }
174 \cs_generate_variant:Nn
175   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn { f }
176 \cs_new_protected:Npn
177   \__siunitx_number_parse_combine_uncert_auxiv:nnnn #1#2#3#4
178   {
179     \tl_set:Nx \l__siunitx_number_parsed_tl
180     {
181       { \tl_head:V \l__siunitx_number_parsed_tl }
182       { \exp_not:n {#1} }
183     }
184     \bool_lazy_and:nnTF
185       { \tl_if_blank_p:n {#2} }
186       { ! \tl_if_blank_p:n {#4} }
187       { 0 }
188       { \exp_not:n {#2} }
189   }
190   {
191     \__siunitx_number_parse_combine_uncert_auxv:w #3#4
192     \q_recursion_tail \q_recursion_stop
193   }
194 }

```



```
195 }
```

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

```
196 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxv:w #1
197 {
198   \quark_if_recursion_tail_stop:N #1
199   \str_if_eq:nnTF {#1} { 0 }
200     { \__siunitx_number_parse_combine_uncert_auxv:w }
201     { \__siunitx_number_parse_combine_uncert_auxvi:w #1 }
202 }
203 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxvi:w
204 #1 \q_recursion_tail \q_recursion_stop
205 {
206   \tl_if_blank:nF {#1}
207     { { S } { \exp_not:n {#1} } }
208 }
```

(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.

```
209 \cs_new_protected:Npn \__siunitx_number_parse_comparator:
210 {
211   \exp_after:wN \__siunitx_number_parse_comparator_aux:Nw
212   \l__siunitx_number_arg_tl \q_stop
213 }
214 \cs_new_protected:Npn \__siunitx_number_parse_comparator_aux:Nw #1#2 \q_stop
215 {
216   \tl_if_in:NnTF \l__siunitx_number_input_comparator_tl {#1}
217     {
218       \tl_set:Nn \l__siunitx_number_comparator_tl {#1}
219       \tl_set:Nn \l__siunitx_number_arg_tl {#2}
220     }
221     { \tl_clear:N \l__siunitx_number_comparator_tl }
222   \tl_if_empty:NF \l__siunitx_number_arg_tl
223     { \__siunitx_number_parse_sign: }
224 }
```

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

```
\__siunitx_number_parse_exponent:
\__siunitx_number_parse_exponent_auxi:w
\__siunitx_number_parse_exponent_auxii:nn
\__siunitx_number_parse_exponent_auxiii:Nw
\__siunitx_number_parse_exponent_auxiv:nn
\__siunitx_number_parse_exponent_zero_test:N
\__siunitx_number_parse_exponent_check:N
\__siunitx_number_parse_exponent_cleanup:N
```

An exponent part of a number has to come at the end and can only occur once. Thus it is relatively easy to parse. First, there is a check that an exponent part is allowed, and if so a split is made (the previous part of the chain checks that there is some content in `\l__siunitx_number_arg_tl` before calling this function). After splitting, if there is no exponent then simply save a default. Otherwise, check for a sign and then store either this or an implicit plus, and the digits after a check that nothing else is present after the `e`. The only slight complication to all of this is allowing an arbitrary token in the input to 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.

```
225 \cs_new_protected:Npn \__siunitx_number_parse_exponent:
```

```

226 {
227   \tl_if_empty:NTF \l__siunitx_number_input_exponent_tl
228   {
229     \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 }
230     \tl_if_empty:NF \l__siunitx_number_parsed_tl
231     { \__siunitx_number_parse_loop: }
232   }
233   {
234     \tl_set:Nx \l__siunitx_number_tmp_tl
235     { \tl_head:V \l__siunitx_number_input_exponent_tl }
236     \tl_map_inline:Nn \l__siunitx_number_input_exponent_tl
237     {
238       \tl_replace_all:NnV \l__siunitx_number_arg_tl
239       {##1} \l__siunitx_number_tmp_tl
240     }
241     \use:x
242     {
243       \cs_set_protected:Npn
244       \exp_not:N \__siunitx_number_parse_exponent_auxi:w
245       #####1 \exp_not:V \l__siunitx_number_tmp_tl
246       #####2 \exp_not:V \l__siunitx_number_tmp_tl
247       #####3 \exp_not:N \q_stop
248     }
249     { \__siunitx_number_parse_exponent_auxii:nn {##1} {##2} }
250     \use:x
251     {
252       \__siunitx_number_parse_exponent_auxi:w
253       \exp_not:V \l__siunitx_number_arg_tl
254       \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_nil
255       \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_stop
256     }
257   }
258 }
259 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxi:w { }
260 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxii:nn #1#2
261 {
262   \quark_if_nil:nTF {#2}
263   { \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 } }
264   {
265     \tl_set:Nn \l__siunitx_number_arg_tl {#1}
266     \tl_if_blank:nTF {#2}
267     { \tl_clear:N \l__siunitx_number_parsed_tl }
268     { \__siunitx_number_parse_exponent_auxiii:Nw #2 \q_stop }
269   }
270   \tl_if_empty:NF \l__siunitx_number_parsed_tl
271   { \__siunitx_number_parse_loop: }
272 }
273 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxiii:Nw #1#2 \q_stop
274 {
275   \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#1}
276   { \__siunitx_number_parse_exponent_auxiv:nn {#1} {#2} }
277   { \__siunitx_number_parse_exponent_auxiv:nn { } {#1#2} }
278   \tl_if_empty:NT \l__siunitx_number_exponent_tl
279   { \tl_clear:N \l__siunitx_number_parsed_tl }

```

```

280 }
281 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxiv:nn #1#2
282 {
283   \bool_lazy_or:nnTF
284     { \l__siunitx_number_explicit_plus_bool }
285     { ! \str_if_eq_p:nn {#1} { + } }
286     { \tl_set:Nn \l__siunitx_number_exponent_tl { {#1} } }
287     { \tl_set:Nn \l__siunitx_number_exponent_tl { { } } }
288   \tl_if_blank:nTF {#2}
289     { \tl_clear:N \l__siunitx_number_parsed_tl }
290     {
291       \__siunitx_number_parse_exponent_zero_test:N #2
292       \q_recursion_tail \q_recursion_stop
293     }
294 }
295 \cs_new_protected:Npn \__siunitx_number_parse_exponent_zero_test:N #1
296 {
297   \quark_if_recursion_tail_stop_do:Nn #1
298   { \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 } }
299   \str_if_eq:nnTF {#1} { 0 }
300   { \__siunitx_number_parse_exponent_zero_test:N }
301   { \__siunitx_number_parse_exponent_check:N #1 }
302 }
303 \cs_new_protected:Npn \__siunitx_number_parse_exponent_check:N #1
304 {
305   \quark_if_recursion_tail_stop:N #1
306   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#1}
307   {
308     \tl_put_right:Nn \l__siunitx_number_exponent_tl {#1}
309     \__siunitx_number_parse_exponent_check:N
310   }
311   { \__siunitx_number_parse_exponent_cleanup:wN }
312 }
313 \cs_new_protected:Npn \__siunitx_number_parse_exponent_cleanup:wN
314 #1 \q_recursion_stop
315 { \tl_clear:N \l__siunitx_number_parsed_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
316 \cs_new_protected:Npn \__siunitx_number_parse_replace:
317 {
318   \__siunitx_number_parse_replace_minus:
319   \exp_last_unbraced:NV \__siunitx_number_parse_replace_aux:nN
320   \c__siunitx_number_parse_sign_replacement_tl
321   { ? } \q_recursion_tail
322   \q_recursion_stop
323 }
324 \cs_set_protected:Npn \__siunitx_number_parse_replace_aux:nN #1#2
325 {
326   \quark_if_recursion_tail_stop:N #2
327   \tl_replace_all:Nnn \l__siunitx_number_arg_tl {#1} {#2}
328   \__siunitx_number_parse_replace_aux:nN

```

```

329 }
330 \tl_const:Nn \c__siunitx_number_parse_sign_replacement_tl
331 {
332   { +- } \mp
333   { +- } \pm
334   { << } \ll
335   { <= } \le
336   { >> } \gg
337   { >= } \ge
338 }
339 \group_begin:
340   \char_set_catcode_active:N \-
341   \cs_new_protected:Npx \__siunitx_number_parse_replace_minus:
342   {
343     \tl_replace_all:Nnn \exp_not:N \l__siunitx_number_arg_tl
344     { \exp_not:N - } { \token_to_str:N - }
345   }
346 \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:nw

```

347 \cs_new_protected:Npn \__siunitx_number_parse_finalise:
348 {
349   \tl_if_empty:NF \l__siunitx_number_parsed_tl
350   {
351     \tl_set:Nx \l__siunitx_number_parsed_tl
352     {
353       { \exp_not:V \l__siunitx_number_comparator_tl }
354       { \exp_not:V \l__siunitx_number_parsed_tl }
355       { \exp_after:wN \__siunitx_number_parse_finalise:nw
356         \l__siunitx_number_exponent_tl \q_stop }
357     }
358   }
359 }
360 \cs_new:Npn \__siunitx_number_parse_finalise:nw #1#2 \q_stop
361 {
362   { \exp_not:n {#1} }
363   { \exp_not:n {#2} }
364 }

```

(End definition for __siunitx_number_parse_finalise: and __siunitx_number_parse_finalise: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_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:N
_siunitx_number_parse_loop_root_swap:NNwNN
_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.

```

365 \cs_new_protected:Npn \__siunitx_number_parse_loop:
366 {
367   \tl_clear:N \l__siunitx_number_partial_tl
368   \exp_after:wN \__siunitx_number_parse_loop_first:NNN

```

```

369     \exp_after:wN \l__siunitx_number_parsed_tl \exp_after:wN \c_true_bool
370     \l__siunitx_number_arg_tl
371     \q_recursion_tail \q_recursion_stop
372 }

```

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.

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

```

373 \cs_new_protected:Npn \__siunitx_number_parse_loop_first:NNN #1#2#3
374 {
375   \quark_if_recursion_tail_stop_do:Nn #3
376   {
377     \bool_if:NTF #2
378     { \tl_put_right:Nn #1 { { 1 } { } { } } }
379     { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
380   }
381   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#3}
382   {
383     \__siunitx_number_parse_loop_main:NNNNN
384     #1 \c_true_bool \c_false_bool #2 #3
385   }
386   {
387     \tl_if_in:NnTF \l_siunitx_number_input_decimal_tl {#3}
388     {
389       \tl_put_right:Nn #1 { { 0 } }
390       \__siunitx_number_parse_loop_after_decimal:NNN #1 #2
391     }
392     { \__siunitx_number_parse_loop_break:wN }
393   }
394 }

```

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 sign token (if allowed): stop collecting this number and restart collection for the second part.

```

395 \cs_new_protected:Npn \__siunitx_number_parse_loop_main:NNNNN #1#2#3#4#5
396 {
397   \quark_if_recursion_tail_stop_do:Nn #5
398   { \__siunitx_number_parse_loop_main_end:NN #1#2 }
399   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#5}
400   { \__siunitx_number_parse_loop_main_digit:NNNNN #1#2#3#4#5 }
401   {
402     \tl_if_in:NnTF \l__siunitx_number_input_decimal_tl {#5}
403     {
404       \bool_if:NTF #2
405       { \__siunitx_number_parse_loop_main_decimal:NN #1 #4 }
406       { \__siunitx_number_parse_loop_break:wN }
407     }
408     {
409       \tl_if_in:NnTF \l__siunitx_number_input_uncert_open_tl {#5}
410       { \__siunitx_number_parse_loop_main_uncert:NNN #1#2 #4 }
411       {
412         \bool_if:NTF #4
413         {
414           \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#5}
415           {
416             \__siunitx_number_parse_loop_main_sign:NNN
417             #1#2 #5
418           }
419           { \__siunitx_number_parse_loop_break:wN }
420         }
421         { \__siunitx_number_parse_loop_break:wN }
422       }
423     }
424   }
425 }

```

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.

```

426 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_end:NN #1#2
427 {
428   \bool_lazy_and:nnT
429   {#2} { \tl_if_empty_p:N \l__siunitx_number_partial_tl }
430   { \tl_set:Nn \l__siunitx_number_partial_tl { 0 } }
431   \tl_put_right:Nx #1
432   {
433     { \exp_not:V \l__siunitx_number_partial_tl }
434     \bool_if:NT #2 { { } }
435     { }
436   }
437 }

```

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.

```

438 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_digit:NNNNN #1#2#3#4#5
439 {
440   \bool_lazy_or:nnTF
441     {#3} { ! \str_if_eq_p:nn {#5} { 0 } }
442     {
443       \tl_put_right:Nn \l__siunitx_number_partial_tl {#5}
444       \__siunitx_number_parse_loop_main:NNNNN #1 #2 \c_true_bool #4
445     }
446     { \__siunitx_number_parse_loop_main:NNNNN #1 #2 \c_false_bool #4 }
447 }

```

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.

```

448 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_decimal:NN #1#2
449 {
450   \__siunitx_number_parse_loop_main_store:NNN #1 \c_false_bool \c_false_bool
451   \__siunitx_number_parse_loop_after_decimal:NNN #1 #2
452 }

```

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.

```

453 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_uncert:NNN #1#2#3
454 {
455   \__siunitx_number_parse_loop_main_store:NNN #1 #2 \c_false_bool
456   \__siunitx_number_parse_loop_uncert:NNNNN
457   #1 \c_true_bool \c_false_bool #3
458 }

```

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.

```

459 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_sign:NNN #1#2#3
460 {
461   \__siunitx_number_parse_loop_main_store:NNN #1 #2 \c_true_bool
462   \tl_set:Nn \l__siunitx_number_flex_tl { {#3} }
463   \__siunitx_number_parse_loop_first:NNN
464   \l__siunitx_number_flex_tl \c_false_bool
465 }

```

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.

```

466 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_store:NNN #1#2#3
467 {
468   \tl_if_empty:NT \l__siunitx_number_partial_tl
469   { \tl_set:Nn \l__siunitx_number_partial_tl { 0 } }
470   \tl_put_right:Nx #1
471   {
472     { \exp_not:V \l__siunitx_number_partial_tl }

```

```

473     \bool_if:NT #2 { { } }
474     \bool_if:NT #3 { { } }
475   }
476   \tl_clear:N \l__siunitx_number_partial_tl
477 }

```

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.

```

478 \cs_new_protected:Npn \__siunitx_number_parse_loop_after_decimal:NNN #1#2#3
479 {
480   \tl_if_blank:FTF { \exp_after:wN \use_none:n #1 }
481   {
482     \quark_if_recursion_tail_stop_do:Nn #3
483     { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
484     \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#1}
485     {
486       \tl_put_right:Nn \l__siunitx_number_partial_tl {#3}
487       \__siunitx_number_parse_loop_main:NNNNN
488       #1 \c_false_bool \c_true_bool #2
489     }
490     { \__siunitx_number_parse_loop_break:wN }
491   }
492   {
493     \__siunitx_number_parse_loop_main:NNNNN
494     #1 \c_false_bool \c_true_bool #2 #3
495   }
496 }

```

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).

```

497 \cs_new_protected:Npn \__siunitx_number_parse_loop_uncert:NNNNN #1#2#3#4#5
498 {
499   \quark_if_recursion_tail_stop_do:Nn #5
500   { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
501   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#5}
502   {
503     \bool_lazy_or:nnTF
504     {#3} { ! \str_if_eq_p:nn {#5} { 0 } }
505     {
506       \tl_put_right:Nn \l__siunitx_number_partial_tl {#5}
507       \__siunitx_number_parse_loop_uncert:NNNNN
508       #1 \c_false_bool \c_true_bool #4
509     }
510     {
511       \__siunitx_number_parse_loop_uncert:NNNNN
512       #1 \c_false_bool \c_false_bool #4
513     }
514   }
515   {
516     \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#5}
517     {

```



```

518         \bool_if:NTF #2
519         { \__siunitx_number_parse_loop_break:wN }
520         {
521             \tl_if_empty:NTF \l__siunitx_number_partial_tl
522             { \tl_put_right:Nx #1 { { } } }
523             {
524                 \tl_set:Nx \l__siunitx_number_partial_tl
525                 { { S } { \exp_not:V \l__siunitx_number_partial_tl } }
526                 \__siunitx_number_parse_loop_main_store:NNN #1
527                 \c_false_bool \c_false_bool
528             }
529             \__siunitx_number_parse_loop_after_uncert:N
530         }
531     }
532     { \__siunitx_number_parse_loop_break:wN }
533 }
534 }

```

No further tokens are allowed after an uncertainty in parenthesis.

```

535 \cs_new_protected:Npn \__siunitx_number_parse_loop_after_uncert:N #1
536 {
537     \quark_if_recursion_tail_stop:N #1
538     \__siunitx_number_parse_loop_break:wN
539 }

```

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.

```

540 \cs_new_protected:Npn \__siunitx_number_parse_loop_break:wN
541 #1 \q_recursion_stop
542 {
543     \tl_clear:N \l__siunitx_number_flex_tl
544     \tl_clear:N \l__siunitx_number_parsed_tl
545 }

```

(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. 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.

```

546 \cs_new_protected:Npn \__siunitx_number_parse_sign:
547 {
548     \exp_after:wN \__siunitx_number_parse_sign_aux:Nw
549     \l__siunitx_number_arg_tl \q_stop
550 }
551 \cs_new_protected:Npn \__siunitx_number_parse_sign_aux:Nw #1#2 \q_stop
552 {
553     \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#1}
554     {
555         \tl_set:Nn \l__siunitx_number_arg_tl {#2}
556         \bool_lazy_and:nnTF
557         { \token_if_eq_charcode_p:NN #1 + }
558         { ! \l__siunitx_number_explicit_plus_bool }
559         { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
560         { \tl_set:Nn \l__siunitx_number_parsed_tl { {#1} } }
561     }

```

```

562     { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
563     \tl_if_empty:NTF \l__siunitx_number_arg_tl
564     { \tl_clear:N \l__siunitx_number_parsed_tl }
565     { \__siunitx_number_parse_exponent: }
566 }

```

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

1.4 Processing numbers

```

\l_siunitx_number_drop_exponent_bool
\l_siunitx_number_drop_uncertainty_bool
\l_siunitx_number_drop_zero_decimal_bool
\l_siunitx_number_exponent_mode_tl
\l_siunitx_number_exponent_fixed_int
\l_siunitx_number_min_decimal_int
\l_siunitx_number_min_integer_int
\l_siunitx_number_round_half_even_bool
\l_siunitx_number_round_min_tl
\l_siunitx_number_round_mode_tl
\l_siunitx_number_round_pad_bool
\l_siunitx_number_round_precision_int
567 \keys_define:nn { siunitx }
568 {
569     drop-exponent .bool_set:N =
570     \l__siunitx_number_drop_exponent_bool ,
571     drop-uncertainty .bool_set:N =
572     \l__siunitx_number_drop_uncertainty_bool ,
573     drop-zero-decimal .bool_set:N =
574     \l_siunitx_number_drop_zero_decimal_bool ,
575     exponent-mode .choices:nn =
576     { engineering , fixed , input , scientific }
577     { \tl_set_eq:NN \l__siunitx_number_exponent_mode_tl \l_keys_choice_tl } ,
578     fixed-exponent .int_set:N =
579     \l_siunitx_number_exponent_fixed_int ,
580     minimum-decimal-digits .int_set:N =
581     \l_siunitx_number_min_decimal_int ,
582     minimum-integer-digits .int_set:N =
583     \l_siunitx_number_min_integer_int ,
584     round-half .choice: ,
585     round-half / even .code:n =
586     { \bool_set_true:N \l__siunitx_number_round_half_even_bool } ,
587     round-half / up .code:n =
588     { \bool_set_false:N \l__siunitx_number_round_half_even_bool } ,
589     round-minimum .tl_set:N =
590     \l_siunitx_number_round_min_tl ,
591     round-mode .choices:nn =
592     { figures , none , places , uncertainty }
593     { \tl_set_eq:NN \l__siunitx_number_round_mode_tl \l_keys_choice_tl } ,
594     round-pad .bool_set:N =
595     \l_siunitx_number_round_pad_bool ,
596     round-precision .int_set:N =
597     \l_siunitx_number_round_precision_int ,
598 }
599 \bool_new:N \l__siunitx_number_round_half_even_bool
600 \tl_new:N \l__siunitx_number_exponent_mode_tl
601 \tl_new:N \l__siunitx_number_round_mode_tl

```

(End definition for \l__siunitx_number_drop_exponent_bool and others.)

\siunitx_number_process:NN

A top-level interface for the processing tools.

__siunitx_number_process:nnnnnnnn

```

602 \cs_new_protected:Npn \siunitx_number_process:NN #1#2
603 {
604     \tl_if_empty:NF #1
605     {

```

```

606     \_siunitx_number_drop_uncertainty:NN #1 #2
607     \exp_after:wN \_siunitx_number_process:nnnnnnn #2 #2 #2
608     \_siunitx_number_drop_exponent:NN #2 #2
609     \_siunitx_number_zero_decimal:NN #2 #2
610     \_siunitx_number_digits:NN #2 #2
611 }
612 }
613 \cs_new_protected:Npn \_siunitx_number_process:nnnnnnn #1#2#3#4#5#6#7#8#9
614 {
615     \bool_lazy_and:nnF
616     { \str_if_eq_p:nn {#3} { 0 } }
617     {
618         \str_if_eq_x_p:nn
619         { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
620     }
621     {
622         \_siunitx_number_exponent:NN #8 #9
623         \_siunitx_number_round:NN #9 #9
624     }
625 }

```

(End definition for `\siunitx_number_process:NN` and `_siunitx_number_process:nnnnnnn`. This function is documented on page 12.)

```

\_siunitx_number_exponent:NN
siunitx_number_exponent_engineering:nnnnnnn
\_siunitx_number_exponent_fixed:nnnnnnn
\_siunitx_number_exponent_input:nnnnnnn
\_siunitx_number_exponent_scientific:nnnnnnn
\_siunitx_number_exponent_scientific:nnnw
\_siunitx_number_exponent_shift:nnn
\_siunitx_number_exponent_shift:nnf
\_siunitx_number_exponent_shift_down:nnnw
\_siunitx_number_exponent_shift_down:nnn
\_siunitx_number_exponent_shift_down:nw
\_siunitx_number_exponent_shift_up:nnn
\_siunitx_number_exponent_shift_up:nnw
\_siunitx_number_exponent_finalise:n
siunitx_number_exponent_engineering_aux:nnnnnnn
\_siunitx_number_exponent_engineering_0:nnnn
\_siunitx_number_exponent_engineering_1:nnnn
\_siunitx_number_exponent_engineering_2:nnnn
\_siunitx_number_exponent_engineering:nNw

```

Manipulating an exponent is done using a single expansion function *unless* dealing with engineering-style output. The latter is easier to handle by first converting to scientific output, then post-processing. (Once `e`-type expansion is generally available, this will be handling using a single `\tl_set:Nx`.)

```

626 \cs_new_protected:Npn \_siunitx_number_exponent:NN #1#2
627 {
628     \tl_set:Nx #2
629     {
630         \cs:w
631         \_siunitx_number_exponent_ \l__siunitx_number_exponent_mode_tl :nnnnnnn
632         \exp_after:wN
633         \cs_end: #1
634     }
635     \str_if_eq:VnT \l__siunitx_number_exponent_mode_tl { engineering }
636     {
637         \tl_set:Nx #1
638         { \exp_after:wN \_siunitx_number_exponent_engineering_aux:nnnnnnn #1 }
639     }
640 }
641 \cs_new:Npn \_siunitx_number_exponent_fixed:nnnnnnn #1#2#3#4#5#6#7
642 {
643     \exp_not:n { {#1} {#2} }
644     \exp_args:Nf \_siunitx_number_exponent_shift:nnn
645     { \int_eval:n { \l__siunitx_number_exponent_fixed_int - (#6#7) } }
646     {#3} {#4}
647     \exp_not:n { {#5} }
648     \exp_not:n { {#6} } { \int_use:N \l__siunitx_number_exponent_fixed_int }
649 }
650 \cs_new:Npn \_siunitx_number_exponent_input:nnnnnnn #1#2#3#4#5#6#7
651 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }

```

To convert to scientific notation, the key question is to find the number of significant places. That is easy enough if the number has a non-zero integer component. For a pure decimal, we have to trim off leading zeros in a loop.

```

652 \cs_new:Npn \__siunitx_number_exponent_scientific:nnnnnnn #1#2#3#4#5#6#7
653 {
654   \exp_not:n { {#1} {#2} }
655   \int_compare:nNnTF { \tl_count:n {#3} } = 1
656   {
657     \str_if_eq:nnTF {#3} { 0 }
658     {
659       \__siunitx_number_exponent_scientific:nnnw
660       { 0 } {#5} { #6#7 } #4 \q_stop
661     }
662     { \exp_not:n { {#3} {#4} {#5} {#6} {#7} } }
663   }
664   {
665     \__siunitx_number_exponent_shift:nnn { \tl_count:n {#3} - 1 } {#3} {#4}
666     \exp_not:n { {#5} }
667     \__siunitx_number_exponent_finalise:n { \tl_count:n {#3} + #6#7 - 1 }
668   }
669 }
670 \cs_new_eq:NN \__siunitx_number_exponent_engineering:nnnnnnn
671 \__siunitx_number_exponent_scientific:nnnnnnn
672 \cs_new:Npn \__siunitx_number_exponent_scientific:nnnw #1#2#3#4#5 \q_stop
673 {
674   \str_if_eq:nnTF {#4} { 0 }
675   {
676     \__siunitx_number_exponent_scientific:nnnw
677     { #1 - 1 } {#2} {#3} #5 \q_stop
678   }
679   {
680     \exp_not:n { {#4} {#5} {#2} }
681     \__siunitx_number_exponent_finalise:n { #1 + #3 - 1 }
682   }
683 }

```

When adjusting the exponent position, there are two paths depending on which way the shift takes place.

```

684 \cs_new:Npn \__siunitx_number_exponent_shift:nnn #1#2#3
685 {
686   \int_compare:nNnTF {#1} > 0
687   { \__siunitx_number_exponent_shift_down:nnnw {#1} {#3} { } #2 \q_stop }
688   {
689     \int_compare:nNnTF {#1} < 0
690     { \__siunitx_number_exponent_shift_up:nnw {#1} {#2} #3 \q_stop }
691     { {#2} {#3} }
692   }
693 }
694 \cs_generate_variant:Nn \__siunitx_number_exponent_shift:nnn { nnf }

```

For shifting the exponent down, there is first a loop to reserve the integer part before doing the work: that of course has to be undone for any remainder at the end of the process.

```

695 \cs_new:Npn \__siunitx_number_exponent_shift_down:nnnw #1#2#3#4#5 \q_stop

```

```

696 {
697   \tl_if_blank:nTF {#5}
698     { \__siunitx_number_exponent_shift_down:nnn {#1} { #4 #3 } {#2} }
699     { \__siunitx_number_exponent_shift_down:nnnw {#1} {#2} { #4 #3 } #5 \q_stop }
700   }
701 \cs_new:Npn \__siunitx_number_exponent_shift_down:nnn #1#2#3
702 {
703   \int_compare:nNnTF {#1} = 0
704     { { \tl_reverse:n {#2} } \exp_not:n { {#3} } }
705     { \__siunitx_number_exponent_shift_down:nw {#1} #2 \q_stop {#3} }
706   }
707 \cs_new:Npn \__siunitx_number_exponent_shift_down:nw #1#2#3 \q_stop #4
708 {
709   \tl_if_blank:nTF {#3}
710     { \__siunitx_number_exponent_shift_down:nnn { #1 - 1 } { 0 } { #2#4 } }
711     { \__siunitx_number_exponent_shift_down:nnn { #1 - 1 } {#3} { #2#4 } }
712   }

```

For shifting the exponent up, we can run out of decimal digits, at which point filling is easy. Other than that a simple loop as we are picking input off the front of the decimal part.

```

713 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnn #1#2#3
714 {
715   \int_compare:nNnTF {#1} = 0
716     { \exp_not:n { {#2} {#3} } }
717     {
718       \tl_if_blank:nTF {#3}
719         {
720           {
721             \exp_not:n {#2}
722             \prg_replicate:nn { \int_abs:n {#1} } { 0 }
723           }
724           { }
725         }
726         { \__siunitx_number_exponent_shift_up:nnw {#1} {#2} #3 \q_stop }
727       }
728   }
729 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnw #1 #2#3#4 \q_stop
730 { \__siunitx_number_exponent_shift_up:nnn { #1 + 1 } { #2 #3 } {#4} }

```

Tidy up the exponent to put the sign in the right place.

```

731 \cs_new:Npn \__siunitx_number_exponent_finalise:n #1
732 {
733   \int_compare:nNnTF {#1} > 0
734     { { } }
735     { { - } }
736     { \int_abs:n {#1} }
737   }

```

This could (and eventually will) be combined with the main function above: that will need e-type expansion. The input has already been normalised such that the integer part is in the range $1 \leq n < 10$. Thus there are only three cases to deal with, depending on the required adjustment to the exponent.

```

738 \cs_new:Npn \__siunitx_number_exponent_engineering_aux:nnnnnn #1#2#3#4#5#6#7
739 {

```

```

740 \exp_not:n { {#1} {#2} }
741 \use:c
742 {
743   __siunitx_number_exponent_engineering_
744   \int_compare:nNnTF {#6#7} < 0
745   {
746     \int_case:nnF { \int_mod:nn { #7 } { 3 } }
747     {
748       { 1 } { 2 }
749       { 2 } { 1 }
750     }
751     { 0 }
752   }
753   { \int_mod:nn {#7} { 3 } }
754   :nnnn
755 }
756 {#3} {#4} {#5} {#6#7}
757 }
758 \cs_new:cpn { __siunitx_number_exponent_engineering_0:nnnn } #1#2#3#4
759 {
760   \exp_not:n { {#1} {#2} {#3} }
761   __siunitx_number_exponent_finalise:n {#4}
762 }
763 \cs_new:cpn { __siunitx_number_exponent_engineering_1:nnnn } #1#2#3#4
764 {
765   \tl_if_blank:nTF {#2}
766   { { \exp_not:n { #1 0 } } { } }
767   {
768     { \exp_not:n {#1} \exp_not:o { \tl_head:w #2 \q_stop } }
769     { \exp_not:f { \tl_tail:n {#2} } }
770   }
771   \exp_not:n { {#3} }
772   __siunitx_number_exponent_finalise:n { #4 - 1 }
773 }
774 \cs_new:cpn { __siunitx_number_exponent_engineering_2:nnnn } #1#2#3#4
775 {
776   \tl_if_blank:nTF {#2}
777   { { \exp_not:n { #1 00 } } { } }
778   { \__siunitx_number_exponent_engineering:nNw {#1} #2 \q_stop }
779   \exp_not:n { {#3} }
780   __siunitx_number_exponent_finalise:n { #4 - 2 }
781 }
782 \cs_new:Npn \__siunitx_number_exponent_engineering:nNw #1#2#3 \q_stop
783 {
784   \tl_if_blank:nTF {#3}
785   { { \exp_not:n { #1#2 0 } } { } }
786   {
787     { \exp_not:n {#1#2} \exp_not:o { \tl_head:w #3 \q_stop } }
788     { \exp_not:f { \tl_tail:n {#3} } }
789   }
790 }

```

(End definition for `__siunitx_number_exponent:NN` and others.)

`__siunitx_number_digits:NN` Forcing a minimum number of digits in each part is quite easy. As the common case is
`__siunitx_number_digits:nnnnnn`
`__siunitx_number_digits:Nn`

that we don't do anything here, there is no real need to optimise the calculation (normally also numbers have only a few digits).

```

791 \cs_new_protected:Npn \__siunitx_number_digits:NN #1#2
792 {
793   \tl_set:Nx #2
794   { \exp_after:wN \__siunitx_number_digits:nnnnnnn #1 }
795 }
796 \cs_new:Npn \__siunitx_number_digits:nnnnnnn #1#2#3#4#5#6#7
797 {
798   \exp_not:n { {#1} {#2} }
799   {
800     \__siunitx_number_digits:Nn \l__siunitx_number_min_integer_int {#3}
801     \exp_not:n {#3}
802   }
803   {
804     \exp_not:n {#4}
805     \__siunitx_number_digits:Nn \l__siunitx_number_min_decimal_int {#4}
806   }
807   \exp_not:n { {#5} {#6} {#7} }
808 }
809 \cs_new:Npn \__siunitx_number_digits:Nn #1#2
810 {
811   \int_compare:nNnT
812     { #1 - \tl_count:n {#2} } > 0
813     { \prg_replicate:nn { #1 - \tl_count:n {#2} } { 0 } }
814 }

```

(End definition for __siunitx_number_digits:NN, __siunitx_number_digits:nnnnnnn, and __siunitx_number_digits:Nn.)

Simple stripping of the exponent.

```

815 \cs_new_protected:Npn \__siunitx_number_drop_exponent:NN #1#2
816 {
817   \bool_if:NT \l__siunitx_number_drop_exponent_bool
818   {
819     \tl_set:Nx #2
820     { \exp_after:wN \__siunitx_number_drop_exponent:nnnnnnn #1 }
821   }
822 }
823 \cs_new:Npn \__siunitx_number_drop_exponent:nnnnnnn #1#2#3#4#5#6#7
824 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} { } { 0 } } }

```

(End definition for __siunitx_number_drop_exponent:NN and __siunitx_number_drop_exponent:nnnnnnn.)

Simple stripping of the uncertainty.

```

825 \cs_new_protected:Npn \__siunitx_number_drop_uncertainty:NN #1#2
826 {
827   \bool_if:NT \l__siunitx_number_drop_uncertainty_bool
828   {
829     \tl_set:Nx #2
830     { \exp_after:wN \__siunitx_number_drop_uncertainty:nnnnnnn #1 }
831   }
832 }
833 \cs_new:Npn \__siunitx_number_drop_uncertainty:nnnnnnn #1#2#3#4#5#6#7
834 { \exp_not:n { {#1} {#2} {#3} {#4} { } {#6} {#7} } }

```

(End definition for `__siunitx_number_drop_uncertainty:NN` and `__siunitx_number_drop_uncertainty:nnnnnnn`.)

`__siunitx_number_round:NN`
`__siunitx_number_round_none:nnnnnnn`

Rounding is at the top level simple enough: fire off the expandable set up which does the work.

```

835 \cs_new_protected:Npn \__siunitx_number_round:NN #1#2
836 {
837   \tl_set:Nx #2
838   {
839     \cs:w
840       __siunitx_number_round_ \l__siunitx_number_round_mode_tl :nnnnnnn
841     \exp_after:wN
842     \cs_end: #1
843   }
844 }
845 \cs_new:Npn \__siunitx_number_round_none:nnnnnnn #1#2#3#4#5#6#7
846 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }

```

(End definition for `__siunitx_number_round:NN` and `__siunitx_number_round_none:nnnnnnn`.)

`__siunitx_number_round:nnn`
`__siunitx_number_round:fnn`
`__siunitx_number_round_auxi:nnnN`
`__siunitx_number_round_auxii:nnnN`
`__siunitx_number_round_auxiii:nnnN`
`__siunitx_number_round_auxiv:nnN`
`__siunitx_number_round_auxv:nnN`
`__siunitx_number_round_auxvi:nnN`
`__siunitx_number_round_auxvii:nnN`
`__siunitx_number_round_auxviii:nnN`
`__siunitx_number_round_final_integer:nnw`
`__siunitx_number_round_final_decimal:nnw`
`__siunitx_number_round_final_output:nn`
`__siunitx_number_round_final_output:ff`
`__siunitx_number_round_final:nn`
`__siunitx_number_round_final:fn`
`__siunitx_number_round_final_shift:nn`
`__siunitx_number_round_final_shift:ff`
`__siunitx_number_round_final_shift:Nw`
`__siunitx_number_round_engineering:nn`
`__siunitx_number_round_fixed:nn`
`__siunitx_number_round_input:nn`
`__siunitx_number_round_scientific:nn`
`__siunitx_number_round_engineering:NNNNn`
`__siunitx_number_round_engineering:nnN`
`__siunitx_number_round_truncate:n`
`__siunitx_number_round_truncate_direct:n`
`__siunitx_number_round_truncate:nnN`

Actually doing the rounding needs us to work from the least significant digit, so we start by reversing the input. We *could* also drop digits in this phase, but tracking everything would be horrible, so we go slightly slower but clearer and split the steps. First we reverse the decimal part, then the integer.

```

847 \cs_new:Npn \__siunitx_number_round:nnn #1#2#3
848 {
849   \__siunitx_number_round_auxi:nnnN {#1} {#2} { }
850   #3 \q_recursion_tail \q_recursion_stop
851 }
852 \cs_generate_variant:Nn \__siunitx_number_round:nnn { f }
853 \cs_new:Npn \__siunitx_number_round_auxi:nnnN #1#2#3#4
854 {
855   \quark_if_recursion_tail_stop_do:Nn #4
856   {
857     \__siunitx_number_round_auxii:nnnN {#1} {#3} { } #2
858     \q_recursion_tail \q_recursion_stop
859   }
860   \__siunitx_number_round_auxi:nnnN {#1} {#2} {#4#3}
861 }
862 \cs_new:Npn \__siunitx_number_round_auxii:nnnN #1#2#3#4
863 {
864   \quark_if_recursion_tail_stop_do:Nn #4
865   {
866     \tl_if_blank:nTF {#2}
867     {
868       \__siunitx_number_round_auxiv:nnnN {#1} { } { } #3
869       \q_recursion_tail \q_recursion_stop
870     }
871     {
872       \__siunitx_number_round_auxiii:nnnN {#1} {#3} { } #2
873       \q_recursion_tail \q_recursion_stop
874     }
875   }
876   \__siunitx_number_round_auxii:nnnN {#1} {#2} {#4#3}
877 }

```


We now have the input reversed plus how many digits we need to discard (#1). We have two functions, one which deals with the decimal part, one of which deals with the integer. In the latter, we should never hit the end before we've dropped all the digits: the fixed-zero is a fall-back in case something weird happens. For the integer case, we need to collect up zeros to pad the length back out correctly later.

```

878 \cs_new:Npn \__siunitx_number_round_auxiii:nnnN #1#2#3#4
879 {
880   \quark_if_recursion_tail_stop_do:Nn #4
881   {
882     \__siunitx_number_round_auxiv:nnnN {#1} { } {#3} #2
883     \q_recursion_tail \q_recursion_stop
884   }
885   \int_compare:nNnTF {#1} > 0
886   {
887     \exp_args:Nf \__siunitx_number_round_auxiii:nnnN
888     { \int_eval:n { #1 - 1 } } {#2} { #4#3 }
889   }
890   { \__siunitx_number_round_auxv:nnN {#3} {#2} #4 }
891 }
892 \cs_new:Npn \__siunitx_number_round_auxiv:nnnN #1#2#3#4
893 {
894   \quark_if_recursion_tail_stop_do:Nn #4
895   { { 0 } { } }
896   \int_compare:nNnTF {#1} > 0
897   {
898     \exp_args:Nf \__siunitx_number_round_auxiv:nnnN
899     { \int_eval:n { #1 - 1 } } { #2 0 } { #4#3 }
900   }
901   { \__siunitx_number_round_auxvi:nnnN {#3} {#2} #4 }
902 }

```

The lead off to rounding proper needs to deal with the half-even rule: it can only apply at this stage, when the *discarded* value can be exactly half.

```

903 \cs_new:Npn \__siunitx_number_round_auxv:nnN #1#2#3
904 {
905   \quark_if_recursion_tail_stop_do:Nn #3
906   {
907     \__siunitx_number_round_auxvi:nnN
908     {#1} { } #2 \q_recursion_tail \q_recursion_stop
909   }
910   \bool_lazy_or:nnTF
911   { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
912   {
913     \bool_lazy_all_p:n
914     {
915       { \l__siunitx_number_round_half_even_bool }
916       { \int_if_odd_p:n {#3} }
917       { \__siunitx_number_round_if_half_p:n {#1} }
918     }
919   }
920   { \__siunitx_number_round_final_decimal:nnw }
921   { \__siunitx_number_round_auxvii:nnN }
922   {#2} { } #3
923 }

```

```

924 \cs_new:Npn \__siunitx_number_round_auxvi:nnnN #1#2#3
925 {
926   \quark_if_recursion_tail_stop_do:Nn #3
927   { { 0 } { } }
928   \bool_lazy_or:nnTF
929   { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
930   {
931     \bool_lazy_all_p:n
932     {
933       { \l__siunitx_number_round_half_even_bool }
934       { \int_if_odd_p:n {#3} }
935       { \__siunitx_number_round_if_half_p:n {#1} }
936     }
937   }
938   { \__siunitx_number_round_final_integer:nnw }
939   { \__siunitx_number_round_auxviii:nnN }
940   { } {#2} #3
941 }

```

The main rounding routines. These are only ever called when there is rounding to do, so there is no need to carry a flag forward. Thus the question to ask is simple: is the next value a 9 or not (as that continues the sequence). There is a general need to handle the case where a zero is rounded up: that automatically means a need to trim the other end.

```

942 \cs_new:Npn \__siunitx_number_round_auxvii:nnN #1#2#3
943 {
944   \quark_if_recursion_tail_stop_do:Nn #3
945   {
946     \str_if_eq:nnTF {#1} { 0 }
947     {
948       \__siunitx_number_round_final_output:ff
949       { 1 }
950       { \__siunitx_number_round_truncate:n {#2} }
951     }
952     {
953       \__siunitx_number_round_auxviii:nnN {#2} { } #1
954       \q_recursion_tail \q_recursion_stop
955     }
956   }
957   \int_compare:nNnTF {#3} = 9
958   { \__siunitx_number_round_auxvii:nnN {#1} { 0 #2 } }
959   {
960     \int_compare:nNnTF {#3} = 0
961     {
962       \__siunitx_number_round_final_decimal:nnw
963       {#1} { 1 \__siunitx_number_round_truncate:n {#2} }
964     }
965     {
966       \__siunitx_number_round_final:fn
967       { \int_eval:n { #3 + 1 } }
968       { \__siunitx_number_round_final_decimal:nnw {#1} {#2} }
969     }
970   }
971 }

```

```

972 \cs_new:Npn \__siunitx_number_round_auxviii:nnN #1#2#3
973 {
974   \quark_if_recursion_tail_stop_do:Nn #3
975   {
976     \tl_if_blank:nTF {#1}
977     {
978       \__siunitx_number_round_final_shift:ff
979       {
980         \exp_last_unbraced:Nf 1
981         { \__siunitx_number_round_truncate_direct:n {#2} } 0
982       }
983       { }
984     }
985     {
986       \__siunitx_number_round_final_shift:ff
987       { 1 #2 }
988       { \__siunitx_number_round_truncate:n {#1} }
989     }
990   }
991   \int_compare:nNnTF {#3} = 9
992   { \__siunitx_number_round_auxviii:nnN {#1} { 0 #2 } }
993   {
994     \__siunitx_number_round_final:fn
995     { \int_eval:n { #3 + 1 } }
996     { \__siunitx_number_round_final_integer:nnw {#1} {#2} }
997   }
998 }

```

Tidying up means grabbing the remaining digits and undoing the reversal.

```

999 \cs_new:Npn \__siunitx_number_round_final_decimal:nnw
1000 #1#2#3 \q_recursion_tail \q_recursion_stop
1001 {
1002   \__siunitx_number_round_final_output:ff
1003   { \tl_reverse:n {#1} }
1004   { \tl_reverse:n {#3} #2 }
1005 }
1006 \cs_new:Npn \__siunitx_number_round_final_integer:nnw
1007 #1#2#3 \q_recursion_tail \q_recursion_stop
1008 {
1009   \__siunitx_number_round_final_output:ff
1010   { \tl_reverse:n {#3} #2 }
1011   {#1}
1012 }
1013 \cs_new:Npn \__siunitx_number_round_final_output:nn #1#2 { {#1} {#2} }
1014 \cs_generate_variant:Nn \__siunitx_number_round_final_output:nn { ff }
1015 \cs_new:Npn \__siunitx_number_round_final:nn #1#2
1016 { #2 #1 }
1017 \cs_generate_variant:Nn \__siunitx_number_round_final:nn { f }

```

Here we deal with the case where rounding applies along with an exponent set based on number of places. We can only get here if an additional integer digit has been added, so there is no need to test for that. There are two cases for action: when using **scientific** mode, where we always need to shift by one, and when using **engineering** mode if we now have four digits. The latter is a bit more work: we need to trim digits off as required.

```

1018 \cs_new:Npn \__siunitx_number_round_final_shift:nn #1#2
1019 {
1020   \str_if_eq:VnTF \l__siunitx_number_round_mode_tl { places }
1021   {
1022     \use:c
1023     { __siunitx_number_round_ \l__siunitx_number_exponent_mode_tl :nn }
1024     {#1} {#2}
1025   }
1026   { {#1} {#2} }
1027 }
1028 \cs_generate_variant:Nn \__siunitx_number_round_final_shift:nn { ff }
1029 \cs_new:Npn \__siunitx_number_round_engineering:nn #1#2
1030 {
1031   \int_compare:nNnTF { \tl_count:n {#1} } = 4
1032   {
1033     \__siunitx_number_round_engineering:NNNNn #1 {#2}
1034     { }
1035     \__siunitx_number_round_final_shift:Nw 3
1036   }
1037   { {#1} {#2} }
1038 }
1039 \cs_new:Npn \__siunitx_number_round_engineering:NNNNn #1#2#3#4#5
1040 {
1041   {#1}
1042   \exp_args:NV \__siunitx_number_round_engineering:nnN
1043   { \l__siunitx_number_round_precision_int } { }
1044   #2#3#4#5 \q_recursion_tail \q_recursion_stop
1045 }
1046 \cs_new:Npn \__siunitx_number_round_engineering:nnN #1#2#3
1047 {
1048   \quark_if_recursion_tail_stop_do:Nn #3 { {#2} }
1049   \int_compare:nNnTF {#1} = { 0 }
1050   { \use_i_delimit_by_q_recursion_stop:nw { {#2} } }
1051   { \__siunitx_number_round_engineering:nnN { #1 - 1 } { #2#3 } }
1052 }
1053 \cs_new:Npn \__siunitx_number_round_fixed:nn #1#2 { {#1} {#2} }
1054 \cs_new:Npn \__siunitx_number_round_input:nn #1#2 { {#1} {#2} }
1055 \cs_new:Npn \__siunitx_number_round_scientific:nn #1#2
1056 {
1057   \__siunitx_number_exponent_shift:nnf
1058   { 1 } {#1} { \__siunitx_number_round_truncate_direct:n {#2} }
1059   { }
1060   \__siunitx_number_round_final_shift:Nw 1
1061 }
1062 \cs_new:Npn \__siunitx_number_round_final_shift:Nw #1#2 \__siunitx_number_round_places_end:nn
1063 { \__siunitx_number_exponent_finalise:n { #3#4 + #1 } }

```

When we have rounded up to the next power of ten, we need to go back and remove one more digit. That only happens when rounding to a number of figures or when dealing with an integer part.

```

1064 \cs_new:Npn \__siunitx_number_round_truncate:n #1
1065 {
1066   \str_if_eq:VnTF \l__siunitx_number_round_mode_tl { figures }
1067   { \__siunitx_number_round_truncate_direct:n {#1} }

```

```

1068     {#1}
1069   }
1070 \cs_new:Npn \__siunitx_number_round_truncate_direct:n #1
1071   {
1072     \__siunitx_number_round_truncate:nnN { } { }
1073     #1 \q_recursion_tail \q_recursion_stop
1074   }
1075 \cs_new:Npn \__siunitx_number_round_truncate:nnN #1#2#3
1076   {
1077     \quark_if_recursion_tail_stop_do:Nn #3 { #1 }
1078     \__siunitx_number_round_truncate:nnN {#1#2} {#3}
1079   }

```

(End definition for __siunitx_number_round:nnn and others.)

__siunitx_number_round_if_half_p:n A simple test for a valuing being exactly half: we can only test digit-by-digit as there is
 __siunitx_number_round_if_half:N no limit on the size of the value given.

```

1080 \prg_new_conditional:Npnn \__siunitx_number_round_if_half:n #1 { p }
1081   {
1082     \int_compare:nNnTF { \tl_head:n { #1 0 } } = 5
1083     {
1084       \exp_after:wN \__siunitx_number_round_if_half:N \use_none:n #1 0
1085       \q_recursion_tail \q_recursion_stop
1086     }
1087     { \prg_return_false: }
1088   }
1089 \cs_new:Npn \__siunitx_number_round_if_half:N #1
1090   {
1091     \quark_if_recursion_tail_stop_do:Nn #1
1092     { \prg_return_true: }
1093     \int_compare:nNnTF {#1} = 0
1094     { \__siunitx_number_round_if_half:N }
1095     { \use_i_delimit_by_q_recursion_stop:nw { \prg_return_false: } }
1096   }

```

(End definition for __siunitx_number_round_if_half_p:n and __siunitx_number_round_if_half:N.)

__siunitx_number_round_pad:nnn The case where we are short of digits is easy enough to handle: generate zeros to pad it out.

```

1097 \cs_new:Npn \__siunitx_number_round_pad:nnn #1#2#3
1098   {
1099     {#2}
1100     {
1101       #3
1102       \bool_if:NT \l__siunitx_number_round_pad_bool
1103       { \prg_replicate:nn {#1} { 0 } }
1104     }
1105   }

```

(End definition for __siunitx_number_round_pad:nnn.)

__siunitx_number_round_figures:nnnnnnn Rounding to a fixed number of significant figures starts by checking that there is no
 __siunitx_number_round_figures_count:nnN uncertainty, and that the number of figures requested is positive: if not, the result is
 __siunitx_number_round_figures_count:nnN always fixed at zero.

```

1106 \cs_new:Npn \__siunitx_number_round_figures:nnnnnnn #1#2#3#4#5#6#7
1107 {
1108   \tl_if_blank:nTF {#5}
1109   {
1110     \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
1111     {
1112       \exp_not:n { {#1} {#2} }
1113       \__siunitx_number_round_figures_count:nnN {#3} {#4} #3#4
1114       \q_recursion_tail \q_recursion_stop
1115       \exp_not:n { { } {#6} {#7} }
1116     }
1117     { { } { } { } { 0 } { } { } { } { } { 0 } }
1118   }
1119   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1120 }

```

The first real step is to count up the number of significant figures. The only tricky issue here is dealing with leading zeros.

```

1121 \cs_new:Npn \__siunitx_number_round_figures_count:nnN #1#2#3
1122 {
1123   \quark_if_recursion_tail_stop_do:Nn #3
1124   { { } { } { } { 0 } { } { } { } { } { 0 } }
1125   \int_compare:nNnTF {#3} = 0
1126   { \__siunitx_number_round_figures_count:nnN {#1} {#2} }
1127   { \__siunitx_number_round_figures_count:nnnN { 1 } {#1} {#2} }
1128 }
1129 \cs_new:Npn \__siunitx_number_round_figures_count:nnnN #1#2#3#4
1130 {
1131   \quark_if_recursion_tail_stop_do:Nn #4
1132   {
1133     \int_compare:nNnTF {#1} > \l__siunitx_number_round_precision_int
1134     {
1135       \__siunitx_number_round:fnn
1136       { \int_eval:n { #1 - \l__siunitx_number_round_precision_int } }
1137       {#2} {#3}
1138     }
1139     {
1140       \__siunitx_number_round_pad:nnn
1141       { \l__siunitx_number_round_precision_int - (#1) } {#2} {#3}
1142     }
1143   }
1144   \exp_args:Nf \__siunitx_number_round_figures_count:nnnN
1145   { \int_eval:n { #1 + 1 } } {#2} {#3}
1146 }

```

(End definition for `__siunitx_number_round_figures:nnnnnnn`, `__siunitx_number_round_figures_count:nnN`, and `__siunitx_number_round_figures_count:nnnN`.)

```

\__siunitx_number_round_places:nnnnnnn
\__siunitx_number_round_places_end:nn
\__siunitx_number_round_places_decimal:nn
\__siunitx_number_round_places_integer:nn

```

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. The no-op end function here is to allow tidying up in some cases: see the finalisation of rounding.

```

1147 \cs_new:Npn \__siunitx_number_round_places:nnnnnnn #1#2#3#4#5#6#7
1148 {

```

```

1149 \tl_if_blank:nTF {#5}
1150 {
1151   \exp_not:n { {#1} {#2} }
1152   \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
1153   { \__siunitx_number_round_places_decimal:nn }
1154   { \__siunitx_number_round_places_integer:nn }
1155   {#3} {#4}
1156   \__siunitx_number_round_places_end:nn {#6} {#7}
1157 }
1158 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1159 }
1160 \cs_new:Npn \__siunitx_number_round_places_end:nn #1#2 { { } \exp_not:n { {#1} {#2} } }
1161 \cs_new:Npn \__siunitx_number_round_places_decimal:nn #1#2
1162 {
1163   \int_compare:nNnTF
1164   { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} } > 0
1165   {
1166     \__siunitx_number_round_pad:nnn
1167     { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} }
1168     {#1} {#2}
1169   }
1170   {
1171     \__siunitx_number_round:fnn
1172     {
1173       \int_eval:n
1174       { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }
1175     }
1176     {#1} {#2}
1177   }
1178 }
1179 \cs_new:Npn \__siunitx_number_round_places_integer:nn #1#2
1180 {
1181   \__siunitx_number_round:fnn
1182   {
1183     \int_eval:n
1184     { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }
1185   }
1186   {#1} {#2}
1187 }

```

(End definition for `__siunitx_number_round_places:nnnnnn` and others.)

`__siunitx_number_round_uncertainty:nnnnnn`
`__siunitx_number_round_uncertainty:nnn`
`__siunitx_number_round_uncertainty:nnnn`

Rounding to an uncertainty can only happen where the result will have some uncertainty left: otherwise we simply drop the uncertainty entirely. Only S-type uncertainties can be used for rounding.

```

1188 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnnn #1#2#3#4#5#6#7
1189 {
1190   \bool_lazy_or:nnTF
1191   { \tl_if_blank_p:n {#5} }
1192   { ! \int_compare_p:nNn \l__siunitx_number_round_precision_int > 0 }
1193   { \exp_not:n { {#1} #2 {#3} {#4} { } #6 {#7} } }
1194   {
1195     \str_if_eq_x:nnTF { \tl_head:n {#5} } { S }
1196     {

```

```

1197         \exp_not:n { {#1} {#2} }
1198         \exp_args:Nno \__siunitx_number_round_uncertainty:nnn
1199         {#3} {#4} { \use_ii:nn #5 }
1200         \exp_not:n { {#6} {#7} }
1201     }
1202     { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1203 }
1204 }

```

Round the uncertainty first: this is needed to get the number of places correct (for the case where the uncertainty rounds up to 1...). Once that is done, it's just a question of working out the digits in the main part.

```

1205 \cs_new:Npn \__siunitx_number_round_uncertainty:nnn #1#2#3
1206 {
1207     \exp_last_unbraced:Nf \__siunitx_number_round_uncertainty:nnnnn
1208     {
1209         \__siunitx_number_round:fnn
1210         { \tl_count:n {#3} - \l__siunitx_number_round_precision_int } { } {#3}
1211     }
1212     {#1} {#2} {#3}
1213 }
1214 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnn #1#2#3#4#5
1215 {
1216     \tl_if_blank:nTF {#1}
1217     {
1218         \__siunitx_number_round:fnn
1219         { \tl_count:n {#5} - \tl_count:n {#2} } {#3} {#4}
1220         { { S } {#2} }
1221     }
1222     {
1223         \__siunitx_number_round:fnn
1224         { \tl_count:n {#5} - \tl_count:n {#2} + 1 } {#3} {#4}
1225         { { S } { #1 \__siunitx_number_round_truncate_direct:n {#2} } }
1226     }
1227 }

```

(End definition for __siunitx_number_round_uncertainty:nnnnnnn, __siunitx_number_round_uncertainty:nnn, and __siunitx_number_round_uncertainty:nnnnn.)

__siunitx_number_zero_decimal:NN
__siunitx_number_zero_decimal:nnnnnnn

Simple stripping of the decimal part if zero.

```

1228 \cs_new_protected:Npn \__siunitx_number_zero_decimal:NN #1#2
1229 {
1230     \bool_if:NT \l__siunitx_number_drop_zero_decimal_bool
1231     {
1232         \tl_set:Nx #2
1233         { \exp_after:wN \__siunitx_number_zero_decimal:nnnnnnn #1 }
1234     }
1235 }
1236 \cs_new:Npn \__siunitx_number_zero_decimal:nnnnnnn #1#2#3#4#5#6#7
1237 {
1238     \exp_not:n { {#1} {#2} {#3} }
1239     \str_if_eq_x:nnTF
1240     { \exp_not:n {#4} }
1241     { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1242     { { } }

```



```

1243     { \exp_not:n { {#4} } }
1244     \exp_not:n { {#5} {#6} {#7} }
1245   }

```

(End definition for `__siunitx_number_zero_decimal:NN` and `__siunitx_number_zero_decimal:nnnnnnn`.)

1.5 Formatting parsed numbers

`\l_siunitx_number_output_decimal_tl`

```

1246 \tl_new:N \l_siunitx_number_output_decimal_tl

```

(End definition for `\l_siunitx_number_output_decimal_tl`. This variable is documented on page 13.)

Keys producing tokens in the output.

```

\l_siunitx_number_bracket_negative_bool
  \l_siunitx_number_bracket_close_tl
\l_siunitx_number_implicit_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_uncert_open_tl
\l_siunitx_number_uncert_separate_bool
  \l_siunitx_number_uncert_separator_tl
  \l_siunitx_number_tight_bool
\l_siunitx_number_unity_mantissa_bool
  \l_siunitx_number_zero_exponent_bool
1247 \keys_define:nn { siunitx }
1248 {
1249   bracket-negative .bool_set:N =
1250     \l_siunitx_number_bracket_negative_bool ,
1251   exponent-base .tl_set:N =
1252     \l_siunitx_number_exponent_base_tl ,
1253   exponent-product .tl_set:N =
1254     \l_siunitx_number_exponent_product_tl ,
1255   group-digits .choice: ,
1256   group-digits / all .code:n =
1257     {
1258       \bool_set_true:N \l_siunitx_number_group_decimal_bool
1259       \bool_set_true:N \l_siunitx_number_group_integer_bool
1260     } ,
1261   group-digits / decimal .code:n =
1262     {
1263       \bool_set_true:N \l_siunitx_number_group_decimal_bool
1264       \bool_set_false:N \l_siunitx_number_group_integer_bool
1265     } ,
1266   group-digits / integer .code:n =
1267     {
1268       \bool_set_false:N \l_siunitx_number_group_decimal_bool
1269       \bool_set_true:N \l_siunitx_number_group_integer_bool
1270     } ,
1271   group-digits / none .code:n =
1272     {
1273       \bool_set_false:N \l_siunitx_number_group_decimal_bool
1274       \bool_set_false:N \l_siunitx_number_group_integer_bool
1275     } ,
1276   group-digits .default:n = all ,
1277   group-minimum-digits .int_set:N =
1278     \l_siunitx_number_group_minimum_int ,
1279   group-separator .tl_set:N =
1280     \l_siunitx_number_group_separator_tl ,
1281   negative-color .tl_set:N =
1282     \l_siunitx_number_negative_color_tl ,
1283   number-close-bracket .tl_set:N =
1284     \l_siunitx_number_bracket_close_tl ,
1285   number-open-bracket .tl_set:N =
1286     \l_siunitx_number_bracket_open_tl ,

```

```

1287 output-close-uncertainty .tl_set:N =
1288   \l__siunitx_number_output_uncert_close_tl ,
1289 output-decimal-marker .tl_set:N =
1290   \l__siunitx_number_output_decimal_tl ,
1291 output-open-uncertainty .tl_set:N =
1292   \l__siunitx_number_output_uncert_open_tl ,
1293 print-implicit-plus .bool_set:N =
1294   \l__siunitx_number_implicit_plus_bool ,
1295 print-unity-mantissa .bool_set:N =
1296   \l__siunitx_number_unity_mantissa_bool ,
1297 print-zero-exponent .bool_set:N =
1298   \l__siunitx_number_zero_exponent_bool ,
1299 separate-uncertainty .bool_set:N =
1300   \l__siunitx_number_uncert_separate_bool ,
1301 uncertainty-separator .tl_set:N =
1302   \l__siunitx_number_uncert_separator_tl ,
1303 tight-spacing .bool_set:N =
1304   \l__siunitx_number_tight_bool
1305 }
1306 \bool_new:N \l__siunitx_number_group_decimal_bool
1307 \bool_new:N \l__siunitx_number_group_integer_bool

```

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

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.

```

\siunitx_number_format:N
\siunitx_number_format:NN
\__siunitx_number_format:Nn
  \_siunitx_number_format:nnnnnnn
  \_siunitx_number_format_comparator:nn
  \_siunitx_number_format_sign:n
  \_siunitx_number_format_sign:N
  \_siunitx_number_format_sign_color:w
  \_siunitx_number_format_sign_brackets:w
  \_siunitx_number_format_integer:nnn
  \_siunitx_number_format_decimal:nn
  \_siunitx_number_format_decimal:fn
  \_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:nnn
  \_siunitx_number_format_uncertainty_unaligned:n
  \_siunitx_number_format_uncertainty_S:nnw
  \_siunitx_number_format_uncertainty_S_aux:nnn
  \_siunitx_number_format_uncertainty_S:nnnw
  \_siunitx_number_format_uncertainty_S:fnw
  \_siunitx_number_format_uncertainty_S:nnw
  \_siunitx_number_format_exponent:nnnn
  \_siunitx_number_format_end:

```

```

1308 \cs_new:Npn \siunitx_number_format:N #1
1309 { \__siunitx_number_format:Nn #1 { } }
1310 \cs_new:Npn \siunitx_number_format:NN #1#2
1311 { \__siunitx_number_format:Nn #1 {#2} }
1312 \cs_new:Npn \__siunitx_number_format:Nn #1#2
1313 {
1314   \tl_if_empty:NF #1
1315   { \exp_after:wN \__siunitx_number_format:nnnnnnn #1 {#2} }
1316 }
1317 \cs_new:Npn \__siunitx_number_format:nnnnnnn #1#2#3#4#5#6#7#8
1318 {
1319   \_siunitx_number_format_comparator:nn {#1} {#8}
1320   \_siunitx_number_format_sign:n {#2}
1321   \_siunitx_number_format_integer:nnn {#3} {#4} {#7}
1322   \_siunitx_number_format_decimal:nn {#4} {#8}
1323   \_siunitx_number_format_uncertainty:nnn {#5} {#4} {#8}
1324   \_siunitx_number_format_exponent:nnnn {#6} {#7} { #3 . #4 } {#8}
1325   \_siunitx_number_format_end:
1326 }

```

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

```

1327 \cs_new:Npn \__siunitx_number_format_comparator:nn #1#2
1328 {
1329   \tl_if_blank:NF {#1}
1330   { \exp_not:n { \mathord {#1} } }
1331   \exp_not:n {#2}
1332 }

```

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.

```

1333 \cs_new:Npn \__siunitx_number_format_sign:n #1
1334 {
1335   \tl_if_blank:nTF {#1}
1336   {
1337     \bool_if:NT \l__siunitx_number_implicit_plus_bool
1338     { \__siunitx_number_format_sign:N + }
1339   }
1340   {
1341     \str_if_eq:nnTF {#1} { - }
1342     {
1343       \tl_if_empty:NF \l__siunitx_number_negative_color_tl
1344       { \__siunitx_number_format_sign_color:w }
1345       \bool_if:NTF \l__siunitx_number_bracket_negative_bool
1346       { \__siunitx_number_format_sign_brackets:w }
1347       { \__siunitx_number_format_sign:N #1 }
1348     }
1349     { \__siunitx_number_format_sign:N #1 }
1350   }
1351 }
1352 \cs_new:Npn \__siunitx_number_format_sign:N #1
1353 {
1354   \bool_if:NTF \l__siunitx_number_tight_bool
1355   { \exp_not:n { \mathord {#1} } }
1356   { \exp_not:n {#1} }
1357 }
1358 \cs_new:Npn
1359   \__siunitx_number_format_sign_color:w #1 \__siunitx_number_format_end:
1360 {
1361   \exp_not:N \textcolor { \exp_not:V \l__siunitx_number_negative_color_tl }
1362   {
1363     #1
1364     \__siunitx_number_format_end:
1365   }
1366 }
1367 \cs_new:Npn
1368   \__siunitx_number_format_sign_brackets:w #1 \__siunitx_number_format_end:
1369 {
1370   \exp_not:V \l__siunitx_number_bracket_open_tl
1371   #1
1372   \exp_not:V \l__siunitx_number_bracket_close_tl
1373   \__siunitx_number_format_end:
1374 }
1375 \cs_new:Npn \__siunitx_number_format_integer:nnn #1#2#3
1376 {
1377   \bool_lazy_all:nF
1378   {

```

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.

```

1379     { \str_if_eq_p:nn {#1} { 1 } }
1380     { \tl_if_blank_p:n {#2} }
1381     { ! \str_if_eq_p:nn {#3} { 0 } }
1382     { ! \l__siunitx_number_unity_mantissa_bool }
1383   }
1384   { \__siunitx_number_format_digits:nn { integer } {#1} }
1385 }
1386 \cs_new:Npn \__siunitx_number_format_decimal:nn #1#2
1387 {
1388   \exp_not:n {#2}
1389   \tl_if_blank:nF {#1}
1390   {
1391     \str_if_eq:VnTF \l_siunitx_number_output_decimal_tl { , }
1392     { \exp_not:N \mathord }
1393     { \use:n }
1394     { \exp_not:V \l_siunitx_number_output_decimal_tl }
1395   }
1396   \exp_not:n {#2}
1397   \__siunitx_number_format_digits:nn { decimal } {#1}
1398 }
1399 \cs_generate_variant:Nn \__siunitx_number_format_decimal:nn { f }
1400 \cs_new:Npn \__siunitx_number_format_digits:nn #1#2
1401 {
1402   \bool_if:cTF { l__siunitx_number_group_ #1 _ bool }
1403   {
1404     \int_compare:nNnTF
1405     { \tl_count:n {#2} } < \l__siunitx_number_group_minimum_int
1406     { \exp_not:n {#2} }
1407     { \use:c { __siunitx_number_format_ #1 _aux:n } {#2} }
1408   }
1409   { \exp_not:n {#2} }
1410 }

```

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.

```

1411 \cs_new:Npn \__siunitx_number_format_integer_aux:n #1
1412 {
1413   \use:c
1414   {
1415     __siunitx_number_format_integer_aux_
1416     \int_eval:n { \int_mod:nn { \tl_count:n {#1} } { 3 } }
1417     :n
1418   } {#1}
1419 }
1420 \cs_new:cpn { __siunitx_number_format_integer_aux_0:n } #1
1421 { \__siunitx_number_format_integer_first:nnNN #1 \q_nil }
1422 \cs_new:cpn { __siunitx_number_format_integer_aux_1:n } #1
1423 { \__siunitx_number_format_integer_first:nnNN { } { } #1 \q_nil }
1424 \cs_new:cpn { __siunitx_number_format_integer_aux_2:n } #1
1425 { \__siunitx_number_format_integer_first:nnNN { } #1 \q_nil }
1426 \cs_new:Npn \__siunitx_number_format_integer_first:nnNN #1#2#3#4
1427 {
1428   \exp_not:n {#1#2#3}

```

```

1429 \quark_if_nil:NF #4
1430 { \__siunitx_number_format_integer_loop:NNNN #4 }
1431 }
1432 \cs_new:Npn \__siunitx_number_format_integer_loop:NNNN #1#2#3#4
1433 {
1434   \exp_not:V \l__siunitx_number_group_separator_tl
1435   \exp_not:n {#1#2#3}
1436   \quark_if_nil:NF #4
1437   { \__siunitx_number_format_integer_loop:NNNN #4 }
1438 }

```

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.

```

1439 \cs_new:Npn \__siunitx_number_format_decimal_aux:n #1
1440 {
1441   \__siunitx_number_format_decimal_loop:NNNN \c_empty_tl
1442   #1 \q_nil \q_nil \q_nil
1443 }
1444 \cs_new:Npn \__siunitx_number_format_decimal_loop:NNNN #1#2#3#4
1445 {
1446   \quark_if_nil:NF #2
1447   {
1448     \exp_not:V #1
1449     \exp_not:n {#2}
1450     \quark_if_nil:NTF #3
1451     { \use_none:n }
1452     {
1453       \exp_not:n {#3}
1454       \quark_if_nil:NTF #4
1455       { \use_none:nn }
1456       {
1457         \exp_not:n {#4}
1458         \__siunitx_number_format_decimal_loop:NNNN
1459         \l__siunitx_number_group_separator_tl
1460       }
1461     }
1462   }
1463 }

```

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.

```

1464 \cs_new:Npn \__siunitx_number_format_uncertainty:nnn #1#2#3
1465 {
1466   \tl_if_blank:NTF {#1}
1467   { \__siunitx_number_format_uncertainty_unaligned:n {#3} }
1468   {
1469     \use:c { \__siunitx_number_format_uncertainty_ \tl_head:n {#1} :nnnw }
1470     {#2} {#3} #1
1471   }

```

```

1472 }
1473 \cs_new:Npn \__siunitx_number_format_uncertainty_unaligned:n #1
1474 { \exp_not:n { #1 #1 #1 #1 } }
1475 \cs_new:Npn \__siunitx_number_format_uncertainty_S:nnnw #1#2#3#4
1476 {
1477   \bool_if:NTF \l__siunitx_number_uncert_separate_bool
1478   {
1479     \exp_not:n {#2}
1480     \__siunitx_number_format_sign:N \pm
1481     \exp_not:n {#2}
1482     \exp_args:Nf \__siunitx_number_format_uncertainty_S_aux:nnn
1483     { \int_eval:n { \tl_count:n {#4} - \tl_count:n {#1} } }
1484     {#4} {#2}
1485   }
1486   {
1487     \exp_not:V \l__siunitx_number_uncert_separator_tl
1488     \exp_not:V \l__siunitx_number_output_uncert_open_tl
1489     \exp_not:n {#4}
1490     \exp_not:V \l__siunitx_number_output_uncert_close_tl
1491     \__siunitx_number_format_uncertainty_unaligned:n {#2}
1492   }
1493 }
1494 \cs_new:Npn \__siunitx_number_format_uncertainty_S_aux:nnn #1#2#3
1495 {
1496   \int_compare:nNnTF {#1} > 0
1497   {
1498     \__siunitx_number_format_uncertainty_S_aux:fnnw
1499     { \int_eval:n { #1 - 1 } }
1500     {#3}
1501     { }
1502     #2 \q_nil
1503   }
1504   {
1505     0
1506     \__siunitx_number_format_decimal:fn
1507     {
1508       \prg_replicate:nn { \int_abs:n {#1} } { 0 }
1509       #2
1510     }
1511     {#3}
1512   }
1513 }
1514 \cs_new:Npn \__siunitx_number_format_uncertainty_S_aux:nnnw #1#2#3#4
1515 {
1516   \quark_if_nil:NF #4
1517   {
1518     \int_compare:nNnTF {#1} = 0
1519     { \__siunitx_number_format_uncertainty_S_aux:nnw {#3#4} {#2} }
1520     {
1521       \__siunitx_number_format_uncertainty_S_aux:fnnw
1522       { \int_eval:n { #1 - 1 } }
1523       {#2}
1524       {#3#4}
1525     }
1526   }

```

```

1526     }
1527   }
1528   \cs_generate_variant:Nn \__siunitx_number_format_uncertainty_S_aux:nnnw { f }
1529   \cs_new:Npn \__siunitx_number_format_uncertainty_S_aux:nnw #1#2#3 \q_nil
1530   {
1531     \__siunitx_number_format_digits:nn { integer } {#1}
1532     \__siunitx_number_format_decimal:nn {#3} {#2}
1533   }

```

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.

```

1534   \cs_new:Npn \__siunitx_number_format_exponent:nnnn #1#2#3#4
1535   {
1536     \exp_not:n {#4}
1537     \bool_lazy_or:nnTF
1538     { \l__siunitx_number_zero_exponent_bool }
1539     { ! \str_if_eq_p:nn {#2} { 0 } }
1540     {
1541       \bool_lazy_and:nnTF
1542       { \str_if_eq_p:nn {#3} { 1. } }
1543       { ! \l__siunitx_number_unity_mantissa_bool }
1544       { \exp_not:n {#4} }
1545       {
1546         \bool_if:NTF \l__siunitx_number_tight_bool
1547         {
1548           \exp_not:N \mathord
1549           { \exp_not:V \l__siunitx_number_exponent_product_tl }
1550         }
1551         { \exp_not:V \l__siunitx_number_exponent_product_tl }
1552         \exp_not:n {#4}
1553       }
1554       \exp_not:V \l__siunitx_number_exponent_base_tl
1555       ~
1556       {
1557         \tl_if_blank:nTF {#1}
1558         {
1559           \bool_if:NT \l__siunitx_number_implicit_plus_bool
1560           { \__siunitx_number_format_sign:N + }
1561         }
1562         { \__siunitx_number_format_sign:N #1 }
1563         \__siunitx_number_format_digits:nn { integer } {#2}
1564       }
1565     }
1566     { \exp_not:n {#4} }
1567   }

```

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

```

1568   \cs_new:Npn \__siunitx_number_format_end: { }

```

(End definition for `\siunitx_number_format:N` and others. These functions are documented on page 13.)

1.6 Miscellaneous tools

`\l__siunitx_number_valid_tl` The list of valid tokens.

```
1569 \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.

```
1570 \prg_new_protected_conditional:Npnn \siunitx_if_number:n #1
1571 { T , F , TF }
1572 {
1573   \group_begin:
1574     \bool_set_true:N \l__siunitx_number_validate_bool
1575     \bool_set_true:N \l__siunitx_number_parse_bool
1576     \siunitx_number_parse:nN {#1} \l__siunitx_number_parsed_tl
1577     \tl_if_empty:NTF \l__siunitx_number_parsed_tl
1578       {
1579         \group_end:
1580         \prg_return_false:
1581       }
1582       {
1583         \group_end:
1584         \prg_return_true:
1585       }
1586 }
```

(End definition for `\siunitx_if_number:nTF`. This function is documented on page 13.)

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

`\siunitx_if_number_token:NTF`

`_siunitx_number_if_number_token_auxi:NN`

`_siunitx_number_if_number_token_auxii:NN`

`_siunitx_number_if_number_token_auxiii:NN`

```
1587 \prg_new_conditional:Npnn \siunitx_if_number_token:N #1
1588 { p , T , F , TF }
1589 {
1590   \_siunitx_number_number_token_auxi:NN #1
1591   \l__siunitx_number_input_decimal_tl
1592   \l__siunitx_number_input_uncert_close_tl
1593   \l__siunitx_number_input_comparator_tl
1594   \l__siunitx_number_input_digit_tl
1595   \l__siunitx_number_input_exponent_tl
1596   \l__siunitx_number_input_ignore_tl
1597   \l__siunitx_number_input_uncert_open_tl
1598   \l__siunitx_number_input_sign_tl
1599   \l__siunitx_number_input_uncert_sign_tl
1600   \q_recursion_tail
1601   \q_recursion_stop
1602 }
1603 \cs_new:Npn \_siunitx_number_number_token_auxi:NN #1#2
1604 {
1605   \quark_if_recursion_tail_stop_do:Nn #2 { \prg_return_false: }
1606   \_siunitx_number_number_token_auxii:NN #1 #2
```



```

1607   \_siunitx_number_number_token_auxi:NN #1
1608 }
1609 \cs_new:Npn \_siunitx_number_number_token_auxii:NN #1#2
1610 {
1611   \exp_after:wN \_siunitx_number_number_token_auxiii:NN \exp_after:wN #1
1612   #2 \q_recursion_tail \q_recursion_stop
1613 }
1614 \cs_new:Npn \_siunitx_number_number_token_auxiii:NN #1#2
1615 {
1616   \quark_if_recursion_tail_stop:N #2
1617   \str_if_eq:nnT {#1} {#2}
1618   {
1619     \use_i_delimit_by_q_recursion_stop:nw
1620     {
1621       \use_i_delimit_by_q_recursion_stop:nw
1622       { \prg_return_true: }
1623     }
1624   }
1625   \_siunitx_number_number_token_auxiii:NN #1
1626 }

```

(End definition for `\siunitx_if_number_token:NTF` and others. This function is documented on page [13](#).)

1.7 Messages

```

1627 \msg_new:nnnn { siunitx } { invalid-number }
1628 { Invalid-number~'#1'. }
1629 {
1630   The~input~'#1'~could-not-be-parsed-as-a-number-following-the~
1631   format-defined-in-module-documentation.
1632 }

```

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.

```

1633 \keys_set:nn { siunitx }
1634 {
1635   bracket-negative           = false ,
1636   drop-exponent             = false ,
1637   drop-uncertainty          = false ,
1638   drop-zero-decimal         = false ,
1639   evaluate-expression       = false ,
1640   exponent-base              = 10 ,
1641   exponent-mode              = input ,
1642   exponent-product          = \times ,
1643   expression                 = #1 ,
1644   fixed-exponent            = 0 ,
1645   group-digits               = all ,
1646   group-minimum-digits      = 4 ,
1647   group-separator           = \, , % (
1648   input-close-uncertainty    = ) ,
1649   input-comparators         = { <=>\approx\ge\gg\le\leq\ll\sim } ,
1650   input-decimal-markers     = { ., } ,

```

```

1651 input-digits           = 0123456789
1652 input-exponent-markers = dDeE
1653 input-ignore          = \,
1654 input-open-uncertainty = (
1655 input-signs            = +-\mp\pm
1656 input-uncertainty-signs = \pm
1657 minimum-decimal-digits = 0
1658 minimum-integer-digits = 0
1659 negative-color         =
1660 number-close-bracket   = )
1661 number-open-bracket     = (
1662 output-close-uncertainty = )
1663 output-decimal-marker   = .
1664 output-open-uncertainty = (
1665 parse-numbers           = true
1666 print-implicit-plus     = false
1667 print-unity-mantissa    = false
1668 print-zero-exponent     = false
1669 round-half              = up
1670 round-minimum           = 0
1671 round-mode              = none
1672 round-pad               = true
1673 round-precision         = 2
1674 separate-uncertainty   = false
1675 track-explicit-plus     = false
1676 uncertainty-separator   =
1677 tight-spacing           = false
1678 }
1679 </package>

```

Part IV

siunitx-print – Printing material with font control

1 Printing quantities

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

The printing routines assume that normal 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 expanded weight, mapping to math version **bold**.
- `lt` Light weight, mapping to math version **light**.
- `l` Light weight, mapping to math version **light**.

1.1 Key-value options

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

<hr/> <hr/> <code>color</code>	<code>color = $\langle color \rangle$</code>
	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/> <code>mode</code>	<code>mode = match math text</code>
	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/> <code>number-color</code>	<code>number-color = $\langle color \rangle$</code>
	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/> <code>number-mode</code>	<code>number-mode = match math text</code>
	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/> <code>propagate-math-font</code>	<code>propagate-math-font = true false</code>
	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/> <code>reset-math-version</code>	<code>reset-math-version = true false</code>
	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/> <code>reset-text-family</code>	<code>reset-text-family = true false</code>
	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/> <code>reset-text-series</code>	<code>reset-text-series = true false</code>
	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/> <code>reset-text-shape</code>	<code>reset-text-shape = true false</code>
	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 = match math text
	Selects which mode (math or text) units are printed in: a choice from the options match , math or text . The option match matches the mode prevailing at the point <code>\siunitx_print:nn</code> is called. The math and text options choose the relevant \TeX mode for printing. The standard setting is math .

2 siunitx-print implementation

Start the DocStrip guards.

```
1 \(*package\)
```

Identify the internal prefix (\LaTeX 3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2 \@@=siunitx_p\print)
```

2.1 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 }
```

`\tl_replace_all:NVn` Required variants.

```
11 \cs_generate_variant:Nn \tl_replace_all:Nnn { NV }
```

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

`\l__siunitx_print_tmp_box` Scratch space.

```
\l__siunitx_print_tmp_tl 12 \box_new:N \l__siunitx_print_tmp_box
13 \tl_new:N \l__siunitx_print_tmp_tl
```

(End definition for `\l__siunitx_print_tmp_box` and `\l__siunitx_print_tmp_tl`.)

`\document`

In order to test math fonts, we need information about the `\fam` used by the various options. This is run as a hook onto `\document`, rather than using `\AtBeginDocument` as it has to come after anything that `fontspec` does (nasty errors arise otherwise). As this is a true one-off, we avoid wasting a box.

```
14 \tl_put_right:Nn \document
15 {
16   \__siunitx_print_store_fam:n { rm }
17   \__siunitx_print_store_fam:n { sf }
18   \__siunitx_print_store_fam:n { tt }
19   \ignorespaces
20 }
21 \cs_new_protected:Npn \__siunitx_print_store_fam:n #1
22 {
23   \group_begin:
24     \hbox_set:Nn \l__siunitx_print_tmp_box
25     {
26       \ensuremath
27       {
28         \use:c { math #1 }
29         { \int_const:cn { c__siunitx_print_math #1 _int } { \fam } }
30       }
31     }
32   \group_end:
33 }
```

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

2.2 Printing routines

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

```
\l_siunitx_print_number_color_tl 34 \tl_new:N \l__siunitx_print_number_mode_tl
\l_siunitx_print_number_mode_tl 35 \tl_new:N \l__siunitx_print_unit_mode_tl
\l_siunitx_print_unit_color_tl 36 \keys_define:nn { siunitx }
\l_siunitx_print_unit_mode_tl 37 {
\l_siunitx_print_math_font_bool 38   color .meta:n =
\l_siunitx_print_math_family_bool 39   { number-color = #1 , unit-color = #1 } ,
\l_siunitx_print_math_weight_bool 40   mode .meta:n =
\l_siunitx_print_text_family_tl 41   { number-mode = #1 , unit-mode = #1 } ,
\l_siunitx_print_text_series_tl 42   number-color .tl_set:N =
\l_siunitx_print_text_shape_tl 43   \l__siunitx_print_number_color_tl ,
44   number-mode .choices:nn =
45   { match , math , text }
46   {
47     \tl_set_eq:NN
48     \l__siunitx_print_number_mode_tl \l_keys_choice_tl
49   } ,
50   propagate-math-font .bool_set:N =
```

```

51     \l__siunitx_print_math_font_bool ,
52     reset-math-version .bool_set:N =
53     \l__siunitx_print_math_version_bool ,
54     reset-text-family .bool_set:N =
55     \l__siunitx_print_text_family_bool ,
56     reset-text-series .bool_set:N =
57     \l__siunitx_print_text_series_bool ,
58     reset-text-shape .bool_set:N =
59     \l__siunitx_print_text_shape_bool ,
60     text-family-to-math .bool_set:N =
61     \l__siunitx_print_math_family_bool ,
62     text-weight-to-math .bool_set:N =
63     \l__siunitx_print_math_weight_bool ,
64     unit-color .tl_set:N =
65     \l__siunitx_print_unit_color_tl ,
66     unit-mode .choices:nn =
67     { match , math , text }
68     {
69         \tl_set_eq:NN
70         \l__siunitx_print_unit_mode_tl \l_keys_choice_tl
71     }
72 }

```

(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.

```

73 \cs_new_protected:Npn \siunitx_print:nn #1#2
74 {
75     \tl_if_empty:cTF { l__siunitx_print_ #1 _color_tl }
76     { \use:n }
77     { \exp_args:Nv \textcolor { l__siunitx_print_ #1 _color_tl } }
78     {
79         \use:c
80         {
81             __siunitx_print_
82             \tl_use:c { l__siunitx_print_ #1 _mode_tl } :n
83         }
84         {#2}
85     }
86 }
87 \cs_generate_variant:Nn \siunitx_print:nn { nV }

```

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

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

```

88 \cs_new_protected:Npn \__siunitx_print_match:n #1
89 {
90     \mode_if_math:TF
91     { \__siunitx_print_math:n {#1} }
92     { \__siunitx_print_text:n {#1} }
93 }

```

(End definition for `__siunitx_print_match:n`.)

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

```

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

```

(End definition for `\l_siunitx_print_series_prop`. This variable is documented on page 56.)

`__siunitx_print_replace_font:N` A simple auxiliary for “zapping” the unit font.

```

99 \cs_new_protected:Npn \__siunitx_print_replace_font:N #1
100 {
101   \tl_if_empty:NF \l_siunitx_unit_font_tl
102   {
103     \tl_replace_all:Nvn #1
104       \l_siunitx_unit_font_tl
105     { \use:n }
106   }
107 }

```

(End definition for `__siunitx_print_replace_font:N`.)

`__siunitx_print_math:n` The first step in setting in math mode is to check on the math version. The starting point is the question of whether text series needs to propagate to math mode: if so, check on the mapping, otherwise check on the current math version.

```

\__siunitx_print_math_auxi:n 108 \cs_new_protected:Npn \__siunitx_print_math:n #1
\__siunitx_print_math_auxii:n 109 {
\__siunitx_print_math_auxiii:n 110   \bool_if:NTF \l__siunitx_print_math_weight_bool
\__siunitx_print_math_auxiv:n 111   {
\__siunitx_print_math_auxv:n 112     \prop_get:NvNTF \l_siunitx_print_series_prop
\__siunitx_print_math_aux:Nn 113       \f@series \l__siunitx_print_tmp_tl
\__siunitx_print_math_aux:cn 114       { \__siunitx_print_math_version:Nn \l__siunitx_print_tmp_tl {#1} }
\__siunitx_print_math_sub:n 115       { \__siunitx_print_math_auxi:n {#1} }
\__siunitx_print_math_super:n 116     }
\__siunitx_print_math_script:n 117   { \__siunitx_print_math_auxi:n {#1} }
\__siunitx_print_math_text:n 118 }
119 \cs_new_protected:Npn \__siunitx_print_math_auxi:n #1
120 {
121   \bool_if:NTF \l__siunitx_print_math_version_bool
122   { \__siunitx_print_math_version:nn { normal } {#1} }
123   { \__siunitx_print_math_auxii:n {#1} }
124 }

```

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 to check 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.

```

125 \cs_new_protected:Npn \__siunitx_print_math_version:nn #1#2
126 {
127   \str_if_eq:NvTF \math@version { #1 }
128   { \__siunitx_print_math_auxii:n {#2} }
129   {
130     \mode_if_math:TF
131     { \text }

```



```

132         { \use:n }
133         {
134             \mathversion {#1}
135             \__siunitx_print_math_auxii:n {#2}
136         }
137     }
138 }
139 \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. The test here is on a string basis as `\f@family` and the `\...default` commands have different `\long` status.

```

140 \cs_new_protected:Npn \__siunitx_print_math_auxii:n #1
141 { \ensuremath { \__siunitx_print_math_auxiii:n {#1} } }
142 \cs_new_protected:Npn \__siunitx_print_math_auxiii:n #1
143 {
144     \bool_if:NTF \l__siunitx_print_math_family_bool
145     {
146         \str_case:x:nnF { \f@family }
147         {
148             { \rmdefault } { \__siunitx_print_math_auxv:n }
149             { \sfdefault } { \__siunitx_print_math_aux:Nn \mathsf }
150             { \ttdefault } { \__siunitx_print_math_aux:Nn \mathtt }
151         }
152         { \__siunitx_print_math_auxiv:n }
153     }
154     { \__siunitx_print_math_auxiv:n }
155     {#1}
156 }

```

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.

```

157 \cs_new_protected:Npn \__siunitx_print_math_auxiv:n #1
158 {
159     \bool_if:NTF \l__siunitx_print_math_font_bool
160     {
161         \int_case:nnF \fam
162         {
163             \c__siunitx_print_mathsf_int { \__siunitx_print_math_aux:Nn \mathsf }
164             \c__siunitx_print_mathtt_int { \__siunitx_print_math_aux:Nn \mathtt }
165         }
166         { \use:n }
167     }
168     { \__siunitx_print_math_auxv:n }
169     {#1}
170 }
171 \cs_new_protected:Npn \__siunitx_print_math_auxv:n #1
172 {
173     \bool_lazy_or:nnTF

```

```

174 { \int_compare_p:nNn \fam = { -1 } }
175 { \int_compare_p:nNn \fam = \c__siunitx_print_mathrm_int }
176 { \use:n }
177 { \mathrm }
178 {#1}
179 }

```

Search-and-replace fun: deal with any font commands in the argument and also inside sub/superscripts.

```

180 \cs_new_protected:Npx \__siunitx_print_math_aux:Nn #1#2
181 {
182   \group_begin:
183     \tl_set:Nn \exp_not:N \l__siunitx_print_tmp_tl {#2}
184     \__siunitx_print_replace_font:N \exp_not:N \l__siunitx_print_tmp_tl
185     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
186       { \char_generate:nn { '\_ } { 8 } }
187       { \exp_not:N \__siunitx_print_math_sub:n }
188     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
189       { ^ }
190       { \exp_not:N \__siunitx_print_math_super:n }
191     #1 { \exp_not:N \tl_use:N \exp_not:N \l__siunitx_print_tmp_tl }
192   \group_end:
193 }
194 \cs_generate_variant:Nn \__siunitx_print_math_aux:Nn { c }
195 \cs_new_protected:Npx \__siunitx_print_math_sub:n #1
196 {
197   \char_generate:nn { '\_ } { 8 }
198   { \exp_not:N \__siunitx_print_math_script:n {#1} }
199 }
200 \cs_new_protected:Npn \__siunitx_print_math_super:n #1
201 { ^ { \__siunitx_print_math_script:n {#1} } }
202 \cs_new_protected:Npn \__siunitx_print_math_script:n #1
203 {
204   \group_begin:
205     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
206     \__siunitx_print_replace_font:N \l__siunitx_print_tmp_tl
207     \tl_use:N \l__siunitx_print_tmp_tl
208   \group_end:
209 }

```

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.

210

(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:NnN __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. 211 \cs_new_protected:Npn __siunitx_print_text:n #1 212 { 213 \text 214 { 215 \bool_if:NT \l__siunitx_print_text_family_bool 216 { \rmfamily } 217 \bool_if:NT \l__siunitx_print_text_series_bool </pre>
--	---

```

218         { \mdseries }
219         \bool_if:NT \l__siunitx_print_text_shape_bool
220         { \upshape }
221         \__siunitx_print_text_replace:n {#1}
222     }
223 }

```

To get math mode material to print in text mode, various search-and-replace steps are needed.

```

224 \cs_new_protected:Npn \__siunitx_print_text_replace:n #1
225 {
226     \group_begin:
227     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
228     \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
229     \tl_use:N \l__siunitx_print_tmp_tl
230     \group_end:
231 }
232 \cs_new_protected:Npx \__siunitx_print_text_replace:N #1
233 {
234     \__siunitx_print_replace_font:N #1
235     \exp_not:N \__siunitx_print_text_replace:NNn #1
236     \exp_not:N \pm
237     { \exp_not:N \textpm }
238     \exp_not:N \mp
239     { \exp_not:n { \ensuremath { \mp } } } }
240 -
241     { \exp_not:N \textminus }
242     \char_generate:nn { '\_ } { 8 }
243     { \exp_not:N \__siunitx_print_text_sub:n }
244     ~
245     { \exp_not:N \__siunitx_print_text_super:n }
246     \exp_not:N \q_recursion_tail
247     { ? }
248     \exp_not:N \q_recursion_stop
249 }
250 \cs_new_protected:Npn \__siunitx_print_text_replace:NNn #1#2#3
251 {
252     \quark_if_recursion_tail_stop:N #2
253     \tl_replace_all:Nnn #1 {#2} {#3}
254     \__siunitx_print_text_replace:NNn #1
255 }

```

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.

```

256 \cs_new_protected:Npn \__siunitx_print_text_sub:n #1
257 {
258     \__siunitx_print_text_scripts:NnN
259     \textsubscript {#1} \__siunitx_print_text_super:n
260 }
261 \cs_new_protected:Npn \__siunitx_print_text_super:n #1
262 {
263     \__siunitx_print_text_scripts:NnN
264     \textsuperscript {#1} \__siunitx_print_text_sub:n
265 }
266 \cs_new_protected:Npn \__siunitx_print_text_scripts:NnN #1#2#3

```

```

267 {
268   \cs_set_protected:Npn \__siunitx_print_text_scripts:
269   {
270     \if_meaning:w \l_peek_token #3
271     \exp_after:wN \__siunitx_print_text_scripts_two:NnNn
272     \else:
273     \exp_after:wN \__siunitx_print_text_scripts_one:Nn
274     \fi:
275     #1 {#2}
276   }
277   \peek_after:Nw \__siunitx_print_text_scripts:
278 }
279 \cs_new_protected:Npn \__siunitx_print_text_scripts: { }

```

In the simple case of one script item, we have to do a search-and-replace to deal with anything inside the argument.

```

280 \cs_new_protected:Npn \__siunitx_print_text_scripts_one:Nn #1#2
281 {
282   \group_begin:
283   \tl_set:Nn \l__siunitx_print_tmp_tl {#2}
284   \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
285   \exp_args:NNV \group_end:
286   #1 \l__siunitx_print_tmp_tl
287 }

```

For the two scripts case, we cannot use `\textsubscript/\textsupsript` 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.

```

288 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:NnNn #1#2#3#4
289 {
290   \cs_if_eq:NNTF #1 \textsubscript
291   { \__siunitx_print_text_scripts_two:nn {#4} {#2} }
292   { \__siunitx_print_text_scripts_two:nn {#2} {#4} }
293 }
294 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:nn #1#2
295 {
296   \group_begin:
297   \exp_not:N \m@th
298   \exp_not:N \ensuremath
299   {
300     ^ { \exp_not:N \__siunitx_print_text_scripts_two:n {#1} }
301     \char_generate:nn { '\_ } { 8 }
302     { \exp_not:N \__siunitx_print_text_scripts_two:n {#2} }
303   }
304   \group_end:
305 }
306 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:n #1
307 {
308   \mbox
309   {
310     \fontsize \sf@size \z@ \selectfont
311     \__siunitx_print_text_scripts_one:Nn \use:n {#1}
312   }
313 }

```

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

2.3 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.

```
314 \keys_set:nn { siunitx }  
315 {  
316   color                = ,  
317   mode                 = math ,  
318   number-color         = ,  
319   number-mode          = math ,  
320   propagate-math-font = false ,  
321   reset-math-version   = true  ,  
322   reset-text-shape     = true  ,  
323   reset-text-series    = true  ,  
324   reset-text-family    = true  ,  
325   text-family-to-math  = false ,  
326   text-weight-to-math  = false ,  
327   unit-color           = ,  
328   unit-mode            = math  
329 }  
330 \end{package}
```

Part V

siunitx-font — Font-related settings

1 siunitx-symbol 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_symbol>
```

```
\l__siunitx_symbol_tmpa_tl Scratch space.
```

```
\l__siunitx_symbol_tmpb_tl
```

```
3 \tl_new:N \l__siunitx_symbol_tmpa_tl
```

```
4 \tl_new:N \l__siunitx_symbol_tmpb_tl
```

(End definition for `\l__siunitx_symbol_tmpa_tl` and `\l__siunitx_symbol_tmpb_tl`.)

```
\__siunitx_symbol_textmu: Some of the TS1 encoding is needed to provide symbols in text mode for 8-bit engines.  
If the user has not loaded the encoding themselves, it is done here before creating the  
required commands.
```

```
5 \bool_lazy_or:nnF  
6 { \sys_if_engine luatex_p: }  
7 { \sys_if_engine xetex_p: }  
8 {  
9   \AtBeginDocument  
10    {  
11      \cs_if_free:cT { T@TS1 }  
12      {  
13        \DeclareFontEncoding { TS1 } { } { }  
14        \DeclareFontSubstitution { TS1 } { cmr } { m } { n }  
15      }  
16      \cs_if_free:NTF \textmu  
17      {  
18        \DeclareTextSymbolDefault \__siunitx_symbol_textmu: { TS1 }  
19        \DeclareTextSymbol \__siunitx_symbol_textmu: { TS1 } { "00B5 }  
20      }  
21      { \cs_new_eq:NN \__siunitx_symbol_textmu: \textmu }  
22    }  
23 }
```

(End definition for `__siunitx_symbol_textmu:`.)

```
\__siunitx_symbol_non_latin:n As in siunitx-unit, but internal in both cases as it's rather specialised.
```

```
\__siunitx_symbol_non_latin:nnnn  
24 \bool_lazy_or:nnTF  
25 { \sys_if_engine luatex_p: }  
26 { \sys_if_engine xetex_p: }  
27 {  
28   \cs_new:Npn \__siunitx_symbol_non_latin:n #1  
29   { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }  
30 }  
31 {
```

```

32 \cs_new:Npn \__siunitx_symbol_non_latin:n #1
33 {
34   \exp_last_unbraced:Nf \__siunitx_symbol_non_latin:nnnn
35   { \char_codepoint_to_bytes:n {#1} }
36 }
37 \cs_new:Npn \__siunitx_symbol_non_latin:nnnn #1#2#3#4
38 {
39   \exp_after:wN \exp_after:wN \exp_after:wN
40   \exp_not:N \char_generate:nn {#1} { 13 }
41   \exp_after:wN \exp_after:wN \exp_after:wN
42   \exp_not:N \char_generate:nn {#2} { 13 }
43 }
44 }

```

(End definition for `__siunitx_symbol_non_latin:n` and `__siunitx_symbol_non_latin:nnnn`.)

`__siunitx_symbol_if_replace:NnT` A test to see if the unit definition which applies is still one we expect: here that means it is just using a (Unicode) codepoint. The comparison is string-based as `unicode-math` (at least) can alter some of them.

```

45 \prg_new_protected_conditional:Npnn \__siunitx_symbol_if_replace:Nn #1#2 { T , TF }
46 {
47   \group_begin:
48   \tl_set:Nx \l__siunitx_symbol_tmpa_tl { \__siunitx_symbol_non_latin:n {#2} }
49   \protected@edef \l__siunitx_symbol_tmpa_tl
50   { \exp_not:N \mathrm { \l__siunitx_symbol_tmpa_tl } }
51   \keys_set:nn { siunitx } { parse-units = false }
52   \siunitx_unit_format:nN {#1} \l__siunitx_symbol_tmpb_tl
53   \str_if_eq:VVTF \l__siunitx_symbol_tmpa_tl \l__siunitx_symbol_tmpb_tl
54   {
55     \group_end:
56     \prg_return_true:
57   }
58   {
59     \group_end:
60     \prg_return_false:
61   }
62 }

```

(End definition for `__siunitx_symbol_if_replace:NnT`.)

`__siunitx_symbol_text_only:n` A short auxiliary to set up text-only symbols.

```

63 \cs_new_protected:Npn \__siunitx_symbol_text_only:n #1
64 {
65   \mode_if_math:TF
66   { \mbox }
67   { \use:n }
68   {#1}
69 }

```

(End definition for `__siunitx_symbol_text_only:n`.)

At the start of the document, fonts are fixed and the user may have altered unit set up. If things are unchanged, we can alter the settings such that they use something “more sensible”.

```

70 \AtBeginDocument
71 {

```

For `\angstrom`, direct input works in text mode so there is only a need to tidy up for math mode. If `fontspec` is loaded then that problem goes away, so nothing needs to be done.

```

72  \__siunitx_symbol_if_replace:NnT \angstrom { "00C5 }
73  {
74    \ifpackageloaded { fontspec }
75    { }
76    {
77      \siunitx_declare_unit:Nx \angstrom
78      {
79        \__siunitx_symbol_text_only:n
80        { \__siunitx_symbol_non_latin:n { "00C5 } }
81      }
82    }
83  }
84  \__siunitx_symbol_if_replace:NnT \ohm { "03A9 }
85  {
86    \siunitx_declare_unit:Nx \ohm
87    {
88      \bool_lazy_or:nnTF
89      { \sys_if_engine luatex_p: }
90      { \sys_if_engine xetex_p: }
91      {
92        \__siunitx_symbol_text_only:n
93        { \__siunitx_symbol_non_latin:n { "03A9 } }
94      }
95      {
96        \exp_not:N \ensuremath
97        {
98          \cs_if_exist:NTF \upOmega
99          { \exp_not:N \upOmega }
100         { \exp_not:N \Omega }
101        }
102      }
103    }
104  }
105  \__siunitx_symbol_if_replace:NnT \micro { "03BC }
106  {
107    \exp_args:NNnx \siunitx_declare_prefix:Nnn \micro { -6 }
108    {
109      \__siunitx_symbol_text_only:n
110      {
111        \bool_lazy_or:nnTF
112        { \sys_if_engine luatex_p: }
113        { \sys_if_engine xetex_p: }
114        { \__siunitx_symbol_non_latin:n { "00B5 } }
115        { \exp_not:N \__siunitx_symbol_textmu: }
116      }
117    }
118  }
119  }

```

`__siunitx_symbol_texttimes:` For the times symbol, only LuaTeX allows us to use the math mode symbol directly. However, that likely won't follow the surrounding font appearance, so in all cases we load

the TS1 version for text. Otherwise much the same as `\textmu` support.

```

120 \AtBeginDocument
121 {
122   \cs_if_free:cT { T@TS1 }
123   {
124     \DeclareFontEncoding { TS1 } { } { }
125     \DeclareFontSubstitution { TS1 } { cmr } { m } { n }
126   }
127   \cs_if_free:NTF \texttimes
128   {
129     \DeclareTextSymbolDefault \__siunitx_symbol_texttimes: { TS1 }
130     \DeclareTextSymbol \__siunitx_symbol_texttimes: { TS1 } { "00D6 }
131   }
132   { \cs_new_eq:NN \__siunitx_symbol_texttimes: \texttimes }
133   \group_begin:
134     \tl_set:Nn \l__siunitx_symbol_tmpa_tl
135     { { } { } { } { 2 } { } { } { } { } { 1 } }
136     \tl_set:Nx \l__siunitx_symbol_tmpa_tl
137     { \siunitx_number_format:N \l__siunitx_symbol_tmpa_tl }
138     \tl_set:Nn \l__siunitx_symbol_tmpb_tl { 2 \times 10 ^ { 1 } }
139     \tl_if_eq:NNTF \l__siunitx_symbol_tmpa_tl \l__siunitx_symbol_tmpb_tl
140     {
141       \group_end:
142       \keys_set:nn { siunitx }
143       {
144         exponent-product =
145         \ifmmode \times \else \__siunitx_symbol_texttimes: \fi
146       }
147     }
148     { \group_end: }
149   }

```

(End definition for `__siunitx_symbol_texttimes:`.)

```

150 \endpackage

```

Part VI

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>
```

Scratch space.

```
\l__siunitx_table_tmp_box
\l__siunitx_table_tmp_dim
\l__siunitx_table_tmp_tl
```

```
3 \box_new:N \l__siunitx_table_tmp_box
4 \dim_new:N \l__siunitx_table_tmp_dim
5 \tl_new:N \l__siunitx_table_tmp_tl
```

(End definition for `\l__siunitx_table_tmp_box`, `\l__siunitx_table_tmp_dim`, and `\l__siunitx_table_tmp_tl`.)

1.1 Interface functions

Used to track that a cell is purely text.

```
6 \bool_new:N \l__siunitx_table_text_bool
```

(End definition for `\l__siunitx_table_text_bool`.)

```
\siunitx_cell_begin:w
\siunitx_cell_end:
```

```
7 \cs_new_protected:Npn \siunitx_cell_begin:w
8 {
9   \bool_set_false:N \l__siunitx_table_text_bool
10  \bool_if:NTF \l_siunitx_number_parse_bool
11    { \__siunitx_table_collect_begin: }
12    { \__siunitx_table_direct_begin: }
13 }
14 \cs_new_protected:Npn \siunitx_cell_end:
15 {
16   \bool_if:NF \l__siunitx_table_text_bool
17   {
18     \bool_if:NTF \l_siunitx_number_parse_bool
19       { \__siunitx_table_collect_end: }
20       { \__siunitx_table_direct_end: }
21   }
22 }
```

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

1.2 Collecting tokens

`\l__siunitx_table_collect_tl` Space for tokens.

```
23 \tl_new:N \l__siunitx_table_collect_tl
```

(End definition for `\l__siunitx_table_collect_tl`.)

`__siunitx_table_collect_begin:`
`__siunitx_table_collect_begin:w`

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 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`. We use an auxiliary to fish out the `\ignorespaces` from the template: that has to go to avoid issues with the peek-ahead code (everything before the `#` needs to be read *before* the Appendix D trick gets applied). Some packages add additional tokens before the `\ignorespaces`, which are dealt with by the delimited argument.

```
24 \cs_new_protected:Npn \__siunitx_table_collect_begin:
25 {
26   \tl_clear:N \l__siunitx_table_collect_tl
27   \if_false: { \fi:
28     \cs_set_protected:Npn \cr
29     {
30       \__siunitx_table_collect_loop:
31       \tex_cr:D
32     }
33   \if_false: } \fi:
34   \__siunitx_table_collect_begin:w
35 }
36 \cs_new_protected:Npn \__siunitx_table_collect_begin:w #1 \ignorespaces
37 { \__siunitx_table_collect_loop: #1 }
```

(End definition for `__siunitx_table_collect_begin:` and `__siunitx_table_collect_begin:w`.)

`__siunitx_table_collect_loop:`
`__siunitx_table_collect_group:n`
`__siunitx_table_collect_token:N`
`__siunitx_table_collect_search:NnF`
`__siunitx_table_collect_search_aux:NNn`

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.

```
38 \cs_new_protected:Npn \__siunitx_table_collect_loop:
39 {
40   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
41   { \__siunitx_table_collect_group:n }
42   { \__siunitx_table_collect_token:N }
43 }
44 \cs_new_protected:Npn \__siunitx_table_collect_group:n #1
45 {
46   \tl_put_right:Nn \l__siunitx_table_collect_tl { {#1} }
47   \__siunitx_table_collect_loop:
48 }
49 \cs_new_protected:Npn \__siunitx_table_collect_token:N #1
```

```

50 {
51   \__siunitx_table_collect_search:NnF #1
52   {
53     \unskip          { \__siunitx_table_collect_loop: }
54     \end              { \tabularnewline \end }
55     \relax           { \relax }
56     \tabularnewline  { \tabularnewline }
57     \siunitx_cell_end: { \siunitx_cell_end: }
58   }
59   {
60     \tl_put_right:Nn \l__siunitx_table_collect_tl {#1}
61     \__siunitx_table_collect_loop:
62   }
63 }
64 \AtBeginDocument
65 {
66   \ifpackageloaded { mdwtab }
67   {
68     \cs_set_protected:Npn \__siunitx_table_collect_token:N #1
69     {
70       \__siunitx_table_collect_search:NnF #1
71       {
72         \@maybe@unskip { \__siunitx_table_collect_loop: }
73         \tab@setcr      { \__siunitx_table_collect_loop: }
74         \unskip         { \__siunitx_table_collect_loop: }
75         \end            { \tabularnewline \end }
76         \relax         { \relax }
77         \tabularnewline { \tabularnewline }
78         \siunitx_cell_end: { \siunitx_cell_end: }
79       }
80       {
81         \tl_put_right:Nn \l__siunitx_table_collect_tl {#1}
82         \__siunitx_table_collect_loop:
83       }
84     }
85   }
86   { }
87 }
88 \cs_new_protected:Npn \__siunitx_table_collect_search:NnF #1#2#3
89 {
90   \__siunitx_table_collect_search_aux:NNn #1
91   #2
92   #1 {#3}
93   \q_stop
94 }
95 \cs_new_protected:Npn \__siunitx_table_collect_search_aux:NNn #1#2#3
96 {
97   \token_if_eq_meaning:NNTF #1 #2
98   { \use_i_delimit_by_q_stop:nw {#3} }
99   { \__siunitx_table_collect_search_aux:NNn #1 }
100 }

```

(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_before_tl Space for tokens.
\l__siunitx_table_number_tl 101 \tl_new:N \l__siunitx_table_before_tl
\l__siunitx_table_after_tl 102 \tl_new:N \l__siunitx_table_number_tl
103 \tl_new:N \l__siunitx_table_after_tl

(End definition for \l__siunitx_table_before_tl, \l__siunitx_table_number_tl, and \l__siunitx_
table_after_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.
104 \cs_new_protected:Npn \__siunitx_table_collect_end:
105 {
106   \protected@edef \l__siunitx_table_collect_tl
107   { \l__siunitx_table_collect_tl }
108   \exp_args:NV \__siunitx_table_split:nNNN
109   \l__siunitx_table_collect_tl
110   \l__siunitx_table_before_tl
111   \l__siunitx_table_number_tl
112   \l__siunitx_table_after_tl
113   \tl_if_empty:NTF \l__siunitx_table_number_tl
114   { \__siunitx_table_print_text:V \l__siunitx_table_before_tl }
115   {
116     \__siunitx_table_print:VVV
117     \l__siunitx_table_before_tl
118     \l__siunitx_table_number_tl
119     \l__siunitx_table_after_tl
120   }
121 }

(End definition for \__siunitx_table_collect_end:.)

\__siunitx_table_split:nNNN Splitting into parts uses the fact that numbers cannot contain groups and that we can
track where we are up to based on the content of the token lists.
\__siunitx_table_split_loop:NNN
\__siunitx_table_split_group:NNNn
\__siunitx_table_split_token:NNNN
122 \cs_new_protected:Npn \__siunitx_table_split:nNNN #1#2#3#4
123 {
124   \tl_clear:N #2
125   \tl_clear:N #3
126   \tl_clear:N #4
127   \__siunitx_table_split_loop:NNN #2#3#4 #1 \q_recursion_tail \q_recursion_stop
128   \__siunitx_table_split_tidy:N #2
129   \__siunitx_table_split_tidy:N #4
130 }
131 \cs_new_protected:Npn \__siunitx_table_split_loop:NNN #1#2#3
132 {
133   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
134   { \__siunitx_table_split_group:NNNn #1#2#3 }
135   { \__siunitx_table_split_token:NNNN #1#2#3 }
136 }
137 \cs_new_protected:Npn \__siunitx_table_split_group:NNNn #1#2#3#4
138 {

```

```

139   \tl_if_empty:NTF #2
140     { \tl_put_right:Nn #1 { {#4} } }
141     { \tl_put_right:Nn #3 { {#4} } }
142   \__siunitx_table_split_loop:NNN #1#2#3
143 }
144 \cs_new_protected:Npn \__siunitx_table_split_token:NNNN #1#2#3#4
145 {
146   \quark_if_recursion_tail_stop:N #4
147   \tl_if_empty:NTF \l__siunitx_table_after_tl
148   {
149     \siunitx_if_number_token:NTF #4
150     { \tl_put_right:Nn #2 {#4} }
151     {
152       \tl_if_empty:NTF #2
153       { \tl_put_right:Nn #1 {#4} }
154       { \tl_put_right:Nn #3 {#4} }
155     }
156   }
157   { \tl_put_right:Nn #3 {#4} }
158   \__siunitx_table_split_loop:NNN #1#2#3
159 }

```

(End definition for `__siunitx_table_split:nNNN` and others.)

```

\__siunitx_table_split_tidy:N
\__siunitx_table_split_tidy:Nn
\__siunitx_table_split_tidy:NV

```

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).

```

160 \cs_new_protected:Npn \__siunitx_table_split_tidy:N #1
161 {
162   \tl_if_empty:NF #1
163   { \__siunitx_table_split_tidy:NV #1 #1 }
164 }
165 \cs_new_protected:Npn \__siunitx_table_split_tidy:Nn #1#2
166 {
167   \str_if_eq:onT { \exp_after:wN { \use:n #2 } } {#2}
168   { \tl_set:Nn #1 { \use:n #2 } }
169 }
170 \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 L^AT_EX 2_ε 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.

```

171 \cs_new_protected:Npn \_siunitx\_table\_skip:n #1
172 {
173   \skip_horizontal:n {#1}
174   \tex_kern:D \c_zero_skip
175 }

```

(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
176 \keys_define:nn { siunitx }
177 {
178   table-column-width .dim_set:N =
179     \l\_siunitx\_table\_column\_width\_dim ,
180   table-fixed-width .bool_set:N =
181     \l\_siunitx\_table\_fixed\_width\_bool
182 }

```

(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
183 \cs_new_protected:Npn \_siunitx\_table\_align\_center:n #1
184 { \_siunitx\_table\_align\_auxi:nn {#1} { Opt~plus~0.5fill } }
185 \cs_new_protected:Npn \_siunitx\_table\_align\_left:n #1
186 { \_siunitx\_table\_align\_auxi:nn {#1} { Opt } }
187 \cs_new_protected:Npn \_siunitx\_table\_align\_right:n #1
188 { \_siunitx\_table\_align\_auxi:nn {#1} { Opt~plus~1fill } }
189 \cs_new_protected:Npn \_siunitx\_table\_align\_auxi:nn #1#2
190 {
191   \bool_if:NTF \l\_siunitx\_table\_fixed\_width\_bool
192     { \hbox_to_wd:nn \l\_siunitx\_table\_column\_width\_dim }
193     { \use:n }
194     {
195       \_siunitx\_table\_skip:n {#2}
196       #1
197       \_siunitx\_table\_skip:n { Opt~plus~1fill - #2 }
198     }
199 }
200 \AtBeginDocument
201 {
202   \@ifpackageloaded { colortbl }
203   {
204     \cs_new_eq:NN

```

```

205         \_siunitx_table_align_auxii:nn
206         \_siunitx_table_align_auxi:nn
207     \cs_set_protected:Npn \_siunitx_table_align_auxi:nn #1#2
208     {
209         \_siunitx_table_skip:n{ Opt~plus~-0.5fill }
210         \_siunitx_table_align_auxii:nn {#1} {#2}
211         \_siunitx_table_skip:n { Opt~plus~-0.5fill }
212     }
213 }
214 { }
215 }

```

(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.

```

216 \keys_define:nn { siunitx }
217 {
218     table-text-alignment .choices:nn =
219     { center , left , right }
220     { \tl_set:Nn \l_siunitx_table_align_text_tl {#1} } ,
221 }
222 \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
223 \cs_new_protected:Npn \_siunitx_table_print_text:n #1
224 {
225     \bool_set_true:N \l_siunitx_table_text_bool
226     \use:c { \_siunitx_table_align_ \l_siunitx_table_align_text_tl :n } {#1}
227 }
228 \cs_generate_variant:Nn \_siunitx_table_print_text:n { V }

```

(End definition for `_siunitx_table_print_text:n`.)

1.6 Number alignment: core ideas

`\l_siunitx_table_integer_box` Boxes for the content before and after the decimal marker.

```

\_siunitx_table_decimal_box
229 \box_new:N \l_siunitx_table_integer_box
230 \box_new:N \l_siunitx_table_decimal_box

```

(End definition for `\l_siunitx_table_integer_box` and `\l_siunitx_table_decimal_box`.)

`_siunitx_table_fil:` A primitive renamed.

```

231 \cs_new_eq:NN \_siunitx_table_fil: \tex_hfil:D

```

(End definition for `_siunitx_table_fil:`.)

`_siunitx_table_cleanup_decimal:w` To remove the excess marker tokens in a decimal part.

```

232 \cs_new:Npn \_siunitx_table_cleanup_decimal:w
233   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
234   { #1#2#3#4#5#6#7 }

```

(End definition for `_siunitx_table_cleanup_decimal:w`.)

`_siunitx_table_center_marker:` When centering on the decimal marker, the easiest approach is to simply re-box the two parts. That is needed whether or not we are parsing numbers, so is best as a short auxiliary.

```

235 \cs_new_protected:Npn \_siunitx_table_center_marker:
236   {
237     \dim_compare:nNnTF
238       { \box_wd:N \l__siunitx_table_integer_box }
239       > { \box_wd:N \l__siunitx_table_decimal_box }
240       {
241         \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
242         { \box_wd:N \l__siunitx_table_integer_box }
243         {
244           \hbox_unpack:N \l__siunitx_table_decimal_box
245           \_siunitx_table_fil:
246         }
247       }
248       {
249         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
250         { \box_wd:N \l__siunitx_table_decimal_box }
251         {
252           \_siunitx_table_fil:
253           \hbox_unpack:N \l__siunitx_table_integer_box
254         }
255       }
256   }

```

(End definition for `_siunitx_table_center_marker:`.)

`\l__siunitx_table_auto_round_bool` Options for tables with defined space.

```

\l__siunitx_table_align_mode_tl
\l__siunitx_table_align_number_tl
257 \keys_define:nn { siunitx }
258   {
259     table-alignment .meta:n =
260       { table-number-alignment = #1 , table-text-alignment = #1 },
261     table-alignment-mode .choices:nn =
262       { none , format , marker }
263       { \tl_set_eq:NN \l__siunitx_table_align_mode_tl \l_keys_choice_tl } ,
264     table-auto-round .bool_set:N =
265       \l__siunitx_table_auto_round_bool ,
266     table-format .code:n =
267       {
268         \_siunitx_table_split:nNNN {#1}
269         \l__siunitx_table_before_model_tl
270         \l__siunitx_table_model_tl
271         \l__siunitx_table_after_model_tl
272         \exp_args:NV \_siunitx_table_generate_model:n \l__siunitx_table_model_tl
273         \tl_set:Nn \l__siunitx_table_align_mode_tl { format }
274       } ,

```

```

275     table-number-alignment .choices:nn =
276     { center , left , right }
277     { \tl_set_eq:NN \l__siunitx_table_align_number_tl \l_keys_choice_tl }
278   }
279   \tl_new:N \l__siunitx_table_align_mode_tl
280   \tl_new:N \l__siunitx_table_align_number_tl

(End definition for \l__siunitx_table_auto_round_bool, \l__siunitx_table_align_mode_tl, and
\l__siunitx_table_align_number_tl.)

```

\l__siunitx_table_format_tl The input and output versions of the model entry in a table.

```

\l__siunitx_table_model_tl
281 \tl_new:N \l__siunitx_table_format_tl
282 \tl_new:N \l__siunitx_table_before_model_tl
283 \tl_new:N \l__siunitx_table_model_tl
284 \tl_new:N \l__siunitx_table_after_model_tl

(End definition for \l__siunitx_table_format_tl and \l__siunitx_table_model_tl.)

```

__siunitx_table_generate_model:n Creating a model for a table at this stage means parsing the format and converting that to an appropriate model. Things are quite straight-forward other than the uncertainty part. At this stage there is no point in formatting the model: that has to happen at point-of-use.

```

285 \cs_new_protected:Npn \__siunitx_table_generate_model:n #1
286 {
287   \group_begin:
288     \bool_set_true:N \l_siunitx_number_parse_bool
289     \keys_set:nn { siunitx } { track-explicit-plus = true }
290     \siunitx_number_parse:nN {#1} \l__siunitx_table_format_tl
291   \exp_args:NNNV \group_end:
292   \tl_set:Nn \l__siunitx_table_format_tl \l__siunitx_table_format_tl
293   \tl_if_empty:NF \l__siunitx_table_format_tl
294   {
295     \exp_after:wN \__siunitx_table_generate_model:nnnnnnn
296     \l__siunitx_table_format_tl
297   }
298 }
299 \cs_new_protected:Npn \__siunitx_table_generate_model:nnnnnnn #1#2#3#4#5#6#7
300 {
301   \tl_set:Nx \l__siunitx_table_model_tl
302   {
303     \exp_not:n { {#1} {#2} }
304     { \prg_replicate:nn {#3} { 8 } }
305     { \prg_replicate:nn { 0 #4 } { 8 } }
306     {
307       \tl_if_blank:NF {#5}
308       {
309         \use:c { __siunitx_table_generate_model_ \tl_head:n {#5} :nw }
310         #5
311       }
312     }
313     \exp_not:n { {#6} }
314     {
315       \int_compare:nNnTF {#7} = 0
316       { 0 }

```

```

317         { \prg_replicate:nn {#7} { 8 } }
318     }
319 }
320 }
321 \cs_new:Npn \__siunitx_table_generate_model_S:nw #1#2
322 { { S } { \prg_replicate:nn {#2} { 8 } } }

```

(End definition for `__siunitx_table_generate_model:n`, `__siunitx_table_generate_model:nnnnnnn`, and `__siunitx_table_generate_model_S:nw`.)

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 `dcolumn`.

After removing the `\ignorespaces` at the start of the cell (see comments for `__siunitx_table_collect_begin:N`), check to see if there is a `{` and branch as appropriate.

```

\__siunitx_table_direct_begin:
\__siunitx_table_direct_begin:w
\__siunitx_table_direct_end:
\__siunitx_table_direct_marker:
\__siunitx_table_direct_marker_switch:
\__siunitx_table_direct_marker_end:
\__siunitx_table_direct_format:
\__siunitx_table_direct_format:nnnnnnn
\__siunitx_table_direct_format:w
\__siunitx_table_direct_format_switch:
\__siunitx_table_direct_format_end:
\__siunitx_table_direct_none:
\__siunitx_table_direct_none_end:
323 \cs_new_protected:Npn \__siunitx_table_direct_begin:
324 { \__siunitx_table_direct_begin:w }
325 \cs_new_protected:Npn \__siunitx_table_direct_begin:w #1 \ignorespaces
326 {
327   #1
328   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
329   { \__siunitx_table_print_text:n }
330   {
331     \m@th
332     \use:c { __siunitx_table_direct_ \l__siunitx_table_align_mode_tl : }
333   }
334 }
335 \cs_new_protected:Npn \__siunitx_table_direct_end:
336 { \use:c { __siunitx_table_direct_ \l__siunitx_table_align_mode_tl _end: } }

```

When centring the content about a decimal marker, the trick is to collect everything into two boxes and then compare the sizes. As we are always in math mode, we can use a math active token to make the switch. The up-front setting of the decimal box deals with the case where there is no decimal part.

```

337 \cs_new_protected:Npn \__siunitx_table_direct_marker:
338 {
339   \hbox_set:Nn \l__siunitx_table_tmp_box
340   { \ensuremath { \mathord { \l__siunitx_number_output_decimal_tl } } }
341   \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
342   { \box_wd:N \l__siunitx_table_tmp_box }
343   { \__siunitx_table_fil: }
344   \hbox_set:Nw \l__siunitx_table_integer_box
345   \c_math_toggle_token
346   \tl_map_inline:Nn \l__siunitx_number_input_decimal_tl
347   {
348     \char_set_active_eq:NN ##1 \__siunitx_table_direct_marker_switch:
349     \char_set_mathcode:nn { '##1 } { "8000 }
350   }
351 }
352 \cs_new_protected:Npn \__siunitx_table_direct_marker_switch:
353 {

```

```

354     \c_math_toggle_token
355   \hbox_set_end:
356   \hbox_set:Nw \l__siunitx_table_decimal_box
357     \c_math_toggle_token
358     \l_siunitx_number_output_decimal_tl
359 }
360 \cs_new_protected:Npn \__siunitx_table_direct_marker_end:
361 {
362   \c_math_toggle_token
363   \hbox_set_end:
364   \__siunitx_table_center_marker:
365   \box_use_drop:N \l__siunitx_table_integer_box
366   \box_use_drop:N \l__siunitx_table_decimal_box
367 }

```

For the version where there is space reserved, first format and decompose that, then create appropriately-sized boxes.

```

368 \cs_new_protected:Npn \__siunitx_table_direct_format:
369 {
370   \tl_set:Nx \l__siunitx_table_tmp_tl
371   { \siunitx_number_format:NN \l__siunitx_table_model_tl \q_nil }
372   \exp_after:wN \__siunitx_table_direct_format_aux:w
373   \l__siunitx_table_tmp_tl \q_stop
374 }
375 \cs_new_protected:Npn \__siunitx_table_direct_format_aux:w
376 #1 \q_nil #2 \q_nil #3 \q_stop
377 {
378   \hbox_set:Nn \l__siunitx_table_tmp_box
379   { \ensuremath { \__siunitx_table_cleanup_decimal:w #3 } }
380   \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
381   { \box_wd:N \l__siunitx_table_tmp_box }
382   { \__siunitx_table_fil: }
383   \hbox_set:Nn \l__siunitx_table_tmp_box { \ensuremath { #1#2 } }
384   \hbox_set_to_wd:Nnw \l__siunitx_table_integer_box
385   { \box_wd:N \l__siunitx_table_tmp_box }
386   \c_math_toggle_token
387   \tl_map_inline:Nn \l_siunitx_number_input_decimal_tl
388   {
389     \char_set_active_eq:NN ##1 \__siunitx_table_direct_format_switch:
390     \char_set_mathcode:nn { '##1 } { "8000 }
391   }
392   \__siunitx_table_fil:
393 }
394 \cs_new_protected:Npn \__siunitx_table_direct_format_switch:
395 {
396   \c_math_toggle_token
397   \hbox_set_end:
398   \hbox_set_to_wd:Nnw \l__siunitx_table_decimal_box
399   { \box_wd:N \l__siunitx_table_decimal_box }
400   \c_math_toggle_token
401   \mathord { \l_siunitx_number_output_decimal_tl }
402 }
403 \cs_new_protected:Npn \__siunitx_table_direct_format_end:
404 {

```

```

405     \c_math_toggle_token
406     \__siunitx_table_fil:
407 \hbox_set_end:
408 \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
409 {
410     \box_use_drop:N \l__siunitx_table_integer_box
411     \box_use_drop:N \l__siunitx_table_decimal_box
412 }
413 }

```

No parsing and no alignment is easy.

```

414 \cs_new_protected:Npn \__siunitx_table_direct_none: { \c_math_toggle_token }
415 \cs_new_protected:Npn \__siunitx_table_direct_none_end: { \c_math_toggle_token }

```

(End definition for __siunitx_table_direct_begin: and others.)

1.8 Printing numbers in cells: main functions

For alignment of text outside of a number.

```

416 \box_new:N \l__siunitx_table_before_box
417 \box_new:N \l__siunitx_table_after_box

```

(End definition for \l__siunitx_table_before_box and \l__siunitx_table_after_box.)

Used to “carry forward” the amount of white space which needs to be inserted after the decimal marker.

```

418 \dim_new:N \l__siunitx_table_carry_dim

```

(End definition for \l__siunitx_table_carry_dim.)

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

```

419 \keys_define:nn { siunitx }
420 {
421     table-align-comparator .bool_set:N =
422     \l__siunitx_table_align_comparator_bool ,
423     table-align-exponent .bool_set:N =
424     \l__siunitx_table_align_exponent_bool ,
425     table-align-text-after .bool_set:N =
426     \l__siunitx_table_align_after_bool ,
427     table-align-text-before .bool_set:N =
428     \l__siunitx_table_align_before_bool ,
429     table-align-uncertainty .bool_set:N =
430     \l__siunitx_table_align_uncertainty_bool
431 }

```

(End definition for \l__siunitx_table_align_comparator_bool and others.)

```

\__siunitx_table_print:nnn
\__siunitx_table_print:VVV
432 \cs_new_protected:Npn \__siunitx_table_print:nnn #1#2#3
433 { \use:c { __siunitx_table_print_ \l__siunitx_table_align_mode_tl :nnn } {#1} {#2} {#3} }
434 \cs_generate_variant:Nn \__siunitx_table_print:nnn { VVV }

\__siunitx_table_print_marker:nnn
\__siunitx_table_print_marker_auxi:w
\__siunitx_table_print_marker_auxii:w
\__siunitx_table_print_marker_auxiii:w
\__siunitx_table_print_format_after:N
\__siunitx_table_print_format_box:Nn
\__siunitx_table_print_none:nnn

```

When centering on the decimal marker, alignment is relatively simple, and close in concept to that used without parsing. First we need to deal with any text before or after the number. For text *before*, there's the case where it has no width and might be a font or color change: that has to be filtered out first. Then we can adjust the size of this material and that after the number such that they are equal. The number itself can then be formatted, splitting at the decimal marker. A bit more size adjustment, then the number itself and any text at the end can be inserted.

```

435 \cs_new_protected:Npn \__siunitx_table_print_marker:nnn #1#2#3
436 {
437   \hbox_set:Nn \l__siunitx_table_before_box {#1}
438   \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { Opt }
439   {
440     \box_clear:N \l__siunitx_table_before_box
441     #1
442   }
443   \hbox_set:Nn \l__siunitx_table_after_box {#3}
444   \dim_compare:nNnTF
445     { \box_wd:N \l__siunitx_table_after_box }
446     > { \box_wd:N \l__siunitx_table_before_box }
447     {
448       \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
449       { \box_wd:N \l__siunitx_table_after_box }
450       {
451         \__siunitx_table_fil:
452         \hbox_unpack:N \l__siunitx_table_before_box
453       }
454     }
455     {
456       \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
457       { \box_wd:N \l__siunitx_table_before_box }
458       {
459         \hbox_unpack:N \l__siunitx_table_after_box
460         \__siunitx_table_fil:
461       }
462     }
463     \box_use_drop:N \l__siunitx_table_before_box
464     \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
465     \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
466     \tl_set:Nx \l__siunitx_table_tmp_tl
467       { \siunitx_number_format:NN \l__siunitx_table_tmp_tl \q_nil }
468     \exp_after:wN \__siunitx_table_print_marker:w
469       \l__siunitx_table_tmp_tl \q_stop
470     \box_use_drop:N \l__siunitx_table_after_box
471   }
472 \cs_new_protected:Npn \__siunitx_table_print_marker:w
473   #1 \q_nil #2 \q_nil #3 \q_stop
474   {
475     \hbox_set:Nn \l__siunitx_table_integer_box
476       { \siunitx_print:nn { number } { #1#2 } }
477     \hbox_set:Nn \l__siunitx_table_decimal_box
478       { \siunitx_print:nn { number } { \__siunitx_table_cleanup_decimal:w #3 } }
479     \__siunitx_table_center_marker:
480     \box_use_drop:N \l__siunitx_table_integer_box

```

```

481 \box_use_drop:N \l__siunitx_table_decimal_box
482 }

```

For positioning based on a format, we have to work part-by-part as there are a number of alignment points to get right. As for the `marker` approach, first we check if the material before the numerical content is of zero width. Next we need to format the model and content numbers, before starting an auxiliary chain to pick out the various parts in order.

```

483 \cs_new_protected:Npn \__siunitx_table_print_format:nnn #1#2#3
484 {
485   \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_before_model_tl }
486   \hbox_set:Nn \l__siunitx_table_before_box {#1}
487   \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { 0pt }
488   {
489     \box_clear:N \l__siunitx_table_before_box
490     #1
491   }
492   \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
493   { \box_wd:N \l__siunitx_table_tmp_box }
494   {
495     \__siunitx_table_fil:
496     \hbox_unpack:N \l__siunitx_table_before_box
497   }
498   \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
499   \group_begin:
500     \bool_if:NT \l__siunitx_table_auto_round_bool
501     {
502       \exp_args:Nx \keys_set:nn { siunitx }
503       {
504         round-mode      = places ,
505         round-pad       = true   ,
506         round-precision =
507           \exp_after:wN \__siunitx_table_print_format:nnnnnn
508             \l__siunitx_table_format_tl
509       }
510     }
511     \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
512     \exp_args:NNNV \group_end:
513     \tl_set:Nn \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
514     \tl_set:Nx \l__siunitx_table_tmp_tl
515     {
516       \siunitx_number_format:NN \l__siunitx_table_model_tl \q_nil
517       \exp_not:N \q_mark
518       \siunitx_number_format:NN \l__siunitx_table_tmp_tl \q_nil
519     }
520     \exp_after:wN \__siunitx_table_print_format_auxi:w
521     \l__siunitx_table_tmp_tl \q_stop
522     \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_after_model_tl }
523     \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
524     { \box_wd:N \l__siunitx_table_tmp_box + \l__siunitx_table_carry_dim }
525     {
526       \bool_if:NT \l__siunitx_table_align_after_bool
527       { \skip_horizontal:n { \l__siunitx_table_carry_dim } }
528       #3
529       \__siunitx_table_fil:

```

```

530     }
531     \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
532     {
533         \box_use_drop:N \l__siunitx_table_before_box
534         \box_use_drop:N \l__siunitx_table_integer_box
535         \box_use_drop:N \l__siunitx_table_decimal_box
536         \box_use_drop:N \l__siunitx_table_after_box
537     }
538 }
539 \cs_new:Npn \__siunitx_table_print_format:nnnnnn #1#2#3#4#5#6#7
540 { 0 #4 }

```

The first numerical part to handle is the comparator. Any white space we need to add goes into the text part *if* alignment is not active (*i.e.* we are looking “backwards” to place this filler).

```

541 \cs_new_protected:Npn \__siunitx_table_print_format_auxi:w
542   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
543   {
544     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1}
545     \bool_if:NTF \l__siunitx_table_align_before_bool
546     {
547       \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
548       { \box_wd:N \l__siunitx_table_tmp_box }
549       {
550         \__siunitx_table_fil:
551         \tl_if_blank:nF {#3}
552         { \siunitx_print:nn { number } {#3} }
553       }
554     }
555     {
556       \__siunitx_table_print_format_box:Nn \l__siunitx_table_integer_box {#3}
557       \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
558       {
559         \box_wd:N \l__siunitx_table_before_box
560         + \box_wd:N \l__siunitx_table_tmp_box
561         - \box_wd:N \l__siunitx_table_integer_box
562       }
563       {
564         \__siunitx_table_fil:
565         \hbox_unpack:N \l__siunitx_table_before_box
566       }
567     }
568     \__siunitx_table_print_format_auxii:w #2 \q_mark #4 \q_stop
569   }

```

The integer part follows much the same pattern, except now it is control of the comparator alignment that determines where the white space goes. As we already have content in the `integer` box, we need to measure how much *extra* material has been added. To avoid using more boxes or re-setting, we do that by recording sizes before and after the change. (In effect, `\l__siunitx_table_tmp_dim` is here “`\l__@@_comparator_dim`”.)

```

570 \cs_new_protected:Npn \__siunitx_table_print_format_auxii:w
571   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
572   {
573     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1}

```



```

574 \bool_lazy_and:nnTF
575 { \l__siunitx_table_align_comparator_bool }
576 { \dim_compare_p:nNn { \box_wd:N \l__siunitx_table_integer_box } > { 0pt } }
577 {
578   \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
579   {
580     \box_wd:N \l__siunitx_table_integer_box
581     + \box_wd:N \l__siunitx_table_tmp_box
582   }
583   {
584     \hbox_unpack:N \l__siunitx_table_integer_box
585     \__siunitx_table_fil:
586     \siunitx_print:nn { number } {#3}
587   }
588 }
589 {
590   \bool_if:NTF \l__siunitx_table_align_before_bool
591   {
592     \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
593     {
594       \box_wd:N \l__siunitx_table_integer_box
595       + \box_wd:N \l__siunitx_table_tmp_box
596     }
597     {
598       \__siunitx_table_fil:
599       \hbox_unpack:N \l__siunitx_table_integer_box
600       \siunitx_print:nn { number } {#3}
601     }
602   }
603   {
604     \dim_set:Nn \l__siunitx_table_tmp_dim
605     { \box_wd:N \l__siunitx_table_integer_box }
606     \hbox_set:Nn \l__siunitx_table_integer_box
607     {
608       \hbox_unpack:N \l__siunitx_table_integer_box
609       \siunitx_print:nn { number } {#3}
610     }
611     \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
612     {
613       \box_wd:N \l__siunitx_table_before_box
614       + \box_wd:N \l__siunitx_table_tmp_box
615       + \l__siunitx_table_tmp_dim
616       - \box_wd:N \l__siunitx_table_integer_box
617     }
618     {
619       \__siunitx_table_fil:
620       \hbox_unpack:N \l__siunitx_table_before_box
621     }
622   }
623 }
624 \__siunitx_table_print_format_auxiii:w #2 \q_mark #4 \q_stop
625 }

```

We now deal with the decimal part: there is nothing already in the decimal box, so the basics are easy. We need to “carry forward” any white space, as where it gets inserted

depends on the options for subsequent parts.

```

626 \cs_new_protected:Npn \__siunitx_table_print_format_auxiii:w
627   #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
628   {
629     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
630     \__siunitx_table_print_format_box:Nn \l__siunitx_table_decimal_box {#4#5}
631     \dim_set:Nn \l__siunitx_table_carry_dim
632       {
633         \box_wd:N \l__siunitx_table_tmp_box
634         - \box_wd:N \l__siunitx_table_decimal_box
635       }
636     \__siunitx_table_print_format_auxiv:w #3 \q_mark #6 \q_stop
637   }

```

Any separated uncertainty is now picked up. That has a number of parts, so the first step is to look for a sign (which will be #1). We then split, either simply tidying up the markers if there is no uncertainty, or setting it.

```

638 \cs_new_protected:Npn \__siunitx_table_print_format_auxiv:w
639   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
640   {
641     \tl_if_blank:nTF {#1}
642       { \__siunitx_table_print_format_auxv:w }
643       { \__siunitx_table_print_format_auxvi:w }
644       #1#2 \q_mark #3#4 \q_stop
645   }
646 \cs_new_protected:Npn \__siunitx_table_print_format_auxv:w
647   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
648   #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
649   { \__siunitx_table_print_format_auxvii:w #4 \q_mark #8 \q_stop }

```

Sorting out the placement of the uncertainty requires both the model and real data widths, so we store the former to avoiding needing more boxes. It's then just a case of putting the carry-over white space in the right place.

```

650 \cs_new_protected:Npn \__siunitx_table_print_format_auxvi:w
651   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
652   #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
653   {
654     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2#3 }
655     \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
656     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #5#6#7 }
657     \__siunitx_table_print_format_after:N \l__siunitx_table_align_uncertainty_bool
658     \__siunitx_table_print_format_auxvii:w #4 \q_mark #8 \q_stop
659   }

```

Finally, we get to the exponent part: the multiplication symbol is #1 and the number itself is #2. The code is almost the same as for uncertainties, which allows a shared auxiliary to be used.

```

660 \cs_new_protected:Npn \__siunitx_table_print_format_auxvii:w
661   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
662   {
663     \tl_if_blank:nF {#2}
664     {
665       \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2 }
666       \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }

```

```

667     \_siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #3#4 }
668     \_siunitx_table_print_format_after:N \l__siunitx_table_align_exponent_bool
669   }
670 }

```

A simple auxiliary to avoid relatively expensive use of the print routine for empty parts.

```

671 \cs_new_protected:Npn \_siunitx_table_print_format_box:Nn #1#2
672 {
673   \hbox_set:Nn #1
674   {
675     \tl_if_blank:nF {#2}
676     { \siunitx_print:nn { number } {#2} }
677   }
678 }

```

A common routine for placing material after the decimal marker and “shuffling”.

```

679 \cs_new_protected:Npn \_siunitx_table_print_format_after:N #1
680 {
681   \bool_if:NTF #1
682   {
683     \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
684     {
685       \box_wd:N \l__siunitx_table_decimal_box
686       + \l__siunitx_table_carry_dim
687       + \box_wd:N \l__siunitx_table_tmp_box
688     }
689     {
690       \hbox_unpack:N \l__siunitx_table_decimal_box
691       \_siunitx_table_fil:
692       \hbox_unpack:N \l__siunitx_table_tmp_box
693     }
694     \dim_set:Nn \l__siunitx_table_carry_dim
695     {
696       \l__siunitx_table_tmp_dim
697       - \box_wd:N \l__siunitx_table_tmp_box
698     }
699   }
700   {
701     \hbox_set:Nn \l__siunitx_table_decimal_box
702     {
703       \hbox_unpack:N \l__siunitx_table_decimal_box
704       \hbox_unpack:N \l__siunitx_table_tmp_box
705     }
706     \dim_add:Nn \l__siunitx_table_carry_dim
707     {
708       \l__siunitx_table_tmp_dim
709       - \box_wd:N \l__siunitx_table_tmp_box
710     }
711   }
712 }

```

With no alignment, everything supplied is treated more-or-less the same as `\num` (but without the `xparse` wrapper).

```

713 \cs_new_protected:Npn \_siunitx_table_print_none:nnn #1#2#3
714 {

```

```

715 \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
716 {
717   #1
718   \siunitx_number_format:nN {#2} \l__siunitx_table_tmp_tl
719   \siunitx_print:nV { number } \l__siunitx_table_tmp_tl
720   #3
721 }
722 }

```

(End definition for `_siunitx_table_print:nnn` and others.)

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.

```

723 \keys_set:nn { siunitx }
724 {
725   table-align-comparator = true   ,
726   table-align-exponent   = true   ,
727   table-align-text-after  = true   ,
728   table-align-text-before = true   ,
729   table-align-uncertainty = true   ,
730   table-alignment        = center ,
731   table-auto-round        = false  ,
732   table-column-width      = 0pt    ,
733   table-fixed-width       = false  ,
734   table-format            = 2.2    ,
735   table-number-alignment  = center ,
736   table-text-alignment    = center ,

```

Out of order as `table-format` sets this implicitly too.

```

737   table-alignment-mode    = marker
738 }
739 \end{package}

```

Part VII

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 commands `\frac`, `\mathrm`, `\mbox`, `\sq` 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> {<power>} {<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. 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.

<code>\siunitx_declare_unit:Nn</code>	<code>\siunitx_declare_unit:Nn <unit> {<meaning>}</code>
<code>\siunitx_declare_unit:Nx</code>	Defines a symbolic <i><unit></i> (which should be a control sequence such as <code>\kilogram</code>) to be converted by the parser to the <i><meaning></i> . 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 <i><meaning></i> will be typeset directly.

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

<code>\l_siunitx_unit_symbolic_seq</code>	This sequence contains all of the symbolic <i><unit></i> names defined : these will be in the form of control sequences such as <code>\kilogram</code> . 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.

<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>	

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

`\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/>	$\langle prefix \rangle \langle unit \rangle \backslash squared$
$\backslash cubed$	$\langle prefix \rangle \langle unit \rangle \backslash cubed$
<hr/>	Pre-defined unit powers which apply to the preceding $\langle prefix \rangle / \langle unit \rangle$ combination.
<hr/>	
$\backslash per$	$\backslash per \langle prefix \rangle \langle unit \rangle \langle power \rangle$
<hr/>	Indicates that the next $\langle prefix \rangle / \langle unit \rangle / \langle power \rangle$ combination is reciprocal, <i>i.e.</i> raises it to the power -1 . This symbolic representation may be applied in addition to a $\backslash power$, and will work correctly if the $\backslash power$ itself is negative. In literal mode $\backslash per$ will print a slash (“/”).
<hr/>	
$\backslash cancel$	$\backslash cancel \langle prefix \rangle \langle unit \rangle \langle power \rangle$
<hr/>	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 $\backslash cancel$, which is assumed to be available at a higher level. In literal mode, the same higher-level $\backslash cancel$ will be applied to the next token. It is the responsibility of the calling code to provide an appropriate definition for $\backslash cancel$ outside of the scope of the unit parser.
<hr/>	
$\backslash highlight$	$\backslash highlight \{ \langle color \rangle \} \langle prefix \rangle \langle unit \rangle \langle power \rangle$
<hr/>	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 $\backslash textcolor$, which is assumed to be available at a higher level. In literal mode, the same higher-level $\backslash textcolor$ will be applied to the next token. It is the responsibility of the calling code to provide an appropriate definition for $\backslash textcolor$ outside of the scope of the unit parser.
<hr/>	
$\backslash of$	$\langle prefix \rangle \langle unit \rangle \langle power \rangle \backslash of \{ \langle qualifier \rangle \}$
<hr/>	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/>	
$\backslash raiseto$ $\backslash tothe$	$\backslash raiseto \{ \langle power \rangle \} \langle prefix \rangle \langle unit \rangle$ $\langle prefix \rangle \langle unit \rangle \backslash tothe \{ \langle power \rangle \}$
<hr/>	Indicates that the $\langle power \rangle$ applies to the current $\langle prefix \rangle / \langle unit \rangle$ combination. As shown, $\backslash raiseto$ applies to the next $\langle unit \rangle$ whereas $\backslash tothe$ applies to the preceding unit. In literal mode the $\backslash power$ will be printed as a superscript attached to the next token ($\backslash raiseto$) or preceding token ($\backslash tothe$) as appropriate.

3.1 Key-value options

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

<hr/> <hr/>	$\text{bracket-unit-denominator} = \text{true false}$
<hr/>	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 $\text{symbol-or-fraction}$). The standard setting is true .

<hr/> <hr/> forbid-literal-units	forbid-literal-units = true false
	Switch which determines if literal units are allowed when parsing is active; does not apply when <code>parse-units</code> is <code>false</code> .
<hr/> <hr/> font-command	font-command = $\langle command \rangle$
	Command applied to text during output of units: should be command usable in math mode for font selection. Notice that in a typical unit this does not (necessarily) apply to all output, for example powers or brackets. The standard setting is <code>\mathrm</code> .
<hr/> <hr/> fraction-command	fraction-command = $\langle command \rangle$
	Command used to create fractional output when <code>per-mode</code> is set to <code>fraction</code> . The standard setting is <code>\frac</code> .
<hr/> <hr/> inter-unit-product	inter-unit-product = $\langle separator \rangle$
	Inserted between unit combinations in parsed mode, and used to replace <code>.</code> and <code>~</code> in literal mode. The standard setting is <code>\,</code> .
<hr/> <hr/> parse-units	parse-units = true false
	Determines whether parsing of unit symbols is attempted or literal mode is used directly. The standard setting is <code>true</code> .
<hr/> <hr/> per-mode	per-mode = fraction power power-positive-first repeated-symbol symbol symbol-or-fraction
	Selects how the negative powers (<code>\per</code>) are formatted: a choice from the options <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> .
<hr/> <hr/> per-symbol	per-symbol = $\langle symbol \rangle$
	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> .
<hr/> <hr/> qualifier-mode	qualifier-mode = bracket combine phrase subscript
	Selects how qualifiers are formatted: a choice from the options <code>bracket</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> joins the material using <code>qualifier-phrase</code> as a link, and <code>subscript</code> formats the qualifier as a subscript. The standard setting is <code>subscript</code> .

<hr/> <hr/> qualifier-phrase	<p>qualifier-phrase = $\langle phrase \rangle$</p> <p>Defines the $\langle phrase \rangle$ used when <code>qualifier-mode</code> is set to <code>phrase</code>.</p>
<hr/> <hr/> sticky-per	<p>sticky-per = true false</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 \TeX <code>\over</code> primitive. The standard setting is <code>false</code>.</p>
<hr/> <hr/> unit-close-bracket	<p>unit-close-bracket = $\langle symbol \rangle$</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	<p>unit-open-bracket = $\langle symbol \rangle$</p> <p>Bracket symbol used to open a matched pair around units when once is required to maintain mathematical logic. The standard setting is <code>(</code>.</p>

4 siunitx-unit implementation

Start the DocStrip guards.

1 $\langle *package \rangle$

Identify the internal prefix (\LaTeX 3 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 $\backslash cs_generate_variant:Nn \backslash tl_replace_all:Nnn \{ NnV \}$

$\backslash l_siunitx_unit_tmp_fp$
 $\backslash l_siunitx_unit_tmp_int$
 $\backslash l_siunitx_unit_tmp_tl$

Scratch space.

4 $\backslash fp_new:N \backslash l_siunitx_unit_tmp_fp$
5 $\backslash int_new:N \backslash l_siunitx_unit_tmp_int$
6 $\backslash tl_new:N \backslash l_siunitx_unit_tmp_tl$

(End definition for $\backslash l_siunitx_unit_tmp_fp$, $\backslash l_siunitx_unit_tmp_int$, and $\backslash l_siunitx_unit_tmp_tl$.)

$\backslash c_siunitx_unit_math_subscript_tl$

Useful tokens with awkward category codes.

7 $\backslash tl_const:Nx \backslash c_siunitx_unit_math_subscript_tl$
8 $\{ \backslash char_generate:nn \{ '_ \} \{ 8 \} \}$

(End definition for $\backslash c_siunitx_unit_math_subscript_tl$.)

$\backslash l_siunitx_unit_parsing_bool$

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 $\backslash bool_new:N \backslash l_siunitx_unit_parsing_bool$

(End definition for `\l__siunitx_unit_parsing_bool`.)

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

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

(End definition for `\l__siunitx_unit_test_bool`.)

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

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

(End definition for `__siunitx_unit_if_symbolic:nTF`.)

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 92.)

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

Importantly, notice that the unit macros are declared as expandable. This is required so that literals can be correctly converted into a token list of material which does not

depend on local redefinitions for the unit macros. That is required so that the unit formatting system can be grouped.

```

22 \cs_new_protected:Npn \__siunitx_unit_set_symbolic:Nnn #1
23   { \__siunitx_unit_set_symbolic:Nnnn #1 { } }
24 \cs_new_protected:Npn \__siunitx_unit_set_symbolic:Npnn #1#2#
25   { \__siunitx_unit_set_symbolic:Nnnn #1 {#2} }
26 \cs_new_protected:Npn \__siunitx_unit_set_symbolic:Nnnn #1#2#3#4
27   {
28     \seq_put_right:Nn \l_siunitx_unit_symbolic_seq {#1}
29     \cs_set:cpn { units ~ > ~ \token_to_str:N #1 } #2
30     {
31       \bool_if:NF \l_siunitx_unit_test_bool
32       {
33         \bool_if:NTF \l_siunitx_unit_parsing_bool
34         {#4}
35         {#3}
36       }
37     }
38   }

```

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

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

```

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 91.)

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

```

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 {#3}
57     \prop_put:Nnn \l_siunitx_unit_prefixes_forward_prop {#3} {#2}
58     \prop_put:Nnn \l_siunitx_unit_prefixes_reverse_prop {#2} {#3}
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 91.)

`\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 91.)

`\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 92.)

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 95.)

`\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 95.)

`\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 95.)

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

```

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 95.)

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

```

97 \keys_define:nn { siunitx }
98 {
99   font-command .tl_set:N =
100     \l_siunitx_unit_font_tl ,
101   inter-unit-product .tl_set:N =
102     \l_siunitx_unit_product_tl
103 }

```

(End definition for `\l_siunitx_unit_font_tl` and `\l_siunitx_unit_product_tl`. This variable is documented on page 92.)

`\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.

```

104 \tl_new:N \l_siunitx_unit_formatted_tl

```


(End definition for \l__siunitx_unit_formatted_tl.)

\siunitx_unit_format:nN
\siunitx_unit_format:nNN
__siunitx_unit_format:nNN
__siunitx_unit_format_aux:

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.

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.

```

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

```

```

140         { \exp_not:V \l__siunitx_unit_formatted_tl }
141         \fp_set:Nn \exp_not:N #3
142         { \fp_use:N \l__siunitx_unit_prefix_fp }
143     }
144     \exp_after:wN \group_end:
145     \__siunitx_unit_format_aux:
146 }
147 \cs_new_protected:Npn \__siunitx_unit_format_aux: { }

```

(End definition for `\siunitx_unit_format:nN` and others. These functions are documented on page 90.)

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.

```

148 \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.

```

149 \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`
`_siunitx_unit_format_literal_auxi:w`
`_siunitx_unit_format_literal_auxii:w`
`_siunitx_unit_format_literal_auxiii:w`
`_siunitx_unit_format_literal_auxiv:w`
`_siunitx_unit_format_literal_auxv:w`
`\l__siunitx_unit_separator_tl`

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.

```

150 \group_begin:
151   \char_set_catcode_active:n { '\~ }
152   \cs_new_protected:Npx \__siunitx_unit_format_literal:n #1
153   {
154     \group_begin:
155     \exp_not:n { \bool_set_false:N \l__siunitx_unit_parsing_bool }
156     \tl_set:Nn \exp_not:N \l__siunitx_unit_tmp_tl {#1}
157     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
158     { \exp_not:N ~ } { . }
159     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
160     { \token_to_str:N ^ } { ^ }
161     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
162     { \token_to_str:N _ } { \c__siunitx_unit_math_subscript_tl }
163     \exp_not:n
164     {
165       \protected@edef \l__siunitx_unit_tmp_tl
166       { \l__siunitx_unit_tmp_tl }
167       \tl_clear:N \l__siunitx_unit_formatted_tl
168       \tl_if_empty:NF \l__siunitx_unit_tmp_tl

```

```

169         {
170             \exp_after:wN \__siunitx_unit_format_literal_auxi:w
171             \l__siunitx_unit_tmp_tl .
172             \q_recursion_tail . \q_recursion_stop
173         }
174         \exp_args:NNNV \group_end:
175         \tl_set:Nn \l__siunitx_unit_formatted_tl
176         \l__siunitx_unit_formatted_tl
177     }
178 }
179 \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.

```

180 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxi:w #1 .
181 {
182     \quark_if_recursion_tail_stop:n {#1}
183     \__siunitx_unit_format_literal_auxii:n {#1}
184     \tl_set_eq:NN \l__siunitx_unit_separator_tl \l__siunitx_unit_product_tl
185     \__siunitx_unit_format_literal_auxi:w
186 }
187 \cs_set_protected:Npn \__siunitx_unit_format_literal_auxii:n #1
188 {
189     \__siunitx_unit_format_literal_auxiii:w
190     #1 / \q_recursion_tail / \q_recursion_stop
191 }
192 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxiii:w #1 /
193 {
194     \quark_if_recursion_tail_stop:n {#1}
195     \__siunitx_unit_format_literal_auxiv:w #1 ^ ^ \q_stop
196     \tl_set:Nn \l__siunitx_unit_separator_tl { / }
197     \__siunitx_unit_format_literal_auxiii:w
198 }

```

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.

```

199 \use:x
200 {
201     \cs_new_protected:Npn \exp_not:N \__siunitx_unit_format_literal_auxiv:w
202     ##1 ^ ##2 ^ ##3 \exp_not:N \q_stop
203     {
204         \exp_not:N \__siunitx_unit_format_literal_auxv:w
205         ##1
206         \c__siunitx_unit_math_subscript_tl
207         \c__siunitx_unit_math_subscript_tl
208         \exp_not:N \q_mark
209         ##2
210         \c__siunitx_unit_math_subscript_tl

```

```

211         \c__siunitx_unit_math_subscript_tl
212         \exp_not:N \q_stop
213     }
214     \cs_new_protected:Npn \exp_not:N \__siunitx_unit_format_literal_auxv:w
215     ##1 \c__siunitx_unit_math_subscript_tl
216     ##2 \c__siunitx_unit_math_subscript_tl ##3
217     \exp_not:N \q_mark
218     ##4 \c__siunitx_unit_math_subscript_tl
219     ##5 \c__siunitx_unit_math_subscript_tl ##6
220     \exp_not:N \q_stop
221     {
222         \tl_set:Nx \exp_not:N \l__siunitx_unit_formatted_tl
223         {
224             \exp_not:N \exp_not:N
225             \exp_not:N \l__siunitx_unit_formatted_tl
226             \exp_not:N \tl_if_empty:NF
227             \exp_not:N \l__siunitx_unit_formatted_tl
228             {
229                 \exp_not:N \exp_not:N
230                 \exp_not:N \l__siunitx_unit_separator_tl
231             }
232             \exp_not:N \tl_if_blank:nF {##1}
233             {
234                 \exp_not:N \exp_not:N
235                 \exp_not:N \l__siunitx_unit_font_tl
236                 { \exp_not:N \exp_not:n {##1} }
237             }
238             \exp_not:N \tl_if_blank:nF {##4}
239             { ^ { \exp_not:N \exp_not:n {##4} } }
240             \exp_not:N \tl_if_blank:nF {##2##5}
241             {
242                 \c__siunitx_unit_math_subscript_tl
243                 {
244                     \exp_not:N \exp_not:N
245                     \exp_not:N \l__siunitx_unit_font_tl
246                     { \exp_not:N \exp_not:n {##2##5} }
247                 }
248             }
249         }
250     }
251 }
252 \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`.

```

253 \keys_define:nn { siunitx }
254 {
255     sticky-per .bool_set:N = \l__siunitx_unit_sticky_per_bool
256 }

```

(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.

```

257 \prop_new:N \l__siunitx_unit_parsed_prop
258 \bool_new:N \l__siunitx_unit_per_bool
259 \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.

```

260 \cs_new_protected:Npn \__siunitx_unit_parse:n #1
261 {
262     \prop_clear:N \l__siunitx_unit_parsed_prop
263     \bool_set_true:N \l__siunitx_unit_parsing_bool
264     \bool_set_false:N \l__siunitx_unit_per_bool
265     \bool_set_false:N \l__siunitx_unit_test_bool
266     \int_zero:N \l__siunitx_unit_position_int
267     #1
268     \int_step_inline:nn \l__siunitx_unit_position_int
269     { \__siunitx_unit_parse_finalise:n {##1} }
270     \__siunitx_unit_parse_finalise:
271 }

```

(End definition for `__siunitx_unit_parse:n`.)

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

```

272 \cs_new_protected:Npn \_siunitx_unit_parse_add:nnnn #1#2#3#4
273 {
274   \tl_set:Nx \l__siunitx_unit_tmp_tl { #1 - #2 }
275   \prop_if_in:NVTF \l__siunitx_unit_parsed_prop
276     \l__siunitx_unit_tmp_tl
277   {
278     \msg_error:nnxx { siunitx } { unit / duplicate-part }
279     { \exp_not:n {#1} } { \token_to_str:N #3 }
280   }
281   {
282     \prop_put:NVn \l__siunitx_unit_parsed_prop
283       \l__siunitx_unit_tmp_tl {#4}
284   }
285 }

```

(End definition for `_siunitx_unit_parse_add:nnnn`.)

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

```

286 \cs_new_protected:Npn \_siunitx_unit_parse_prefix:Nn #1#2
287 {
288   \int_set:Nn \l__siunitx_unit_tmp_int { \l__siunitx_unit_position_int + 1 }
289   \_siunitx_unit_parse_add:nnnn { prefix }
290   { \int_use:N \l__siunitx_unit_tmp_int } {#1} {#2}
291 }
292 \cs_new_protected:Npn \_siunitx_unit_parse_power:nnN #1#2#3
293 {
294   \tl_set:Nx \l__siunitx_unit_tmp_tl
295     { unit- \int_use:N \l__siunitx_unit_position_int }
296   \bool_lazy_or:nnTF
297     {#3}
298     {
299       \prop_if_in_p:NV
300         \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
301     }
302     {
303       \_siunitx_unit_parse_add:nnnn { power }
304       {
305         \int_eval:n
306           { \l__siunitx_unit_position_int \bool_if:NT #3 { + 1 } }
307       }
308       {#1} {#2}
309     }
310   {
311     \msg_error:nnxx { siunitx }
312       { unit / part-before-unit } { power } { \token_to_str:N #1 }
313   }
314 }

```

```

315 \cs_new_protected:Npn \__siunitx_unit_parse_qualifier:nn #1#2
316 {
317   \tl_set:Nx \l__siunitx_unit_tmp_tl
318   { unit- \int_use:N \l__siunitx_unit_position_int }
319   \prop_if_in:NVTF \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
320   {
321     \__siunitx_unit_parse_add:nnnn { qualifier }
322     { \int_use:N \l__siunitx_unit_position_int } {#1} {#2}
323   }
324   {
325     \msg_error:nnnn { siunitx }
326     { unit / part-before-unit } { qualifier } { \token_to_str:N #1 }
327   }
328 }

```

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

```

329 \cs_new_protected:Npn \__siunitx_unit_parse_special:n #1
330 {
331   \__siunitx_unit_parse_add:nnnn { special }
332   { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
333   {#1} {#1}
334 }

```

(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.

```

335 \cs_new_protected:Npn \__siunitx_unit_parse_unit:Nn #1#2
336 {
337   \int_incr:N \l__siunitx_unit_position_int
338   \__siunitx_unit_parse_add:nnnn { unit }
339   { \int_use:N \l__siunitx_unit_position_int }
340   {#1} {#2}
341   \bool_if:NT \l__siunitx_unit_per_bool
342   {
343     \__siunitx_unit_parse_add:nnnn { per }
344     { \int_use:N \l__siunitx_unit_position_int }
345     { \per } { true }
346   }
347 }

```

(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.

```

348 \cs_new_protected:Npn \__siunitx_unit_parse_per:
349 {
350   \bool_if:NTF \l__siunitx_unit_sticky_per_bool

```

```

351     {
352       \bool_set_true:N \l__siunitx_unit_per_bool
353       \cs_set_protected:Npn \per
354         { \msg_error:nn { siunitx } { unit / duplicate-sticky-per } }
355     }
356     {
357       \__siunitx_unit_parse_add:nnnn
358       { per } { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
359       { \per } { true }
360     }
361   }

```

(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.

```

362 \cs_new_protected:Npn \__siunitx_unit_parse_finalise:n #1
363 {
364   \tl_set:Nx \l__siunitx_unit_tmp_tl { per- #1 }
365   \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
366   {
367     \prop_remove:NV \l__siunitx_unit_parsed_prop
368     \l__siunitx_unit_tmp_tl
369     \tl_set:Nx \l__siunitx_unit_tmp_tl { power- #1 }
370     \prop_get:NVNTF
371       \l__siunitx_unit_parsed_prop
372       \l__siunitx_unit_tmp_tl
373       \l__siunitx_unit_part_tl
374       {
375         \tl_set:Nx \l__siunitx_unit_part_tl
376           { \fp_eval:n { \l__siunitx_unit_part_tl * -1 } }
377         \fp_compare:nNnTF \l__siunitx_unit_part_tl = 1
378           {
379             \prop_remove:NV \l__siunitx_unit_parsed_prop
380             \l__siunitx_unit_tmp_tl
381           }
382           {
383             \prop_put:NVV \l__siunitx_unit_parsed_prop
384             \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
385           }
386       }
387   }
388   \prop_put:NVN \l__siunitx_unit_parsed_prop
389     \l__siunitx_unit_tmp_tl { -1 }
390 }
391 }
392 }

```

(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.


```

393 \cs_new_protected:Npn \__siunitx_unit_parse_finalise:
394 {
395   \clist_map_inline:nn { per , power , prefix }
396   {
397     \tl_set:Nx \l__siunitx_unit_tmp_tl
398     { ##1 - \int_eval:n { \l__siunitx_unit_position_int + 1 } }
399     \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
400     { \msg_error:nnn { siunitx } { unit / dangling-part } { ##1 } }
401   }
402 }

```

(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_forbid_literal_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
403 \keys_define:nn { siunitx }
404 {
405   bracket-unit-denominator .bool_set:N =
406   \l__siunitx_unit_denominator_bracket_bool ,
407   forbid-literal-units .bool_set:N =
408   \l__siunitx_unit_forbid_literal_bool ,
409   fraction-command .tl_set:N =
410   \l__siunitx_unit_fraction_function_tl ,
411   parse-units .bool_set:N =
412   \l__siunitx_unit_parse_bool ,
413   per-mode .choice: ,
414   per-mode / fraction .code:n =
415   {
416     \bool_set_false:N \l__siunitx_unit_autofrac_bool
417     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
418     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
419     \bool_set_true:N \l__siunitx_unit_two_part_bool
420   } ,
421   per-mode / power .code:n =
422   {
423     \bool_set_false:N \l__siunitx_unit_autofrac_bool
424     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
425     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
426     \bool_set_false:N \l__siunitx_unit_two_part_bool
427   } ,
428   per-mode / power-positive-first .code:n =
429   {
430     \bool_set_false:N \l__siunitx_unit_autofrac_bool
431     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
432     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
433     \bool_set_true:N \l__siunitx_unit_two_part_bool
434   } ,
435   per-mode / repeated-symbol .code:n =
436   {
437     \bool_set_false:N \l__siunitx_unit_autofrac_bool
438     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
439     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
440     \bool_set_false:N \l__siunitx_unit_two_part_bool

```

```

441     } ,
442     per-mode / symbol .code:n =
443     {
444         \bool_set_false:N \l__siunitx_unit_autofrac_bool
445         \bool_set_true:N \l__siunitx_unit_per_symbol_bool
446         \bool_set_true:N \l__siunitx_unit_powers_positive_bool
447         \bool_set_true:N \l__siunitx_unit_two_part_bool
448     } ,
449     per-mode / symbol-or-fraction .code:n =
450     {
451         \bool_set_true:N \l__siunitx_unit_autofrac_bool
452         \bool_set_true:N \l__siunitx_unit_per_symbol_bool
453         \bool_set_true:N \l__siunitx_unit_powers_positive_bool
454         \bool_set_true:N \l__siunitx_unit_two_part_bool
455     } ,
456     per-symbol .tl_set:N =
457         \l__siunitx_unit_per_symbol_tl ,
458     qualifier-mode .choices:nn =
459         { bracket , combine , phrase , subscript }
460         { \tl_set_eq:NN \l__siunitx_unit_qualifier_mode_tl \l_keys_choice_tl } ,
461     qualifier-phrase .tl_set:N =
462         \l__siunitx_unit_qualifier_phrase_tl ,
463     unit-close-bracket .tl_set:N =
464         \l__siunitx_unit_bracket_close_tl ,
465     unit-open-bracket .tl_set:N =
466         \l__siunitx_unit_bracket_open_tl
467 }

```

(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.

```
468 \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.

```
469 \bool_new:N \l__siunitx_unit_font_bool
```

(End definition for `\l__siunitx_unit_font_bool`.)

`\l__siunitx_unit_autofrac_bool` `\l__siunitx_unit_powers_positive_bool` `\l__siunitx_unit_per_symbol_bool` `\l__siunitx_unit_two_part_bool` Dealing with the various ways that reciprocal (`\per`) can be handled requires a few different switches.

```

470 \bool_new:N \l__siunitx_unit_autofrac_bool
471 \bool_new:N \l__siunitx_unit_per_symbol_bool
472 \bool_new:N \l__siunitx_unit_powers_positive_bool
473 \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).

```
474 \bool_new:N \l__siunitx_unit_numerator_bool
```

(End definition for `\l__siunitx_unit_numerator_bool`.)

$\backslash l_siunitx_unit_qualifier_mode_tl$	<p>For storing the text of options which are best handled by picking function names.</p> <pre> 475 \tl_new:N \l__siunitx_unit_qualifier_mode_tl (End definition for \l__siunitx_unit_qualifier_mode_tl.) </pre>
$\backslash l_siunitx_unit_prefix_power_bool$	<p>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).</p> <pre> 476 \bool_new:N \l__siunitx_unit_prefix_power_bool (End definition for \l__siunitx_unit_prefix_power_bool.) </pre>
$\backslash l_siunitx_unit_prefix_fp$	<p>When converting prefixes to powers, the calculations are done as an fp.</p> <pre> 477 \fp_new:N \l__siunitx_unit_prefix_fp (End definition for \l__siunitx_unit_prefix_fp.) </pre>
$\backslash l_siunitx_unit_current_tl$ $\backslash l_siunitx_unit_part_tl$	<p>Building up the (partial) formatted unit requires some token list storage. Each part of the unit combination that is recovered also has to be placed in a token list: this is a dedicated one to leave the scratch variables available.</p> <pre> 478 \tl_new:N \l__siunitx_unit_current_tl 479 \tl_new:N \l__siunitx_unit_part_tl (End definition for \l__siunitx_unit_current_tl and \l__siunitx_unit_part_tl.) </pre>
$\backslash l_siunitx_unit_denominator_tl$	<p>For fraction-like units, space is needed for the denominator as well as the numerator (which is handled using $\backslash l_siunitx_unit_formatted_tl$).</p> <pre> 480 \tl_new:N \l__siunitx_unit_denominator_tl (End definition for \l__siunitx_unit_denominator_tl.) </pre>
$\backslash l_siunitx_unit_total_int$	<p>The formatting routine needs to know both the total number of units and the current unit. Thus an int is required in addition to $\backslash l_siunitx_unit_position_int$.</p> <pre> 481 \int_new:N \l__siunitx_unit_total_int (End definition for \l__siunitx_unit_total_int.) </pre>
$\backslash_siunitx_unit_format_parsed:$ $\backslash_siunitx_unit_format_parsed_aux:n$	<p>The main formatting routine is essentially a loop over each position, reading the various parts of the unit to build up complete unit combination.</p> <pre> 482 \cs_new_protected:Npn __siunitx_unit_format_parsed: 483 { 484 \int_set_eq:NN \l__siunitx_unit_total_int \l__siunitx_unit_position_int 485 \tl_clear:N \l__siunitx_unit_denominator_tl 486 \tl_clear:N \l__siunitx_unit_formatted_tl 487 \fp_zero:N \l__siunitx_unit_prefix_fp 488 \int_zero:N \l__siunitx_unit_position_int 489 \int_do_while:nNnn 490 \l__siunitx_unit_position_int < \l__siunitx_unit_total_int 491 { 492 \bool_set_false:N \l__siunitx_unit_bracket_bool 493 \tl_clear:N \l__siunitx_unit_current_tl 494 \bool_set_false:N \l__siunitx_unit_font_bool 495 \bool_set_true:N \l__siunitx_unit_numerator_bool 496 \int_incr:N \l__siunitx_unit_position_int 497 \clist_map_inline:nn { prefix , unit , qualifier , power , special } </pre>

```

498         { \_siunitx_unit_format_parsed_aux:n {##1} }
499         \_siunitx_unit_format_output:
500     }
501     \_siunitx_unit_format_finalise:
502 }
503 \cs_new_protected:Npn \_siunitx_unit_format_parsed_aux:n #1
504 {
505     \tl_set:Nx \l__siunitx_unit_tmp_tl
506     { #1 - \int_use:N \l__siunitx_unit_position_int }
507     \prop_get:NVNT \l__siunitx_unit_parsed_prop
508     \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
509     { \use:c { \_siunitx_unit_format_ #1 : } }
510 }

```

(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.

```

511 \cs_new:Npn \_siunitx_unit_format_bracket:N #1
512 {
513     \bool_if:NTF \l__siunitx_unit_bracket_bool
514     {
515         \exp_not:V \l__siunitx_unit_bracket_open_tl
516         \exp_not:V #1
517         \exp_not:V \l__siunitx_unit_bracket_close_tl
518     }
519     { \exp_not:V #1 }
520 }

```

(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.

```

521 \cs_new_protected:Npn \_siunitx_unit_format_power:
522 {
523     \_siunitx_unit_format_font:
524     \exp_after:wN \_siunitx_unit_format_power_aux:wTF
525     \l__siunitx_unit_part_tl - \q_stop
526     { \_siunitx_unit_format_power_negative: }
527     { \_siunitx_unit_format_power_positive: }
528 }
529 \cs_new:Npn \_siunitx_unit_format_power_aux:wTF #1 - #2 \q_stop
530 { \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 $\neq 1$).

```

531 \cs_new_protected:Npn \_siunitx_unit_format_power_positive:
532 { \_siunitx_unit_format_power_superscript: }

```

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

```

533 \cs_new_protected:Npn \__siunitx_unit_format_power_negative:
534 {
535   \bool_set_false:N \l__siunitx_unit_numerator_bool
536   \bool_if:NTF \l__siunitx_unit_powers_positive_bool
537   {
538     \tl_set:Nx \l__siunitx_unit_part_tl
539     {
540       \exp_after:wN \__siunitx_unit_format_power_negative_aux:w
541       \l__siunitx_unit_part_tl \q_stop
542     }
543     \str_if_eq:VnF \l__siunitx_unit_part_tl { 1 }
544     { \__siunitx_unit_format_power_superscript: }
545   }
546   { \__siunitx_unit_format_power_superscript: }
547 }
548 \cs_new:Npn \__siunitx_unit_format_power_negative_aux:w - #1 \q_stop
549 { \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.

```

550 \cs_new_protected:Npn \__siunitx_unit_format_power_superscript:
551 {
552   \tl_set:Nx \l__siunitx_unit_current_tl
553   {
554     \__siunitx_unit_format_bracket:N \l__siunitx_unit_current_tl
555     ~ { \exp_not:V \l__siunitx_unit_part_tl }
556   }
557   \bool_set_false:N \l__siunitx_unit_bracket_bool
558 }

```

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

<pre> __siunitx_unit_format_prefix: __siunitx_unit_format_prefix_power: __siunitx_unit_format_prefix_symbol: </pre>	<p>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.</p>
--	---

```

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

```

```

576         { \l__siunitx_unit_tmp_tl * \l__siunitx_unit_part_tl }
577     }
578     { \__siunitx_unit_format_prefix_symbol: }
579 }
580 \cs_new_protected:Npn \__siunitx_unit_format_prefix_symbol:
581 { \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. Notice that when the qualifier is just treated as “text”, the auxiliary is actually a no-op.

```

582 \cs_new_protected:Npn \__siunitx_unit_format_qualifier:
583 {
584     \use:c
585     {
586         __siunitx_unit_format_qualifier_
587         \l__siunitx_unit_qualifier_mode_tl :
588     }
589     \tl_put_right:NV \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
590 }
591 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_bracket:
592 {
593     \__siunitx_unit_format_font:
594     \tl_set:Nx \l__siunitx_unit_part_tl
595     {
596         \exp_not:V \l__siunitx_unit_bracket_open_tl
597         \exp_not:V \l__siunitx_unit_font_tl
598         { \exp_not:V \l__siunitx_unit_part_tl }
599         \exp_not:V \l__siunitx_unit_bracket_close_tl
600     }
601 }
602 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_combine: { }
603 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_phrase:
604 {
605     \__siunitx_unit_format_font:
606     \tl_set:Nx \l__siunitx_unit_part_tl
607     {
608         \exp_not:V \l__siunitx_unit_qualifier_phrase_tl
609         \exp_not:V \l__siunitx_unit_font_tl
610         { \exp_not:V \l__siunitx_unit_part_tl }
611     }
612 }
613 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_subscript:
614 {
615     \__siunitx_unit_format_font:
616     \tl_set:Nx \l__siunitx_unit_part_tl
617     {
618         \c__siunitx_unit_math_subscript_tl
619         {
620             \exp_not:V \l__siunitx_unit_font_tl
621             { \exp_not:V \l__siunitx_unit_part_tl }
622         }
623     }

```

```

623     }
624 }

```

(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.

```

625 \cs_new_protected:Npn \_siunitx_unit_format_special:
626 {
627   \tl_set:Nx \l__siunitx_unit_current_tl
628   {
629     \exp_not:V \l__siunitx_unit_part_tl
630     { \exp_not:V \l__siunitx_unit_current_tl }
631   }
632 }

```

(End definition for `_siunitx_unit_format_special`.)

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

```

633 \cs_new_protected:Npn \_siunitx_unit_format_unit:
634 {
635   \tl_put_right:NV
636   \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
637 }

```

(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

```

638 \cs_new_protected:Npn \_siunitx_unit_format_output:
639 {
640   \_siunitx_unit_format_font:
641   \bool_set_false:N \l__siunitx_unit_bracket_bool
642   \use:c
643   {
644     \_siunitx_unit_format_output_
645     \bool_if:NTF \l__siunitx_unit_numerator_bool
646     { aux: }
647     { denominator: }
648   }
649 }
650 \cs_new_protected:Npn \_siunitx_unit_format_output_aux:
651 {
652   \_siunitx_unit_format_output_aux:nV { formatted }
653   \l__siunitx_unit_product_tl
654 }

```

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.

```

655 \cs_new_protected:Npn \__siunitx_unit_format_output_denominator:
656 {
657   \bool_if:NTF \l__siunitx_unit_two_part_bool
658   {
659     \bool_lazy_and:nnT
660     { \l__siunitx_unit_denominator_bracket_bool }
661     { ! \tl_if_empty_p:N \l__siunitx_unit_denominator_tl }
662     { \bool_set_true:N \l__siunitx_unit_bracket_bool }
663     \__siunitx_unit_format_output_aux:nV { denominator }
664     \l__siunitx_unit_product_tl
665   }
666   {
667     \bool_lazy_and:nnT
668     { \l__siunitx_unit_per_symbol_bool }
669     { \tl_if_empty_p:N \l__siunitx_unit_formatted_tl }
670     { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
671     \__siunitx_unit_format_output_aux:nv { formatted }
672     {
673       \l__siunitx_unit_
674       \bool_if:NTF \l__siunitx_unit_per_symbol_bool
675       { per_symbol }
676       { product }
677       _tl
678     }
679   }
680 }
681 \cs_new_protected:Npn \__siunitx_unit_format_output_aux:nn #1#2
682 {
683   \tl_set:cx { \l__siunitx_unit_ #1 _tl }
684   {
685     \exp_not:v { \l__siunitx_unit_ #1 _tl }
686     \tl_if_empty:cF { \l__siunitx_unit_ #1 _tl }
687     { \exp_not:n {#2} }
688     \exp_not:V \l__siunitx_unit_current_tl
689   }
690 }
691 \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.

```

692 \cs_new_protected:Npn \__siunitx_unit_format_font:
693 {
694   \bool_if:NF \l__siunitx_unit_font_bool
695   {
696     \tl_set:Nx \l__siunitx_unit_current_tl
697     {
698       \exp_not:V \l__siunitx_unit_font_tl
699       { \exp_not:V \l__siunitx_unit_current_tl }
700     }

```



```

701         \bool_set_true:N \l__siunitx_unit_font_bool
702     }
703 }

```

(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:
704 \cs_new_protected:Npn \__siunitx_unit_format_finalise:
705 {
706     \tl_if_empty:NF \l__siunitx_unit_denominator_tl
707     {
708         \bool_if:NTF \l__siunitx_unit_powers_positive_bool
709         { \__siunitx_unit_format_finalise_fractional: }
710         { \__siunitx_unit_format_finalise_power: }
711     }
712 }

```

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.

```

713 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fractional:
714 {
715     \tl_if_empty:NT \l__siunitx_unit_formatted_tl
716     { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
717     \bool_if:NTF \l__siunitx_unit_autofrac_bool
718     { \__siunitx_unit_format_finalise_autofrac: }
719     {
720         \bool_if:NTF \l__siunitx_unit_per_symbol_bool
721         { \__siunitx_unit_format_finalise_symbol: }
722         { \__siunitx_unit_format_finalise_fraction: }
723     }
724 }

```

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.

```

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

```

```

739     }
740 }

```

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

```

741 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fraction:
742 {
743   \tl_set:Nx \l__siunitx_unit_formatted_tl
744   {
745     \exp_not:V \l__siunitx_unit_fraction_function_tl
746     { \exp_not:V \l__siunitx_unit_formatted_tl }
747     { \exp_not:V \l__siunitx_unit_denominator_tl }
748   }
749 }
750 \cs_new_protected:Npn \__siunitx_unit_format_finalise_symbol:
751 {
752   \tl_set:Nx \l__siunitx_unit_formatted_tl
753   {
754     \exp_not:V \l__siunitx_unit_formatted_tl
755     \exp_not:V \l__siunitx_unit_per_symbol_tl
756     \__siunitx_unit_format_bracket:N \l__siunitx_unit_denominator_tl
757   }
758 }

```

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.

```

759 \cs_new_protected:Npn \__siunitx_unit_format_finalise_power:
760 {
761   \tl_if_empty:NTF \l__siunitx_unit_formatted_tl
762   {
763     \tl_set_eq:NN
764     \l__siunitx_unit_formatted_tl
765     \l__siunitx_unit_denominator_tl
766   }
767   {
768     \tl_set:Nx \l__siunitx_unit_formatted_tl
769     {
770       \exp_not:V \l__siunitx_unit_formatted_tl
771       \exp_not:V \l__siunitx_unit_product_tl
772       \exp_not:V \l__siunitx_unit_denominator_tl
773     }
774   }
775 }

```

(End definition for __siunitx_unit_format_finalise: and others.)

4.8 Non-Latin character support

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

```

776 \bool_lazy_or:nnTF
777 { \sys_if_engine luatex_p: }
778 { \sys_if_engine xetex_p: }
779 {
780   \cs_new:Npn \__siunitx_unit_non_latin:n #1
781     { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }

```

```

782 }
783 {
784   \cs_new:Npn \__siunitx_unit_non_latin:n #1
785   {
786     \exp_last_unbraced:Nf \__siunitx_unit_non_latin:nnnn
787     { \char_codepoint_to_bytes:n {#1} }
788   }
789   \cs_new:Npn \__siunitx_unit_non_latin:nnnn #1#2#3#4
790   {
791     \exp_after:wN \exp_after:wN \exp_after:wN
792     \exp_not:N \char_generate:nn {#1} { 13 }
793     \exp_after:wN \exp_after:wN \exp_after:wN
794     \exp_not:N \char_generate:nn {#2} { 13 }
795   }
796 }

```

(End definition for `__siunitx_unit_non_latin:n` and `__siunitx_unit_non_latin:nnnn`.)

4.9 Pre-defined unit components

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

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

\metre

\meter 797 \siunitx_declare_unit:Nn \kilogram { \kilo \gram }

\mole 798 \siunitx_declare_unit:Nn \metre { m }

\kelvin 799 \siunitx_declare_unit:Nn \meter { \metre }

\candela 800 \siunitx_declare_unit:Nn \mole { mol }

\second 801 \siunitx_declare_unit:Nn \second { s }

\ampere 802 \siunitx_declare_unit:Nn \ampere { A }

803 \siunitx_declare_unit:Nn \kelvin { K }

804 \siunitx_declare_unit:Nn \candela { cd }

(End definition for `\kilogram` and others. These functions are documented on page 92.)

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

805 \siunitx_declare_unit:Nn \gram { g }

(End definition for `\gram`. This function is documented on page 92.)

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

\zepto 806 \siunitx_declare_prefix:Nnn \yocto { -24 } { y }

\atto 807 \siunitx_declare_prefix:Nnn \zepto { -21 } { z }

\femto 808 \siunitx_declare_prefix:Nnn \atto { -18 } { a }

\pico 809 \siunitx_declare_prefix:Nnn \femto { -15 } { f }

\nano 810 \siunitx_declare_prefix:Nnn \pico { -12 } { p }

\micro 811 \siunitx_declare_prefix:Nnn \nano { -9 } { n }

\milli 812 \exp_args:NNnx \siunitx_declare_prefix:Nnn

\centi 813 \micro { -6 } { __siunitx_unit_non_latin:n { "03BC } }

\deci 814 \siunitx_declare_prefix:Nnn \milli { -3 } { m }

815 \siunitx_declare_prefix:Nnn \centi { -2 } { c }

816 \siunitx_declare_prefix:Nnn \deci { -1 } { d }

(End definition for `\yocto` and others. These functions are documented on page 93.)

```

\deca Now the large ones.
\deka 817 \siunitx_declare_prefix:Nnn \deca { 1 } { da }
\hecto 818 \siunitx_declare_prefix:Nnn \deka { 1 } { da }
\kilo 819 \siunitx_declare_prefix:Nnn \hecto { 2 } { h }
\mega 820 \siunitx_declare_prefix:Nnn \kilo { 3 } { k }
\giga 821 \siunitx_declare_prefix:Nnn \mega { 6 } { M }
\tera 822 \siunitx_declare_prefix:Nnn \giga { 9 } { G }
\peta 823 \siunitx_declare_prefix:Nnn \tera { 12 } { T }
\exa 824 \siunitx_declare_prefix:Nnn \peta { 15 } { P }
\zetta 825 \siunitx_declare_prefix:Nnn \exa { 18 } { E }
\yotta 826 \siunitx_declare_prefix:Nnn \zetta { 21 } { Z }
827 \siunitx_declare_prefix:Nnn \yotta { 24 } { Y }

```

(End definition for \deca and others. These functions are documented on page 93.)

```

\becquerel Named derived units: first half of alphabet.
\degreeCelsius 828 \siunitx_declare_unit:Nn \becquerel { Bq }
\coulomb 829 \siunitx_declare_unit:Nn \degreeCelsius
\farad 830 { \ensuremath { { } ^ { \circ } } } \kern -\scriptspace C }
\gray 831 \siunitx_declare_unit:Nn \coulomb { C }
\hertz 832 \siunitx_declare_unit:Nn \farad { F }
\henry 833 \siunitx_declare_unit:Nn \gray { Gy }
\joule 834 \siunitx_declare_unit:Nn \hertz { Hz }
\katal 835 \siunitx_declare_unit:Nn \henry { H }
\lumen 836 \siunitx_declare_unit:Nn \joule { J }
\lux 837 \siunitx_declare_unit:Nn \katal { kat }
838 \siunitx_declare_unit:Nn \lumen { lm }
839 \siunitx_declare_unit:Nn \lux { lx }

```

(End definition for \becquerel and others. These functions are documented on page 93.)

```

\newton Named derived units: second half of alphabet.
\ohm 840 \siunitx_declare_unit:Nn \newton { N }
\pascal 841 \siunitx_declare_unit:Nx \ohm { \_siunitx_unit_non_latin:n { "03A9 } }
\radian 842 \siunitx_declare_unit:Nn \pascal { Pa }
\siemens 843 \siunitx_declare_unit:Nn \radian { rad }
\sievert 844 \siunitx_declare_unit:Nn \siemens { S }
\steradian 845 \siunitx_declare_unit:Nn \sievert { Sv }
\tesla 846 \siunitx_declare_unit:Nn \steradian { sr }
\volt 847 \siunitx_declare_unit:Nn \tesla { T }
\watt 848 \siunitx_declare_unit:Nn \volt { V }
\weber 849 \siunitx_declare_unit:Nn \watt { W }
850 \siunitx_declare_unit:Nn \weber { Wb }

```

(End definition for \newton and others. These functions are documented on page 93.)

```

\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 851 \siunitx_declare_unit:Nn \day { d }
\litre 852 \siunitx_declare_unit:Nn \hectare { ha }
\liter 853 \siunitx_declare_unit:Nn \hour { h }
\minute 854 \siunitx_declare_unit:Nn \litre { L }
\tonne 855 \siunitx_declare_unit:Nn \liter { \litre }
856 \siunitx_declare_unit:Nn \minute { min }
857 \siunitx_declare_unit:Nn \tonne { t }

```

(End definition for `\day` and others. These functions are documented on page 93.)

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

```

858 \siunitx_declare_unit:Nn \arcminute { { } ' }
859 \siunitx_declare_unit:Nn \arcsecond { { } '' }
860 \siunitx_declare_unit:Nn \degree { { } ^ { \circ } }
```

(End definition for `\arcminute`, `\arcsecond`, and `\degree`. These functions are documented on page 94.)

`\astronomicalunit` A few units based on physical measurements exist: these ones are accepted for use with the International System.

```

861 \siunitx_declare_unit:Nn \astronomicalunit { au }
862 \siunitx_declare_unit:Nn \atomicmassunit { u }
863 \siunitx_declare_unit:Nn \dalton { Da }
864 \siunitx_declare_unit:Nn \electronvolt { eV }
```

(End definition for `\astronomicalunit` and others. These functions are documented on page 94.)

`\nuaction` Natural units based on physical constants.

```

865 \siunitx_declare_unit:Nn \nuaction { \ensuremath { \mathit { \hbar } } }
866 \siunitx_declare_unit:Nx \numass
867 {
868   \exp_not:N \ensuremath
869   {
870     \exp_not:N \mathit { m }
871     \c__siunitx_unit_math_subscript_tl { \exp_not:N \mathrm { e } }
872   }
873 }
874 \siunitx_declare_unit:Nx \nuspeed
875 {
876   \exp_not:N \ensuremath
877   { \exp_not:N \mathit { c } \c__siunitx_unit_math_subscript_tl { 0 } }
878 }
879 \siunitx_declare_unit:Nn \nutime
880 { \numass \per \numass \per \nuspeed \squared }
```

(End definition for `\nuaction` and others. These functions are documented on page 94.)

`\auaction` Atomic units based on physical constants.

```

881 \siunitx_declare_unit:Nn \auaction { \ensuremath { \mathit { \hbar } } }
882 \siunitx_declare_unit:Nn \aucharge { \ensuremath { \mathit { e } } }
883 \siunitx_declare_unit:Nx \auenergy
884 {
885   \exp_not:N \ensuremath
886   {
887     \exp_not:N \mathit { E }
888     \c__siunitx_unit_math_subscript_tl { \exp_not:N \mathrm { h } }
889   }
890 }
891 \siunitx_declare_unit:Nx \aulength
892 {
893   \exp_not:N \ensuremath
894   { \exp_not:N \mathit { a } \c__siunitx_unit_math_subscript_tl { 0 } }
895 }
```


4.10 Messages

```

930 \msg_new:nnnn { siunitx } { unit / dangling-part }
931 { Found~#1~part~with~no~unit. }
932 {
933   Each~#1~part~must~be~associated~with~a~unit:~a~#1~part~was~found~
934   but~no~following~unit~was~given.
935 }
936 \msg_new:nnnn { siunitx } { unit / duplicate-part }
937 { Duplicate~#1~part:~#2. }
938 {
939   Each~unit~may~have~only~one~#1:\\
940   the~additional~#1~part~'~#2'~will~be~ignored.
941 }
942 \msg_new:nnnn { siunitx } { unit / duplicate-sticky-per }
943 { Duplicate~\token_to_str:N \per. }
944 {
945   When~the~'sticky-per'~option~is~active,~only~one~
946   \token_to_str:N \per \ may~appear~in~a~unit.
947 }
948 \msg_new:nnnn { siunitx } { unit / literal }
949 { Literal~units~disabled. }
950 {
951   You~gave~the~literal~input~'~#1'~
952   but~literal~unit~output~is~disabled.
953 }
954 \msg_new:nnnn { siunitx } { unit / part-before-unit }
955 { Found~#1~part~before~first~unit:~#2. }
956 {
957   The~#1~part~'~#2'~must~follow~after~a~unit:~
958   it~cannot~appear~before~any~units~and~will~therefore~be~ignored.
959 }

```

4.11 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.

```

960 \keys_set:nn { siunitx }
961 {
962   font-command           = \mathrm ,
963   bracket-unit-denominator = true ,
964   forbid-literal-units    = false ,
965   fraction-command        = \frac ,
966   inter-unit-product      = \, ,
967   parse-units             = true ,
968   per-mode                = power ,
969   per-symbol              = / ,
970   qualifier-mode          = subscript ,
971   qualifier-phrase        = ,
972   sticky-per              = false ,
973   unit-close-bracket      = ) , % (
974   unit-open-bracket       = ( , % )
975 }
976 \end{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 VIII

siunitx-abbreviations – Abbreviations

<code>\A</code>	Abbreviations for currents.
<code>\pA</code>	
<code>\nA</code>	
<code>\uA</code>	
<code>\mA</code>	
<code>\kA</code>	

<code>\fg</code>	Abbreviations for masses.
<code>\pg</code>	
<code>\ng</code>	
<code>\ug</code>	
<code>\mg</code>	
<code>\g</code>	
<code>\kg</code>	
<code>\amu</code>	

<code>\K</code>	Abbreviations for temperature.
-----------------	--------------------------------

<code>\m</code>	Abbreviations for lengths.
<code>\pm</code>	
<code>\nm</code>	
<code>\um</code>	
<code>\mm</code>	
<code>\cm</code>	
<code>\dm</code>	
<code>\km</code>	

<code>\s</code>	Abbreviations for times.
<code>\as</code>	
<code>\fs</code>	
<code>\ps</code>	
<code>\ns</code>	
<code>\us</code>	
<code>\ms</code>	

<code>\Hz</code>	Abbreviations for frequencies.
<code>\mHz</code>	
<code>\kHz</code>	
<code>\MHz</code>	
<code>\GHz</code>	
<code>\THz</code>	

<hr/>	
<code>\mol</code>	Abbreviations for moles.
<code>\fmol</code>	
<code>\pmol</code>	
<code>\nmol</code>	
<code>\umol</code>	
<code>\mmol</code>	
<code>\kmol</code>	
<hr/>	
<code>\V</code>	Abbreviations for potentials.
<code>\pV</code>	
<code>\nV</code>	
<code>\uV</code>	
<code>\mV</code>	
<code>\kV</code>	
<hr/>	
<code>\hl</code>	Abbreviations for volumes.
<code>\l</code>	
<code>\ml</code>	
<code>\ul</code>	
<code>\hL</code>	
<code>\L</code>	
<code>\mL</code>	
<code>\uL</code>	
<hr/>	
<code>\W</code>	Abbreviations for powers.
<code>\uW</code>	
<code>\mW</code>	
<code>\kW</code>	
<code>\MW</code>	
<code>\GW</code>	
<hr/>	
<code>\kJ</code>	Abbreviations for energies.
<code>\J</code>	
<code>\mJ</code>	
<code>\uJ</code>	
<code>\eV</code>	
<code>\meV</code>	
<code>\keV</code>	
<code>\MeV</code>	
<code>\GeV</code>	
<code>\TeV</code>	
<hr/>	
<code>\N</code>	Abbreviations for forces.
<code>\mN</code>	
<code>\kN</code>	
<code>\MN</code>	
<hr/>	
<code>\Pa</code>	Abbreviations for pressures.
<code>\kPa</code>	
<code>\MPa</code>	
<code>\GPa</code>	

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

`\F` Abbreviations for capacitance.
`\fF`
`\pF`

`\dB` Abbreviation for decibel.

`\kWh` Abbreviation for kilowatt-hours.

1 siunitx-abbreviation implementation

Start the DocStrip guards.

```
1 \<{*package}
```

The abbreviation file contains a number of short (mainly two or three letter) versions of the usual long names. They are divided up into related groups, mainly to avoid an overly long list in one place.

`\A` Currents.

```
\pA 2 \DeclareSIUnit \A { \ampere }
\nA 3 \DeclareSIUnit \pA { \pico \ampere }
\uA 4 \DeclareSIUnit \nA { \nano \ampere }
\mA 5 \DeclareSIUnit \uA { \micro \ampere }
\kA 6 \DeclareSIUnit \mA { \milli \ampere }
7 \DeclareSIUnit \kA { \kilo \ampere }
```

(End definition for \A and others. These functions are documented on page 126.)

`\Hz` Then frequencies.

```
\mHz 8 \DeclareSIUnit \Hz { \hertz }
\kHz 9 \DeclareSIUnit \mHz { \milli \hertz }
\MHz 10 \DeclareSIUnit \kHz { \kilo \hertz }
\GHz 11 \DeclareSIUnit \MHz { \mega \hertz }
\THz 12 \DeclareSIUnit \GHz { \giga \hertz }
13 \DeclareSIUnit \THz { \tera \hertz }
```

(End definition for \Hz and others. These functions are documented on page 126.)

`\mol` Amounts of substance (moles).

```
\fmol 14 \DeclareSIUnit \mol { \mole }
\pmol 15 \DeclareSIUnit \fmol { \femto \mole }
\nmol 16 \DeclareSIUnit \pmol { \pico \mole }
\umol 17 \DeclareSIUnit \nmol { \nano \mole }
\mmol 18 \DeclareSIUnit \umol { \micro \mole }
\kmol 19 \DeclareSIUnit \mmol { \milli \mole }
20 \DeclareSIUnit \kmol { \kilo \mole }
```

(End definition for \mol and others. These functions are documented on page 127.)

\V Potentials.

```

\pV 21 \DeclareSIUnit \V { \volt }
\nV 22 \DeclareSIUnit \pV { \pico \volt }
\uV 23 \DeclareSIUnit \nV { \nano \volt }
\mV 24 \DeclareSIUnit \uV { \micro \volt }
\kV 25 \DeclareSIUnit \mV { \milli \volt }
      26 \DeclareSIUnit \kV { \kilo \volt }

```

(End definition for \V and others. These functions are documented on page 127.)

\hL Volumes.

```

\l 27 \DeclareSIUnit \hL { \hecto \litre }
\mL 28 \DeclareSIUnit \l { \litre }
\uL 29 \DeclareSIUnit \mL { \milli \litre }
\hL 30 \DeclareSIUnit \uL { \micro \litre }
\L 31 \DeclareSIUnit \hL { \hecto \liter }
\mL 32 \DeclareSIUnit \L { \liter }
\uL 33 \DeclareSIUnit \mL { \milli \liter }
      34 \DeclareSIUnit \uL { \micro \liter }

```

(End definition for \hL and others. These functions are documented on page 127.)

\fg Masses.

```

\pg 35 \DeclareSIUnit \fg { \femto \gram }
\ng 36 \DeclareSIUnit \pg { \pico \gram }
\ug 37 \DeclareSIUnit \ng { \nano \gram }
\mg 38 \DeclareSIUnit \ug { \micro \gram }
\g 39 \DeclareSIUnit \mg { \milli \gram }
\kg 40 \DeclareSIUnit \g { \gram }
\amu 41 \DeclareSIUnit \kg { \kilo \gram }
      42 \DeclareSIUnit \amu { \atomicmassunit }

```

(End definition for \fg and others. These functions are documented on page 126.)

\W Energies and powers

```

\uW 43 \DeclareSIUnit \W { \watt }
\mW 44 \DeclareSIUnit \uW { \micro \watt }
\kW 45 \DeclareSIUnit \mW { \milli \watt }
\MW 46 \DeclareSIUnit \kW { \kilo \watt }
\GW 47 \DeclareSIUnit \MW { \mega \watt }
\kJ 48 \DeclareSIUnit \GW { \giga \watt }
\J 49 \DeclareSIUnit \J { \joule }
\mJ 50 \DeclareSIUnit \uJ { \micro \joule }
\uJ 51 \DeclareSIUnit \mJ { \milli \joule }
\kJ 52 \DeclareSIUnit \kJ { \kilo \joule }
\eV 53 \DeclareSIUnit \eV { \electronvolt }
\meV 54 \DeclareSIUnit \meV { \milli \electronvolt }
\keV 55 \DeclareSIUnit \keV { \kilo \electronvolt }
\MeV 56 \DeclareSIUnit \MeV { \mega \electronvolt }
\GeV 57 \DeclareSIUnit \GeV { \giga \electronvolt }
\TeV 58 \DeclareSIUnit \TeV { \tera \electronvolt }
\kWh 59 \DeclareSIUnit [ inter-unit-product = ] \kWh { \kilo \watt \hour }

```

(End definition for \W and others. These functions are documented on page 127.)

\m Lengths.

```
\pm 60 \DeclareSIUnit \m { \metre }  
\nm 61 \DeclareSIUnit \pm { \pico \metre }  
\um 62 \DeclareSIUnit \nm { \nano \metre }  
\mm 63 \DeclareSIUnit \um { \micro \metre }  
\cm 64 \DeclareSIUnit \mm { \milli \metre }  
\dm 65 \DeclareSIUnit \cm { \centi \metre }  
\km 66 \DeclareSIUnit \dm { \deci \metre }  
67 \DeclareSIUnit \km { \kilo \metre }
```

(End definition for \m and others. These functions are documented on page 126.)

\K Temperatures.

```
68 \DeclareSIUnit \K { \kelvin }
```

(End definition for \K. This function is documented on page 126.)

\dB

```
69 \DeclareSIUnit \dB { \deci \bel }
```

(End definition for \dB. This function is documented on page 128.)

\F Capacitance.

```
\fF 70 \DeclareSIUnit \F { \farad }  
\pF 71 \DeclareSIUnit \fF { \femto \farad }  
72 \DeclareSIUnit \pF { \pico \farad }
```

(End definition for \F, \fF, and \pF. These functions are documented on page 128.)

\N Forces.

```
\mN 73 \DeclareSIUnit \N { \newton }  
\kN 74 \DeclareSIUnit \mN { \milli \newton }  
\MN 75 \DeclareSIUnit \kN { \kilo \newton }  
76 \DeclareSIUnit \MN { \mega \newton }
```

(End definition for \N and others. These functions are documented on page 127.)

\Pa Pressures.

```
\kPa 77 \DeclareSIUnit \Pa { \pascal }  
\MPa 78 \DeclareSIUnit \kPa { \kilo \pascal }  
\GPa 79 \DeclareSIUnit \MPa { \mega \pascal }  
80 \DeclareSIUnit \GPa { \giga \pascal }
```

(End definition for \Pa and others. These functions are documented on page 127.)

\mohm Resistances.

```
\kohm 81 \DeclareSIUnit \mohm { \milli \ohm }  
\Mohm 82 \DeclareSIUnit \kohm { \kilo \ohm }  
83 \DeclareSIUnit \Mohm { \mega \ohm }
```

(End definition for \mohm, \kohm, and \Mohm. These functions are documented on page 128.)

```

\s Finally, times.
\as 84 \DeclareSIUnit \s { \second }
\fs 85 \DeclareSIUnit \as { \atto \second }
\ps 86 \DeclareSIUnit \fs { \femto \second }
\ns 87 \DeclareSIUnit \ps { \pico \second }
\us 88 \DeclareSIUnit \ns { \nano \second }
\ms 89 \DeclareSIUnit \us { \micro \second }
90 \DeclareSIUnit \ms { \milli \second }

(End definition for \s and others. These functions are documented on page 126.)
91 \end{package}

```

Part IX

siunitx-emulation – Emulation

1 siunitx-emulation implementation

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.

```
1 <@@=siunitx>
```

1.1 Version 2

Start the DocStrip guards.

```
2 <*package>
   Some messages.
3 \msg_new:nnn { siunitx } { option-deprecated }
4 {
5   Option~"#1"~has~been~deprecated~in~this~release.\\ \\
6   Use~"#2"~as~a~replacement.
7 }
8 \msg_new:nnn { siunitx } { option-removed }
9 { Option~"#1"~has~been~removed~in~this~release. }
```

```
\_siunitx_option_deprecated:nn
\_siunitx_option_deprecated:nnn
\_siunitx_option_deprecated:nnV
```

Abstract out a simple wrapper.

```
10 \cs_new_protected:Npn \_siunitx_option_deprecated:nn #1#2
11 {
12   \msg_info:nnnn { siunitx } { option-deprecated } {#1} {#2}
13   \keys_set:nn { siunitx } {#2}
14 }
15 \cs_new_protected:Npn \_siunitx_option_deprecated:nnn #1#2#3
16 {
17   \msg_info:nnnn { siunitx } { option-deprecated } {#1} {#2}
18   \keys_set:nn { siunitx } { #2 = #3 }
19 }
20 \cs_generate_variant:Nn \_siunitx_option_deprecated:nnn { nnV }
```

(End definition for `_siunitx_option_deprecated:nn` and `_siunitx_option_deprecated:nnn`.)

1.2 Document commands

`\si` A straight copy of `\unit`.

```
21 \NewDocumentCommand \si { 0 { } m }
22 {
23   \mode_leave_vertical:
24   \group_begin:
25     \keys_set:nn { siunitx } {#1}
26     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
27     \siunitx_print:nV { unit } \l__siunitx_tmp_tl
28   \group_end:
29 }
```

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

`\SI` Almost the same as `\qty`, but with the addition pre-unit.

```

30 \NewDocumentCommand \SI { O { } m o m }
31 {
32   \mode_leave_vertical:
33   \group_begin:
34     \keys_set:nn { siunitx } {#1}
35     \IfNoValueF {#3}
36     {
37       \siunitx_unit_format:nN {#3} \l__siunitx_tmp_tl
38       \siunitx_print:nV { unit } \l__siunitx_tmp_tl
39       \nobreak % TEMP
40     }
41     \siunitx_number_format:nN {#2} \l__siunitx_tmp_tl
42     \siunitx_print:nV { number } \l__siunitx_tmp_tl
43     \, \nobreak % TEMP
44     \siunitx_unit_format:nN {#4} \l__siunitx_tmp_tl
45     \siunitx_print:nV { unit } \l__siunitx_tmp_tl
46   \group_end:
47 }

```

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

1.2.1 Number options

For the basic emulation, just set up some information.

```

48 \keys_define:nn { siunitx }
49 {
50   input-protect-tokens .code:n =
51   {
52     \msg_warning:nn { siunitx } { option-removed }
53     { input-protect-tokens }
54   }
55 }

```

Options for number processing: largely removals.

```

56 \keys_define:nn { siunitx }
57 {
58   add-decimal-zero .code:n =
59   {
60     \__siunitx_option_deprecated:nn
61     { add-decimal-zero }
62     { minimum-decimal-digits~=-1 }
63   } ,
64   add-integer-zero .code:n =
65   {
66     \msg_warning:nnn { siunitx } { option-removed }
67     { add-integer-zero }
68   } ,
69   explicit-sign .code:n =
70   {
71     \str_if_eq:nnTF {#1} { + }
72     {
73       \__siunitx_option_deprecated:nn
74       { explicit-sign }

```



```

75         { print-implicit-plus~~true }
76     }
77     {
78         \msg_warning:nnn { siunitx } { option-removed }
79         { explicit-sign }
80     }
81 },
82 retain-explicit-plus .code:n =
83 {
84     \__siunitx_option_deprecated:nnV
85     { retain-explicit-plus }
86     { track-explicit-plus }
87     \l_keys_value_tl
88 },
89 retain-unity-mantissa .code:n =
90 {
91     \__siunitx_option_deprecated:nnV
92     { retain-unity-mantissa }
93     { print-unity-mantissa }
94     \l_keys_value_tl
95 },
96 retain-zero-exponent .code:n =
97 {
98     \__siunitx_option_deprecated:nnV
99     { retain-zero-exponent }
100    { print-zero-exponent }
101    \l_keys_value_tl
102 },
103 omit-uncertainty .code:n =
104 {
105     \__siunitx_option_deprecated:nnV
106     { omit-uncertainty }
107     { drop-uncertainty }
108     \l_keys_value_tl
109 },
110 scientific-notation .choice: ,
111 scientific-notation / engineering .code:n =
112 {
113     \__siunitx_option_deprecated:nn
114     { scientific-notation~~engineering }
115     { exponent-mode~~engineering }
116 },
117 scientific-notation / fixed .code:n =
118 {
119     \__siunitx_option_deprecated:nn
120     { scientific-notation~~fixed }
121     { exponent-mode~~fixed }
122 },
123 scientific-notation / false .code:n =
124 {
125     \__siunitx_option_deprecated:nn
126     { scientific-notation~~false }
127     { exponent-mode~~none }
128 },

```

```

129 scientific-notation / true .code:n =
130 {
131   \__siunitx_option_deprecated:nn
132     { scientific-notation~~true }
133     { exponent-mode~~scientific }
134   } ,
135 zero-decimal-to-integer .code:n =
136 {
137   \__siunitx_option_deprecated:nn
138     { zero-decimal-to-integer }
139     { drop-zero-decimal }
140   }
141 }

```

1.2.2 Unit options

```

142 \keys_define:nn { siunitx }
143 {
144   literal-superscript-as-power .code:n =
145   {
146     \msg_warning:nnn { siunitx } { option-removed }
147     { literal-superscript-as-power }
148   } ,
149   per-mode / reciprocal .code:n =
150   {
151     \__siunitx_option_deprecated:nn
152       { per-mode~~reciprocal }
153       { per-mode~~power }
154   } ,
155   per-mode / reciprocal-positive-first .code:n =
156   {
157     \__siunitx_option_deprecated:nn
158       { per-mode~~reciprocal-positive-first }
159       { per-mode~~power-positive-first }
160   } ,
161   power-font .code:n =
162   {
163     \msg_warning:nnn { siunitx } { option-removed }
164     { power-font }
165   } ,
166   qualifier-mode / brackets .code:n =
167   {
168     \__siunitx_option_deprecated:nn
169       { qualifier-mode~~brackets }
170       { qualifier-mode~~bracket }
171   } ,
172   qualifier-mode / space .meta:n =
173   {
174     \msg_info:nnnn { siunitx } { option-deprecated }
175     { qualifier-mode~~space }
176     { qualifier-mode~~phrase"~plus~"qualifier-phrase=\ }
177     \keys_set:nn
178       { siunitx }
179       { qualifier-mode = phrase, qualifier-phrase = \ }

```

```

180     } ,
181     qualifier-mode / text .meta:n =
182     {
183         \__siunitx_option_deprecated:nn
184         { qualifier-mode~~text }
185         { qualifier-mode~~combine }
186     }
187 }

```

1.2.3 Table options

All straight-forward emulation.

```

188 \keys_define:nn { siunitx }
189 {
190     table-align-text-post .code:n =
191     {
192         \__siunitx_option_deprecated:nnV
193         { table-align-text-post }
194         { table-align-text-after }
195         \l_keys_value_tl
196     } ,
197     table-align-text-pre .code:n =
198     {
199         \__siunitx_option_deprecated:nnV
200         { table-align-text-pre }
201         { table-align-text-before }
202         \l_keys_value_tl
203     } ,
204     table-number-alignment / center-decimal-marker .code:n =
205     {
206         \msg_info:nnnn { siunitx } { option-deprecated }
207         { table-number-alignment~~center-decimal-marker }
208         { table-alignment-mode~~marker }
209         \keys_set:nn
210         { siunitx }
211         { table-alignment-mode = marker }
212     } ,
213     table-omit-exponent .code:n =
214     {
215         \msg_info:nnnn { siunitx } { option-deprecated }
216         { table-omit-exponent }
217         { drop-uncertainty~and~exponent-mode~~fixed }
218         \str_if_eq:VnTF \l_keys_value_tl { true }
219         {
220             \keys_set:nn
221             { siunitx }
222             {
223                 drop-uncertainty = true ,
224                 exponent-mode    = fixed
225             }
226         }
227         {
228             \keys_set:nn
229             { siunitx }
230             { drop-uncertainty = false }

```

```

231     }
232   } ,
233   table-parse-only .code:n =
234   {
235     \msg_info:nnnn { siunitx } { option-deprecated }
236     { table-parse-only }
237     { table-alignment-mode~=-none }
238     \str_if_eq:VnT \l_keys_value_tl { true }
239     {
240       \keys_set:nn
241       { siunitx }
242       { table-alignment-mode = none }
243     }
244   } ,
245   table-space-text-post .code:n =
246   {
247     \msg_info:nnnn { siunitx } { option-deprecated }
248     { table-space-text-post }
249     { table-format }
250     \tl_set:Nn \l__siunitx_table_after_model_tl {#1}
251   } ,
252   table-space-text-pre .code:n =
253   {
254     \msg_info:nnnn { siunitx } { option-deprecated }
255     { table-space-text-post }
256     { table-format }
257     \tl_set:Nn \l__siunitx_table_before_model_tl {#1}
258   }
259 }

\__siunitx_option_table_format:n
\__siunitx_option_table_comparator:nnnnnnn
siunitx_option_table_figures-decimal:nnnnnnnn
siunitx_option_table_figures-exponent:nnnnnnnn
siunitx_option_table_figures-integer:nnnnnnnn
siunitx_option_table_figures-uncertainty:nnnnnnnn
siunitx_option_table_sign-exponent:nnnnnnnn
siunitx_option_table_sign-mantissa:nnnnnnnn

260 \cs_new_protected:Npn \__siunitx_option_table_format:n #1
261 {
262   \msg_info:nnnn { siunitx } { option-deprecated }
263   { table- #1 }
264   { table-format }
265   \tl_set:Nx \l__siunitx_table_format_tl
266   {
267     \cs:w __siunitx_option_table_ #1 :nnnnnnnn
268     \exp_after:wN \exp_after:wN \exp_after:wN \cs_end:
269     \exp_after:wN \l__siunitx_table_format_tl
270     \exp_after:wN { \l_keys_value_tl }
271   }
272   \exp_after:wN \__siunitx_table_generate_model:nnnnnnn
273   \l__siunitx_table_format_tl
274 }
275
276 \cs_new:Npn \__siunitx_option_table_comparator:nnnnnnnn #1#2#3#4#5#6#7#8
277 { \exp_not:n { {#8} {#2} {#3} {#4} {#5} {#6} {#7} } }
278 \cs_new:cpn { __siunitx_option_table_figures-decimal:nnnnnnnn }
279 #1#2#3#4#5#6#7#8
280 { \exp_not:n { {#1} {#2} {#3} {#8} {#5} {#6} {#7} } }
281 \cs_new:cpn { __siunitx_option_table_figures-exponent:nnnnnnnn }
282 #1#2#3#4#5#6#7#8

```

```

283 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#8} } }
284 \cs_new:cpn { __siunitx_option_table_figures-integer:nnnnnnnn }
285 #1#2#3#4#5#6#7#8
286 { \exp_not:n { {#1} {#2} {#8} {#4} {#5} {#6} {#7} } }
287 \cs_new:cpn { __siunitx_option_table_figures-uncertainty:nnnnnnnn }
288 #1#2#3#4#5#6#7#8
289 { \exp_not:n { {#1} {#2} {#8} {#4} {#5} {#6} {#7} } }
290 \cs_new:cpn { __siunitx_option_table_sign-exponent:nnnnnnnn }
291 #1#2#3#4#5#6#7#8
292 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#8} {#7} } }
293 \cs_new:cpn { __siunitx_option_table_sign-mantissa:nnnnnnnn }
294 #1#2#3#4#5#6#7#8
295 { \exp_not:n { {#1} {#8} {#3} {#4} {#5} {#6} {#7} } }

```

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

Options which all use the same emulation set up.

```

296 \keys_define:nn { siunitx }
297 {
298   table-comparator .code:n =
299   { \__siunitx_option_table_format:n { comparator } } ,
300   table-figures-decimal .code:n =
301   { \__siunitx_option_table_format:n { figures-decimal } } ,
302   table-figures-exponent .code:n =
303   { \__siunitx_option_table_format:n { figures-exponent } } ,
304   table-figures-integer .code:n =
305   { \__siunitx_option_table_format:n { figures-integer } } ,
306   table-figures-uncertainty .code:n =
307   { \__siunitx_option_table_format:n { figures-uncertainty } } ,
308   table-sign-exponent .code:n =
309   { \__siunitx_option_table_format:n { sign-exponent } } ,
310   table-sign-mantissa .code:n =
311   { \__siunitx_option_table_format:n { sign-mantissa } }
312 }
313 \</package>

```

1.2.4 Loadable configuration

```

314 < *v2 >

```

Emulation for input options: `input-protect-tokens` is a simple no-op, whilst `input-symbols` is combined with `input-digits`, so the latter is also re-defined.

```

315 \keys_define:nn { siunitx }
316 {
317   input-digits .code:n =
318   {
319     \tl_set:Nn \l__siunitx_number_input_realdigits_tl {#1}
320     \tl_set:Nx \l__siunitx_number_input_digit_tl
321     {
322       \exp_not:V \l__siunitx_number_input_realdigits_tl
323       \exp_not:V \l__siunitx_number_input_symbol_tl
324     }
325   } ,
326   input-protect-tokens .code:n =
327   {

```

```

328     \msg_info:nn { siunitx } { option-deprecated }
329     { input-protect-tokens }
330     { input-digits }
331   } ,
332   input-symbols .code:n =
333   {
334     \tl_set:Nn \l__siunitx_number_input_symbol_tl {#1}
335     \tl_set:Nx \l__siunitx_number_input_digit_tl
336       {
337         \exp_not:V \l__siunitx_number_input_realdigits_tl
338         \exp_not:V \l__siunitx_number_input_symbol_tl
339       }
340   }
341 }
342 \tl_new:N \l__siunitx_number_input_realdigits_tl
343 \tl_new:N \l__siunitx_number_input_symbol_tl

(End definition for \l__siunitx_number_input_realdigits_tl and \l__siunitx_number_input_symbol_
tl.)

Standard settings where appropriate.
344 \keys_set:nn { siunitx }
345 {
346   input-symbols = \dots\pi
347 }

The old s column type is handled by using the functionality of collcell.
348 \RequirePackage { collcell }
349 \AtBeginDocument
350 {
351   \__siunitx_declare_column:Nnn s
352     { \collectcell \unit }
353     { \endcollectcell }
354 }
355 </v2>

```

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

\, 43, 63, *90*, 966, 1647, 1653
 \- 340
 \\ 5, 939
 _ 8, 186, 197, 242, 301
 \~ 151
 \sq 89, 176, 179, 946

A

\A 2, *126*
 \ampere 2, 3, 4, 5, 6, 7, *92*, *797*
 \amu *35*, *126*
 \ang 4, *68*
 angle-mode 6
 \angstrom 72, 77, *94*, *907*
 \approx 1649
 arc-separator 6
 arc-separator-over-decimal 6
 \arcminute *94*, 166, *858*
 \arcsecond *94*, 167, *858*
 \as *84*, *126*
 \astronomicalunit *94*, *861*
 \AtBeginDocument 5,
 9, 33, 64, 70, 120, 133, 145, 200, 349
 \atomicmassunit 42, *94*, *861*
 \atto 85, *93*, *806*
 \auaction *94*, *881*
 \aucharge *94*, *881*
 \auenergy *94*, *881*
 \aulength *94*, *881*
 \aumass *94*, *881*
 \autime *94*, *881*

B

\bar *94*, *907*
 \barn *94*, *907*
 \becquerel *93*, *828*
 \bel 69, *94*, *907*
 \bfsries 56
 \bohr *94*, *881*
 \boldmath 57

bool commands:

\bool_if:NTF 10, 16,
 17, 18, 31, 33, 39, 54, 78, 88, 110,
 121, 122, 125, 131, 137, 144, 159,
 191, 215, 217, 219, 306, 341, 350,
 377, 404, 412, 434, 473, 474, 500,

513, 518, 526, 536, 545, 561, 590,
 645, 657, 674, 681, 694, 708, 717,
 720, 817, 827, 1102, 1230, 1337,
 1345, 1354, 1402, 1477, 1546, 1559
 \bool_lazy_all:nTF 1377
 \bool_lazy_all_p:n 913, 931
 \bool_lazy_and:nnTF 104,
 184, 428, 556, 574, 615, 659, 667, 1541
 \bool_lazy_or:nnTF
 5, 24, 88, 111, 173, 283,
 296, 440, 503, 776, 910, 928, 1190, 1537
 \bool_new:N 6,
 9, 10, 35, 36, 75, 258, 468, 469, 470,
 471, 472, 473, 474, 476, 599, 1306, 1307
 \bool_set_false:N 9, 10,
 14, 19, 20, 107, 155, 264, 265, 416,
 417, 423, 424, 425, 426, 430, 431,
 432, 437, 440, 444, 492, 494, 535,
 557, 588, 641, 1264, 1268, 1273, 1274
 \bool_set_true:N 9,
 14, 15, 112, 225, 263, 288, 352, 418,
 419, 433, 438, 439, 445, 446, 447,
 451, 452, 453, 454, 495, 586, 662,
 701, 1258, 1259, 1263, 1269, 1574, 1575
 \c_false_bool 46, 96, 384, 446, 450,
 455, 457, 464, 488, 494, 508, 512, 527
 \c_true_bool 43, 93,
 369, 384, 444, 457, 461, 488, 494, 508

box commands:

\box_clear:N 440, 489
 \box_new:N ... 3, 12, 229, 230, 416, 417
 \box_use_drop:N 365, 366, 410, 411,
 463, 470, 480, 481, 533, 534, 535, 536
 \box_wd:N 238,
 239, 242, 250, 342, 381, 385, 399,
 438, 445, 446, 449, 457, 487, 493,
 524, 548, 559, 560, 561, 576, 580,
 581, 594, 595, 605, 613, 614, 616,
 633, 634, 655, 666, 685, 687, 697, 709
 bracket-negative 13
 bracket-unit-denominator 95

C

\cancel 82, *90*, *95*, *101*
 \candela *92*, *797*
 \centi 65, *93*, *806*
 \char 927

<code>\degreeCelsius</code>	93, 828	184, 185, 187, 188, 190, 191, 198,
<code>\deka</code>	93, 817	201, 202, 204, 208, 212, 214, 217,
dim commands:		220, 222, 224, 225, 226, 227, 229,
<code>\dim_add:Nn</code>	706	230, 232, 234, 235, 235, 236, 236,
<code>\dim_compare:nNnTF</code>	237, 438, 444, 487	237, 238, 238, 239, 240, 241, 243,
<code>\dim_compare_p:nNn</code>	576	244, 244, 245, 245, 246, 246, 247,
<code>\dim_new:N</code>	4, 418	248, 254, 255, 297, 298, 300, 302,
<code>\dim_set:Nn</code>	604, 631, 655, 666, 694	343, 344, 517, 792, 794, 868, 870,
<code>\dm</code>	60, 126	871, 876, 877, 885, 887, 888, 893,
<code>\document</code>	14, 59	894, 898, 900, 901, 1361, 1392, 1548
<code>\dots</code>	346	<code>\exp_not:n</code>
drop-exponent	13	107,
drop-uncertainty	13	128, 140, 155, 162, 163, 182, 188,
drop-zero-decimal	13	207, 224, 229, 234, 236, 239, 239,
<code>\dyne</code>	94, 917	244, 245, 246, 246, 253, 254, 255,
		277, 279, 280, 283, 286, 289, 292,
		295, 303, 313, 322, 323, 337, 338,
		353, 354, 362, 363, 433, 472, 515,
		516, 517, 519, 525, 549, 555, 596,
		597, 598, 599, 608, 609, 610, 619,
		620, 621, 629, 630, 643, 647, 648,
		651, 654, 662, 666, 680, 685, 687,
		688, 698, 699, 704, 716, 721, 735,
		736, 737, 738, 740, 745, 746, 747,
		754, 755, 760, 766, 768, 769, 770,
		771, 771, 772, 777, 779, 785, 787,
		788, 798, 801, 804, 807, 824, 834,
		846, 1112, 1115, 1119, 1151, 1158,
		1160, 1193, 1197, 1200, 1202, 1238,
		1240, 1243, 1244, 1330, 1331, 1355,
		1356, 1361, 1370, 1372, 1388, 1394,
		1396, 1406, 1409, 1428, 1434, 1435,
		1448, 1449, 1453, 1457, 1474, 1479,
		1481, 1487, 1488, 1489, 1490, 1536,
		1544, 1549, 1551, 1552, 1554, 1566
		<code>\expandafter</code>
		125
		exponent-base
		13
		exponent-mode
		13
		exponent-product
		14
		expression
		14
		F
		<code>\F</code>
		70, 128
		<code>\fam</code>
		29, 59, 161, 174, 175
		<code>\farad</code>
		70, 71, 72, 93, 828
		<code>\femto</code>
		15, 35, 71, 86, 93, 806
		<code>\fF</code>
		70, 128
		<code>\fg</code>
		35, 126
		<code>\fi</code>
		145
		fi commands:
		<code>\fi:</code>
		27, 33, 274
		fill-arc-degrees
		6
		fill-arc-minutes
		6
		fill-arc-seconds
		6
		fixed-exponent
		14
<code>\degreeCelsius</code>	93, 828	
<code>\deka</code>	93, 817	
dim commands:		
<code>\dim_add:Nn</code>	706	
<code>\dim_compare:nNnTF</code>	237, 438, 444, 487	
<code>\dim_compare_p:nNn</code>	576	
<code>\dim_new:N</code>	4, 418	
<code>\dim_set:Nn</code>	604, 631, 655, 666, 694	
<code>\dm</code>	60, 126	
<code>\document</code>	14, 59	
<code>\dots</code>	346	
drop-exponent	13	
drop-uncertainty	13	
drop-zero-decimal	13	
<code>\dyne</code>	94, 917	
		E
<code>\electronvolt</code>	53, 54, 55, 56, 57, 58, 94, 861	
<code>\else</code>	145	
else commands:		
<code>\else:</code>	272	
<code>\end</code>	54, 75	
<code>\endcollectcell</code>	353	
<code>\endinput</code>	13	
<code>\ensuremath</code>	24, 26, 56, 90, 96,	
	141, 239, 298, 340, 379, 383, 830,	
	865, 868, 876, 881, 882, 885, 893, 898	
<code>\erg</code>	94, 917	
<code>\eV</code>	43, 127	
evaluate-expression	13	
<code>\exa</code>	93, 817	
exp commands:		
<code>\exp_after:wN</code>	39, 41, 107,	
	108, 111, 115, 122, 135, 137, 144,	
	146, 167, 170, 211, 268, 269, 270,	
	271, 273, 273, 295, 355, 368, 369,	
	372, 468, 480, 507, 520, 524, 540,	
	548, 607, 632, 638, 791, 793, 794,	
	820, 830, 841, 1084, 1233, 1315, 1611	
<code>\exp_args:Nc</code>	122	
<code>\exp_args:Nf</code>	644, 887, 898, 1144, 1482	
<code>\exp_args:NNc</code>	123	
<code>\exp_args:Nnno</code>	1198	
<code>\exp_args:NNNV</code>	25, 97, 174, 291, 512, 729	
<code>\exp_args:NNnx</code>	107, 812	
<code>\exp_args:NNV</code>	16, 285	
<code>\exp_args:Nv</code>	108, 272, 1042	
<code>\exp_args:Nv</code>	77	
<code>\exp_args:Nx</code>	56, 502	
<code>\exp_last_unbraced:Nn</code>		
	34, 319, 786, 980, 1207	
<code>\exp_not:N</code>		
	40, 42, 50, 96, 99, 100, 115, 139,	
	141, 156, 157, 158, 159, 161, 183,	

<code>\fmol</code>	14 , 127	<code>\hbox_set_end:</code>	355 , 363 , 397 , 407
<code>font-command</code>	96	<code>\hbox_set_to_wd:Nnn</code> 241 , 249 , 341 , 380 , 448 , 456 ,
<code>\fontsize</code>	310		492 , 523 , 547 , 557 , 578 , 592 , 611 , 683
<code>forbid-literal-units</code>	96	<code>\hbox_set_to_wd:Nnw</code>	384 , 398
fp commands:		<code>\hbox_to_wd:nn</code>	192
<code>\fp_add:Nn</code>	575	<code>\hbox_unpack:N</code> 244 , 253 , 452 , 459 , 496 , 565 ,
<code>\fp_compare:nNnTF</code>	377		584 , 599 , 608 , 620 , 690 , 692 , 703 , 704
<code>\fp_eval:n</code> ..	42 , 57 , 66 , 67 , 69 , 89 , 376	<code>\hectare</code>	93 , 851
<code>\fp_new:N</code>	4 , 477	<code>\hecto</code>	27 , 31 , 93 , 817
<code>\fp_set:Nn</code>	141	<code>\henry</code>	93 , 828
<code>\fp_use:N</code>	142	<code>\hertz</code>	8 , 9 , 10 , 11 , 12 , 13 , 93 , 828
<code>\fp_zero:N</code>	121 , 487	<code>\highlight</code>	82 , 95 , 101
<code>\l_tmpa_fp</code>	91	<code>\hL</code>	27 , 127
<code>\frac</code>	90 , 965	<code>\hl</code>	27 , 127
<code>fraction-command</code>	96	<code>\hour</code>	59 , 93 , 851
<code>\fs</code>	84 , 126	<code>\Hz</code>	8 , 126
G			
<code>\g</code>	35 , 126	I	
<code>\gal</code>	94 , 917	if commands:	
<code>\gauss</code>	94 , 917	<code>\if_false:</code>	27 , 33
<code>\ge</code>	337 , 1649	<code>\if_meaning:w</code>	270
<code>\geq</code>	1649	<code>\ifmmode</code>	145
<code>\GeV</code>	43 , 127	<code>\IfNoValueF</code>	35
<code>\gg</code>	336 , 1649	<code>\ignorespaces</code>	19 , 36 , 100 , 325
<code>\GHz</code>	8 , 126	<code>input-close-uncertainty</code>	14 , 14
<code>\giga</code>	12 , 48 , 57 , 80 , 93 , 817	<code>input-comparators</code>	14
<code>\GPa</code>	77 , 127	<code>input-decimal-markers</code>	14
<code>\gram</code> ..	35 , 36 , 37 , 38 , 39 , 40 , 41 , 92 , 797 , 805	<code>input-digits</code>	14
<code>\gray</code>	93 , 828	<code>input-exponent-markers</code>	14
group commands:		<code>input-open-uncertainty</code>	14
<code>\group_begin:</code>	13 , 16 ,	<code>input-signs</code>	14
	22 , 23 , 24 , 33 , 47 , 59 , 71 , 79 , 84 , 86 ,	<code>input-uncertainty-signs</code>	14
	88 , 97 , 117 , 133 , 150 , 154 , 182 , 204 ,	int commands:	
	226 , 282 , 287 , 296 , 339 , 499 , 727 , 1573	<code>\int_abs:n</code>	157 , 722 , 736 , 1508
<code>\c_group_begin_token</code> ...	40 , 133 , 328	<code>\int_case:nNnTF</code>	161 , 746
<code>\group_end:</code>	16 ,	<code>\int_compare:nNnTF</code>	142 , 164 ,
	25 , 25 , 28 , 28 , 32 , 46 , 55 , 59 , 66 ,		315 , 655 , 686 , 689 , 703 , 715 , 733 ,
	74 , 83 , 92 , 97 , 102 , 131 , 141 , 144 ,		744 , 811 , 885 , 896 , 957 , 960 , 991 ,
	148 , 174 , 179 , 192 , 208 , 230 , 285 ,		1031 , 1049 , 1082 , 1093 , 1110 , 1125 ,
	291 , 304 , 346 , 512 , 729 , 1579 , 1583		1133 , 1152 , 1163 , 1404 , 1496 , 1518
<code>group-digits</code>	14	<code>\int_compare_p:nNn</code> 174 , 175 , 911 , 929 , 1192
<code>group-minimum-digits</code>	14	<code>\int_const:Nn</code>	29
<code>group-separator</code>	14	<code>\int_do_while:nNnn</code>	489
<code>\GW</code>	43 , 127	<code>\int_eval:n</code> ..	149 , 305 , 332 , 358 , 398 ,
H			
<code>\hartree</code>	94 , 881		645 , 888 , 899 , 967 , 995 , 1136 , 1145 ,
<code>\hbar</code>	90 , 865 , 881		1173 , 1183 , 1416 , 1483 , 1499 , 1522
hbox commands:		<code>\int_if_odd_p:n</code>	916 , 934
<code>\hbox_set:Nn</code> 24 , 339 , 378 , 383 , 437 , 443 ,	<code>\int_incr:N</code>	337 , 496
	475 , 477 , 485 , 486 , 522 , 606 , 673 , 701	<code>\int_mod:nn</code>	746 , 753 , 1416
<code>\hbox_set:Nw</code>	344 , 356	<code>\int_new:N</code>	5 , 259 , 481
		<code>\int_set:Nn</code>	288

<code>\int_set_eq:NN</code>	484	<code>\lumen</code>	93, 828
<code>\int_step_inline:nn</code>	268	<code>\lux</code>	93, 828
<code>\int_use:N</code>	290, 295, 318, 322, 339, 344, 506, 571, 648		
<code>\int_zero:N</code>	266, 488		
<code>inter-unit-product</code>	96		
J			
<code>\J</code>	43, 127		
<code>\joule</code>	49, 50, 51, 52, 93, 828		
K			
<code>\K</code>	68, 126		
<code>\kA</code>	2, 126		
<code>\katal</code>	93, 828		
<code>\kelvin</code>	68, 92, 797		
<code>\kern</code>	830		
<code>\keV</code>	43, 127		
keys commands:			
<code>\l_keys_choice_tl</code>			
.	48, 70, 263, 277, 460, 577, 593		
<code>\keys_define:nn</code>	4, 10, 30, 36, 48, 56, 97, 142, 176, 188, 216, 253, 257, 296, 315, 403, 419, 567, 1247		
<code>\keys_set:nn</code>	13, 18, 25, 34, 51, 60, 72, 80, 87, 89, 98, 105, 142, 149, 157, 177, 186, 209, 220, 228, 240, 289, 314, 344, 502, 723, 960, 1633		
<code>\l_keys_value_tl</code>	87, 94, 101, 108, 195, 202, 218, 238, 270		
<code>\kg</code>	35, 126		
<code>\kHz</code>	8, 126		
<code>\kilo</code>	7, 10, 20, 26, 41, 46, 52, 55, 59, 67, 75, 78, 82, 93, 106, 797, 817		
<code>\kilogram</code>	92, 92, 93, 797		
<code>\kJ</code>	43, 127		
<code>\km</code>	60, 126		
<code>\kmol</code>	14, 127		
<code>\kN</code>	73, 127		
<code>\knot</code>	94, 907		
<code>\kohm</code>	81, 128		
<code>\kPa</code>	77, 127		
<code>\kV</code>	21, 127		
<code>\kW</code>	43, 127		
<code>\kWh</code>	43, 128		
L			
<code>\L</code>	27, 127		
<code>\l</code>	27, 127		
<code>\le</code>	335, 1649		
<code>\leq</code>	1649		
<code>\liter</code>	31, 32, 33, 34, 93, 851		
<code>\litre</code>	27, 28, 29, 30, 93, 851		
<code>\ll</code>	334, 1649		
M			
<code>\m</code>	60, 126		
<code>\mA</code>	2, 126		
math commands:			
<code>\c_math_toggle_token</code>	345, 354, 357, 362, 386, 396, 400, 405, 414, 415		
<code>\mathchoice</code>	90, 118, 734		
<code>\mathit</code>	90, 865, 870, 877, 881, 882, 887, 894, 900		
<code>\mathord</code>	340, 401, 1330, 1355, 1392, 1548		
<code>\mathrm</code>	50, 56, 90, 177, 871, 888, 901, 962		
<code>\mathsf</code>	62, 149, 163		
<code>\mathhtt</code>	62, 150, 164		
<code>\mathversion</code>	56, 57, 58, 134		
<code>\maxwell</code>	94, 917		
<code>\mbox</code>	66, 90, 308		
<code>\mdseries</code>	56, 57, 218		
<code>\mega</code>	11, 47, 56, 76, 79, 83, 93, 817		
<code>\MessageBreak</code>	10		
<code>\meter</code>	92, 120, 797		
<code>\metre</code>	60, 61, 62, 63, 64, 65, 66, 67, 92, 93, 106, 120, 797		
<code>\MeV</code>	43, 127		
<code>\meV</code>	43, 127		
<code>\mg</code>	35, 126		
<code>\MHz</code>	8, 126		
<code>\mHz</code>	8, 126		
<code>\micro</code>	5, 18, 24, 30, 34, 38, 44, 50, 63, 89, 93, 105, 107, 806		
<code>\milli</code>	6, 9, 19, 25, 29, 33, 39, 45, 51, 54, 64, 74, 81, 90, 93, 806		
<code>\millimetremercury</code>	94, 907		
<code>minimum-decimal-digits</code>	14		
<code>minimum-integer-digits</code>	14		
<code>\minute</code>	93, 94, 851		
<code>\mJ</code>	43, 127		
<code>\mL</code>	27, 127		
<code>\ml</code>	27, 127		
<code>\mm</code>	60, 126		
<code>\mmol</code>	14, 127		
<code>\MN</code>	73, 127		
<code>\mN</code>	73, 127		
<code>mode</code>	57		
mode commands:			
<code>\mode_if_math:TF</code>	65, 90, 130		
<code>\mode_leave_vertical:</code>			
.	23, 32, 58, 70, 78, 87, 96		
<code>\Mohm</code>	81, 128		
<code>\mohm</code>	81, 128		
<code>\mol</code>	14, 127		
<code>\mole</code>	14, 15, 16, 17, 18, 19, 20, 92, 797		

\mp	238, 239, 332, 1655	output-open-uncertainty	15
\MPa	77, 127	\over	97
\ms	84, 126		
msg commands:		P	
\msg_error:nn	354	\Pa	77, 127
\msg_error:nnn	26, 127, 132, 400	\pA	2, 126
\msg_error:nnnn	278, 311, 325	\PackageError	7
\msg_info:nn	328	parse-numbers	15
\msg_info:nnnn	12, 17, 174, 206, 215, 235, 247, 254, 262	parse-units	96
\msg_new:nnn	3, 8	\pascal	77, 78, 79, 80, 93, 840
\msg_new:nnnn	17, 930, 936, 942, 948, 954, 1627	peek commands:	
\msg_warning:nn	52	\peek_after:Nw	277
\msg_warning:nnn	66, 78, 146, 163	\peek_catcode_ignore_spaces:NTF	40, 133, 328
\mV	21, 127	\per	79, 95, 96, 97, 100, 106, 106, 108, 109, 111, 345, 353, 359, 880, 904, 943, 946
\MW	43, 127	per-mode	96
\mW	43, 127	per-symbol	96
		\percent	94, 927
N		\peta	93, 817
\N	73, 127	\pF	70, 128
\nA	2, 126	\pg	35, 126
\nano	4, 17, 23, 37, 62, 88, 93, 806	\phot	94, 917
\nauticalmile	94, 907	\pi	346
negative-color	14	\pico	3, 16, 22, 36, 61, 72, 87, 93, 806
\neper	94, 907	\pm	60, 98, 126, 236, 333, 1480, 1655, 1656
\newcolumntype	119, 139	\pmol	14, 127
\NewDocumentCommand	21, 30, 40, 44, 48, 52, 56, 68, 76, 85, 94, 104	\poise	94, 917
\newton	73, 74, 75, 76, 93, 840	\power	95, 95
\ng	35, 126	prg commands:	
\nm	60, 126	\prg_new_conditional:Npnn	1080, 1587
\nmol	14, 127	\prg_new_protected_conditional:Npnn	11, 45, 1570
\nobreak	39, 43, 63, 175, 181	\prg_replicate:nn	157, 304, 305, 317, 322, 619, 722, 813, 1103, 1241, 1508
\ns	84, 126	\prg_return_false:	19, 60, 1087, 1095, 1580, 1605
\nuaction	94, 865	\prg_return_true:	18, 56, 1092, 1584, 1622
\num	68, 88	print-implicit-plus	15
\numass	94, 865	print-unity-mantissa	15
number-angle-product	6	print-zero-exponent	15
number-close-bracket	15	prop commands:	
number-color	57	\prop_clear:N	262
number-mode	57	\prop_get:NnNTF	112, 370, 507, 567, 572
number-open-bracket	15	\prop_if_empty:NTF	127
\nuspeed	94, 865	\prop_if_in:NnTF	275, 319, 365, 399
\nutime	94, 865	\prop_if_in_p:Nn	299
\nV	21, 127	\prop_new:N	60, 61, 94, 257
		\prop_put:Nnn	57, 58, 95, 96, 97, 98, 282, 383, 388
O		\prop_remove:Nn	367, 379
\oersted	94, 917	propagate-math-font	57
\of	88, 95		
\ohm	81, 82, 83, 84, 86, 93, 840		
\Omega	100		
output-close-uncertainty	15		
output-decimal-marker	15		

<code>\ProvidesExplPackage</code>	15	<code>reset-text-shape</code>	57
<code>\ps</code>	84, 126	<code>\rmdefault</code>	148
<code>\pV</code>	21, 127	<code>\rmfamily</code>	56, 57, 216
Q			
<code>\qty</code>	56, 133	<code>round-half</code>	15
<code>qualifier-mode</code>	96	<code>round-minimum</code>	15
<code>qualifier-phrase</code>	97	<code>round-mode</code>	15
<code>quark commands:</code>		<code>round-pad</code>	15
<code>\q_mark</code>	208,	<code>round-precision</code>	15
217, 517, 542, 568, 571, 624, 627,		S	
636, 639, 644, 647, 649, 651, 658, 661		<code>\s</code>	84, 126
<code>\q_nil</code>	233, 254,	<code>\scriptspace</code>	830
371, 376, 467, 473, 516, 518, 542,		<code>\second</code> 84, 85, 86, 87, 88, 89, 90, 92, 94, 797	
571, 627, 639, 647, 648, 651, 652,		<code>\selectfont</code>	310
661, 1421, 1423, 1425, 1442, 1502, 1529		<code>separate-uncertainty</code>	15
<code>\quark_if_nil:N</code> TF		<code>seq commands:</code>	
.. 1429, 1436, 1446, 1450, 1454, 1516		<code>\seq_map_inline:Nn</code>	118
<code>\quark_if_nil:n</code> TF	262	<code>\seq_new:N</code>	21
<code>\quark_if_recursion_tail_stop:N</code> .		<code>\seq_put_right:Nn</code>	28
... 146, 198, 252, 305, 326, 537, 1616		<code>\sfdefault</code>	149
<code>\quark_if_recursion_tail_stop:n</code> .		<code>\SI</code>	30
... 182, 194		<code>\si</code>	21
<code>\quark_if_recursion_tail_stop-</code>		<code>\siemens</code>	93, 840
<code>do:Nn</code> ... 297, 375, 397, 482, 499,		<code>\sievert</code>	93, 840
855, 864, 880, 894, 905, 926, 944,		<code>\sim</code>	1649
974, 1048, 1077, 1091, 1123, 1131, 1605		<code>\sisetup</code>	104
<code>\q_recursion_stop</code>	127, 172,	<code>siunitx commands:</code>	
190, 192, 204, 248, 292, 314, 322,		<code>\siunitx_angle:n</code>	6, 6, 37, 109
371, 379, 483, 500, 541, 850, 858,		<code>\siunitx_angle:nnn</code> .	6, 6, 37, 112, 113
869, 873, 883, 908, 954, 1000, 1007,		<code>\siunitx_cell_begin:w</code>	7, 99, 150
1044, 1073, 1085, 1114, 1601, 1612		<code>\siunitx_cell_end:</code> 7, 57, 78, 101, 152	
<code>\q_recursion_tail</code> 127, 172, 190, 192,		<code>\siunitx_declare_power:NNn</code>	
204, 246, 292, 321, 371, 850, 858,		... 39, 42, 91, 928, 929	
869, 873, 883, 908, 954, 1000, 1007,		<code>\siunitx_declare_power:NnN</code>	91
1044, 1073, 1085, 1114, 1600, 1612		<code>\siunitx_declare_prefix:Nn</code> 48, 91, 91	
<code>\q_stop</code>	93, 195,	<code>\siunitx_declare_prefix:Nnn</code>	
202, 212, 212, 214, 220, 247, 255,		... 46, 48, 91, 91,	
268, 273, 356, 360, 373, 376, 469,		107, 806, 807, 808, 809, 810, 811,	
473, 521, 525, 529, 541, 542, 548,		812, 814, 815, 816, 817, 818, 819,	
549, 551, 568, 571, 624, 627, 636,		820, 821, 822, 823, 824, 825, 826, 827	
639, 644, 648, 649, 652, 658, 660,		<code>\siunitx_declare_qualifier:Nn</code> ...	
661, 672, 677, 687, 690, 695, 699,		... 50, 62, 91	
705, 707, 726, 729, 768, 778, 782, 787		<code>\siunitx_declare_unit:Nn</code> 54, 68, 77,	
R			
<code>\radian</code>	93, 840	86, 92, 797, 798, 799, 800, 801, 802,	
<code>\raiseto</code>	91, 95	803, 804, 805, 828, 829, 831, 832,	
<code>\relax</code>	55, 76	833, 834, 835, 836, 837, 838, 839,	
<code>\renewcommand</code>	123	840, 841, 842, 843, 844, 845, 846,	
<code>\RequirePackage</code> ... 3, 3, 4, 9, 39, 116, 348		847, 848, 849, 850, 851, 852, 853,	
<code>reset-math-version</code>	57	854, 855, 856, 857, 858, 859, 860,	
<code>reset-text-family</code>	57	861, 862, 863, 864, 865, 866, 874,	
<code>reset-text-series</code>	57	879, 881, 882, 883, 891, 896, 904,	
		905, 906, 907, 909, 910, 911, 912,	
		913, 914, 915, 916, 917, 918, 919,	
		920, 921, 922, 923, 924, 925, 926, 927	

\siunitx_format_number:nN	6	\l__siunitx_angle_fill_minutes_-	
\siunitx_if_number:nTF	13, 1570	bool	4
\siunitx_if_number_p:n	13	\l__siunitx_angle_fill_seconds_-	
\siunitx_if_number_token:N	13, 13, 149, 1587	bool	4
\siunitx_if_number_token_p:N	1587	\l__siunitx_angle_force_arc_bool	4, 39
\siunitx_number_format:N	13, 22, 122, 137, 1308	\l__siunitx_angle_force_decimal_-	4, 54
\siunitx_number_format:NN	13, 371, 467, 516, 518, 1308	\l__siunitx_angle_minutes_tl	80, 129
\siunitx_number_format:nN	12, 14, 41, 45, 61, 81, 718	\l__siunitx_angle_product_tl	4, 176
\l__siunitx_number_input_decimal_-		\l__siunitx_angle_seconds_tl	80, 130
tl	13, 29, 41, 346, 387, 387, 402, 1591	\l__siunitx_angle_separator_tl	4, 182
\l__siunitx_number_output_-		\l__siunitx_angle_sign:nnnnnn	84
decimal_tl	13, 340, 358, 401, 1246, 1290, 1391, 1394	\l__siunitx_angle_sign_tl	83, 94, 98, 107, 156
\siunitx_number_parse:nN	12, 19, 76, 145, 290, 464, 498, 1576	\l__siunitx_angle_tmp_tl	3, 145, 146, 152, 177, 178
\l__siunitx_number_parse_bool	9, 10, 10, 13, 17, 18, 62, 78, 288, 1575	__siunitx_declare_column:Nnn	117, 351
\siunitx_number_process:N	12	__siunitx_load_check:	17
\siunitx_number_process:NN	12, 20, 465, 511, 602	__siunitx_load_check:n	20, 32, 36
\siunitx_print:nn	27, 38, 42, 45, 56, 57, 57, 57, 58, 62, 65, 73, 82, 91, 174, 178, 476, 478, 552, 586, 600, 609, 676, 719	\l__siunitx_number_arg_tl	22, 25, 68, 86, 92, 94, 212, 219, 222, 238, 253, 265, 327, 343, 370, 549, 555, 563
\l__siunitx_print_series_prop	56, 58, 94, 112	\l__siunitx_number_bracket_-	
\l__siunitx_unit_font_tl	92, 97, 101, 104, 235, 245, 597, 609, 620, 698	close_tl	1247, 1372
\siunitx_unit_format:nN	26, 37, 44, 52, 64, 90, 90, 90, 91, 102, 105, 177	\l__siunitx_number_bracket_-	
\siunitx_unit_format:nNN	91, 105	negative_bool	1247, 1345
\siunitx_unit_power_set:NnN	101	\l__siunitx_number_bracket_open_-	
\l__siunitx_unit_symbolic_seq	21, 28, 92, 118	tl	1247, 1370
siunitx internal commands:		\l__siunitx_number_comparator_tl	69, 218, 221, 353
__siunitx_angle:nnn	73, 106	__siunitx_number_digits:NN	610, 791
__siunitx_angle_arc_convert:n	37	__siunitx_number_digits:Nn	791
__siunitx_angle_arc_print:nnn	46, 127, 163	__siunitx_number_digits:nnnnnn	791
__siunitx_angle_arc_print_-		__siunitx_number_drop_exponent:NN	608, 815
aux:nnn	163	__siunitx_number_drop_exponent:nnnnnn	815
__siunitx_angle_arc_sign:nn	84	\l__siunitx_number_drop_exponent_-	
__siunitx_angle_arc_sign:nnn	59, 84	bool	567, 817
\l__siunitx_angle_astronomy_bool	4	__siunitx_number_drop_uncertainty:NN	606, 825
\l__siunitx_angle_degrees_tl	45, 47, 80, 128	__siunitx_number_drop_uncertainty:nnnnnn	825
__siunitx_angle_extract_-		\l__siunitx_number_drop_uncertainty_-	
sign:nnnnnnnn	84	bool	567, 827
\l__siunitx_angle_fill_degrees_-		\l__siunitx_number_drop_zero_-	
bool	4	decimal_bool	567, 1230
		\l__siunitx_number_explicit_-	
		plus_bool	30, 284, 558
		__siunitx_number_exponent:NN	622, 626

\l__siunitx_number_group_-		\l__siunitx_number_output_-	
decimal_bool	1247	uncert_open_tl	1247 , 1488
\l__siunitx_number_group_-		__siunitx_number_parse:nN	76
integer_bool	1247	__siunitx_number_parse_check: ..	96 , 100
\l__siunitx_number_group_-		__siunitx_number_parse_combine_-	
minimum_int	1247 , 1405	uncert:	118 , 133
\l__siunitx_number_group_-		__siunitx_number_parse_combine_-	
separator_tl	1247 , 1434 , 1459	uncert_auxi:nnnnnnnn	133
__siunitx_number_if_number_-		__siunitx_number_parse_combine_-	
token_auxi:NN	1587	uncert_auxii:nnnnn	133
__siunitx_number_if_number_-		__siunitx_number_parse_combine_-	
token_auxii:NN	1587	uncert_auxiii:nnnnnn	133
__siunitx_number_if_number_-		__siunitx_number_parse_combine_-	
token_auxiii:NN	1587	uncert_auxiv:nnnn	133
\l__siunitx_number_implicit_-		__siunitx_number_parse_combine_-	
plus_bool	1247 , 1337 , 1559	uncert_auxv:w	133
\l__siunitx_number_input_-		__siunitx_number_parse_combine_-	
comparator_tl	30 , 216 , 1593	uncert_auxvi:w	133
\l__siunitx_number_input_digit_-		__siunitx_number_parse_comparator:	95 , 209
tl	30 , 306 , 320 , 335 , 381 , 399 , 484 , 501 , 1594	__siunitx_number_parse_comparator_-	
\l__siunitx_number_input_-		aux:Nw	209
exponent_tl	30 , 227 , 235 , 236 , 1595	__siunitx_number_parse_exponent:	225 , 565
\l__siunitx_number_input_ignore_-		__siunitx_number_parse_exponent_-	
tl	30 , 1596	auxi:w	225
\l__siunitx_number_input_-		__siunitx_number_parse_exponent_-	
realdigits_tl	315	auxii:nn	225
\l__siunitx_number_input_sign_tl	30 , 275 , 414 , 553 , 1598	__siunitx_number_parse_exponent_-	
\l__siunitx_number_input_symbol_-		auxiii:Nw	225
tl	315	__siunitx_number_parse_exponent_-	
\l__siunitx_number_input_tl	73 , 92 , 128	auxiv:nn	225
\l__siunitx_number_input_uncert_-		__siunitx_number_parse_exponent_-	
close_tl	30 , 516 , 1592	check:N	225
\l__siunitx_number_input_uncert_-		__siunitx_number_parse_exponent_-	
open_tl	30 , 409 , 1597	cleanup:N	225
\l__siunitx_number_input_uncert_-		__siunitx_number_parse_exponent_-	
sign_tl	30 , 116 , 1599	cleanup:wN	311 , 313
\l__siunitx_number_min_decimal_-		__siunitx_number_parse_exponent_-	
int	567 , 805	zero_test:N	225
\l__siunitx_number_min_integer_-		__siunitx_number_parse_finalise:	131 , 347
int	567 , 800	__siunitx_number_parse_finalise:nw	347
\l__siunitx_number_negative_-		__siunitx_number_parse_loop:	231 , 271 , 365
color_tl	1247 , 1343 , 1361	__siunitx_number_parse_loop_-	
__siunitx_number_number_token_-		after_decimal:NNN	365
auxi:NN	1590 , 1603 , 1607	__siunitx_number_parse_loop_-	
__siunitx_number_number_token_-		after_uncert:N	365
auxii:NN	1606 , 1609	__siunitx_number_parse_loop_-	
__siunitx_number_number_token_-		break:wN	365
auxiii:NN	1611 , 1614 , 1625		
\l__siunitx_number_output_-			
uncert_close_tl	1247 , 1490		

_siunitx_number_parse_loop_- first:N	365	_siunitx_number_round_auxiii:nnnN	847
_siunitx_number_parse_loop_- first:NNN	368 , 373 , 463	_siunitx_number_round_auxiv:nnN	847
_siunitx_number_parse_loop_- main:NNNNN	365	_siunitx_number_round_auxiv:nnnN	868 , 882 , 892 , 898
_siunitx_number_parse_loop_- main_decimal:NN	365	_siunitx_number_round_auxv:nnN	847
_siunitx_number_parse_loop_- main_digit:NNNNN	365	_siunitx_number_round_auxvi:nN	847
_siunitx_number_parse_loop_- main_end:NN	365	_siunitx_number_round_auxvi:nnN	907
_siunitx_number_parse_loop_- main_sign:NNN	365	_siunitx_number_round_auxvi:nnnN	901 , 924
_siunitx_number_parse_loop_- main_store:NNN	365	_siunitx_number_round_auxvii:nnN	847
_siunitx_number_parse_loop_- main_uncert:NNN	365	_siunitx_number_round_auxviii:nnN	847
_siunitx_number_parse_loop_- root_swap:NNwNN	365	_siunitx_number_round_engineering:nn	847
_siunitx_number_parse_loop_- uncert:NNNNN	365	_siunitx_number_round_engineering:nnN	847
_siunitx_number_parse_replace:	93 , 316	_siunitx_number_round_engineering:nnNn	847
_siunitx_number_parse_replace_- aux:nN	316	_siunitx_number_round_figures:nnnnnnn	1106
_siunitx_number_parse_replace_- minus:	318 , 341	_siunitx_number_round_figures_- count:nnN	1106
_siunitx_number_parse_replace_- sign:	316	_siunitx_number_round_figures_- count:nnnN	1106
_siunitx_number_parse_sign:	223 , 546	_siunitx_number_round_final:nn	847
_siunitx_number_parse_sign_- aux:Nw	546	_siunitx_number_round_final_- decimal:nnw	847
\c_siunitx_number_parse_sign_- replacement_tl	316	_siunitx_number_round_final_- integer:nnw	847
\l_siunitx_number_parsed_tl 19 , 20 , 20 , 22 , 72 , 85 , 98 , 107 , 119 , 121 , 123 , 137 , 144 , 179 , 181 , 230 , 267 , 270 , 279 , 289 , 315 , 349 , 351 , 354 , 369 , 544 , 559 , 560 , 562 , 564 , 1576 , 1577		_siunitx_number_round_final_- output:nn	847
\l_siunitx_number_partial_tl 74 , 367 , 429 , 430 , 433 , 443 , 468 , 469 , 472 , 476 , 486 , 506 , 521 , 524 , 525		_siunitx_number_round_final_- shift:nn	847
_siunitx_number_process:nnnnnnn	602	_siunitx_number_round_final_- shift:Nw	847
_siunitx_number_round:NN .	623 , 835	_siunitx_number_round_fixed:nn	847
_siunitx_number_round:nnn 847 , 1135 , 1171 , 1181 , 1209 , 1218 , 1223		\l_siunitx_number_round_half_- even_bool	567 , 915 , 933
_siunitx_number_round_auxi:nnnN	847	_siunitx_number_round_if_- half:N	1080
_siunitx_number_round_auxii:nnnN	847	_siunitx_number_round_if_- half:n	1080
		_siunitx_number_round_if_half_- p:n	917 , 935 , 1080
		_siunitx_number_round_input:nn	847
		\l_siunitx_number_round_min_tl	567
		\l_siunitx_number_round_mode_tl	567 , 840 , 1020 , 1066
		_siunitx_number_round_none:nnnnnnn	835

_siunitx_number_round_pad:nnn .	_siunitx_option_deprecated:nnn
..... 1097 , 1140 , 1166 10 , 84 , 91 , 98 , 105 , 192 , 199
\l_siunitx_number_round_pad_	_siunitx_option_table_comparator:nnnnnnn
bool 567 , 1102 260
_siunitx_number_round_places:nnnnnnn	_siunitx_option_table_comparator:nnnnnnnn
..... 1147 276
_siunitx_number_round_places_	_siunitx_option_table_figures-decimal:nnnnnnnn
decimal:nn 1147 260
_siunitx_number_round_places_	_siunitx_option_table_figures-exponent:nnnnnnnn
end:nn 1062 , 1147 260
_siunitx_number_round_places_	_siunitx_option_table_figures-integer:nnnnnnnn
integer:nn 1147 260
\l_siunitx_number_round_	_siunitx_option_table_figures-uncertainty:nnnnnnnn
precision_int 567 , 260
1043 , 1110 , 1133 , 1136 , 1141 , 1152 ,	_siunitx_option_table_format:n
1164 , 1167 , 1174 , 1184 , 1192 , 1210	260 , 299 , 301 , 303 , 305 , 307 , 309 , 311
_siunitx_number_round_scientific:nn	_siunitx_option_table_sign-exponent:nnnnnnnn
..... 1055 260
_siunitx_number_round_scientific:c:nn	_siunitx_option_table_sign-mantissa:nnnnnnnn
..... 847 260
_siunitx_number_round_truncate:n	_siunitx_print_match:n 88
..... 847	_siunitx_print_math:n 91 , 108
_siunitx_number_round_truncate:nnN	_siunitx_print_math_aux:Nn .. 108
..... 847	_siunitx_print_math_auxi:n .. 108
_siunitx_number_round_truncate_	_siunitx_print_math_auxii:n .. 108
direct:n 847 , 1225	_siunitx_print_math_auxiii:n .. 108
_siunitx_number_round_uncertainty:nnn	_siunitx_print_math_auxiv:n .. 108
..... 1188	_siunitx_print_math_auxv:n .. 108
_siunitx_number_round_uncertainty:nnnnn	\l_siunitx_print_math_family_
..... 1188	bool 34 , 144
_siunitx_number_round_uncertainty:nnnnnn	_siunitx_print_math_font_bool
..... 1188 34 , 159
\l_siunitx_number_tight_bool ...	_siunitx_print_math_script:n . 108
..... 1247 , 1354 , 1546	_siunitx_print_math_sub:n ... 108
\l_siunitx_number_tmp_tl 7 ,	_siunitx_print_math_super:n .. 108
114 , 117 , 234 , 239 , 245 , 246 , 254 , 255	_siunitx_print_math_text:n .. 108
\l_siunitx_number_uncert_	_siunitx_print_math_version:nn 108
separate_bool 1247 , 1477	\l_siunitx_print_math_version_
\l_siunitx_number_uncert_	bool 34 , 121
separator_tl 1247 , 1487	\l_siunitx_print_math_weight_
\l_siunitx_number_unity_	bool 34 , 110
mantissa_bool ... 1247 , 1382 , 1543	\c_siunitx_print_mathrm_int 14 , 175
\l_siunitx_number_valid_tl .. 1569	\c_siunitx_print_mathsf_int 14 , 163
\l_siunitx_number_validate_bool	\c_siunitx_print_mathtt_int 14 , 164
..... 75 , 125 , 1574	\l_siunitx_print_number_color_
_siunitx_number_zero_decimal:NN	tl 34
..... 609 , 1228	\l_siunitx_print_number_mode_tl 34
_siunitx_number_zero_decimal:nnnnnnn	_siunitx_print_replace_font:N .
..... 1228 99 , 184 , 206 , 234
\l_siunitx_number_zero_exponent_	_siunitx_print_store_fam:n ... 14
bool 1247 , 1538	_siunitx_print_text:n 92 , 211
_siunitx_option_deprecated:nn .	\l_siunitx_print_text_family_
..... 10 , 60 , 73 , 113 ,	bool 55 , 215
119 , 125 , 131 , 137 , 151 , 157 , 168 , 183	\l_siunitx_print_text_family_tl 34

_siunitx_print_text_replace:N	211	_siunitx_table_align_auxii:nn	183
_siunitx_print_text_replace:n	211	\l_siunitx_table_align_before_-	
_siunitx_print_text_replace:NnN		bool	419 , 545 , 590
.....	211	_siunitx_table_align_center:n	183
_siunitx_print_text_scripts: .	211	\l_siunitx_table_align_comparator_-	
_siunitx_print_text_scripts:NnN		bool	419 , 575
.....	211	\l_siunitx_table_align_exponent_-	
_siunitx_print_text_scripts_-		bool	419 , 668
one:Nn	273 , 280 , 311	_siunitx_table_align_left:n	183
_siunitx_print_text_scripts_-		\l_siunitx_table_align_mode_tl	
one:NnN	211	257 , 332 , 336 , 433
_siunitx_print_text_scripts_-		\l_siunitx_table_align_number_-	
two:n	211	tl	257 , 408 , 531 , 715
_siunitx_print_text_scripts_-		_siunitx_table_align_right:n	183
two:nn	211	\l_siunitx_table_align_text_tl	
_siunitx_print_text_scripts_-		216 , 226
two:NnNn	211	\l_siunitx_table_align_uncertainty_-	
\l_siunitx_print_text_series_-		bool	419 , 657
bool	57 , 217	\l_siunitx_table_auto_round_-	
\l_siunitx_print_text_series_tl	34	bool	257 , 500
\l_siunitx_print_text_shape_-		\l_siunitx_table_before_box	
bool	59 , 219	416 , 437 , 438 , 440 , 446 , 448 , 452 , 457 , 463 , 486 , 487 , 489 , 492 , 496 , 533 , 557 , 559 , 565 , 611 , 613 , 620
\l_siunitx_print_text_shape_tl	34	\l_siunitx_table_before_model_-	
_siunitx_print_text_sub:n	211	tl	257 , 269 , 282 , 485
_siunitx_print_text_super:n	211	\l_siunitx_table_before_tl	
\l_siunitx_print_tmp_box	12 , 24	101 , 110 , 114 , 117
\l_siunitx_print_tmp_tl	12 , 113 , 114 , 183 , 184 , 185 , 188 , 191 , 205 , 206 , 207 , 227 , 228 , 229 , 283 , 284 , 286	\l_siunitx_table_carry_dim	
\l_siunitx_print_unit_color_tl	34	418 , 524 , 527 , 631 , 686 , 694 , 706
\l_siunitx_print_unit_mode_tl	34	_siunitx_table_center_marker:	
_siunitx_symbol_if_replace:Nn	45	235 , 364 , 479
_siunitx_symbol_if_replace:NnTF		_siunitx_table_cleanup_-	
.....	45 , 72 , 84 , 105	decimal:w	232 , 379 , 478
_siunitx_symbol_non_latin:n		_siunitx_table_collect_begin:	
.....	24 , 48 , 80 , 93 , 114	11 , 24
_siunitx_symbol_non_latin:nnnn	24	_siunitx_table_collect_begin:N	80
_siunitx_symbol_text_only:n		_siunitx_table_collect_begin:w	24
.....	63 , 79 , 92 , 109	_siunitx_table_collect_end:	19 , 104
_siunitx_symbol_textmu:	5 , 115	_siunitx_table_collect_group:n	38
_siunitx_symbol_texttimes:	120	_siunitx_table_collect_loop:	
\l_siunitx_symbol_tmpa_tl		30 , 37 , 38
..	3 , 48 , 49 , 50 , 53 , 134 , 136 , 137 , 139	_siunitx_table_collect_-	
\l_siunitx_symbol_tmpb_tl		search:NnTF	38
.....	3 , 52 , 53 , 138 , 139	_siunitx_table_collect_search_-	
\l_siunitx_table_after_box	416 , 443 , 445 , 449 , 456 , 459 , 470 , 523 , 536	aux:NnN	38
\l_siunitx_table_after_model_tl		\l_siunitx_table_collect_tl	
.....	250 , 271 , 284 , 522	23 , 26 , 46 , 60 , 81 , 106 , 107 , 109
\l_siunitx_table_after_tl		_siunitx_table_collect_token:N	38
.....	101 , 112 , 119 , 147	\l_siunitx_table_column_width_-	
\l_siunitx_table_align_after_-		dim	176 , 192
bool	419 , 526	\l_siunitx_table_decimal_box	
_siunitx_table_align_auxi:nn	183	.	229 , 239 , 241 , 244 , 250 , 341 , 356 ,

366, 380, 398, 399, 411, 477, 481,
 535, 630, 634, 683, 685, 690, 701, 703
 _siunitx_table_direct_begin: ..
 12, 323
 _siunitx_table_direct_begin:w 323
 _siunitx_table_direct_end: 20, 323
 _siunitx_table_direct_format: 323
 _siunitx_table_direct_format:nnnnnn
 323
 _siunitx_table_direct_format:w 323
 _siunitx_table_direct_format_
 aux:w 372, 375
 _siunitx_table_direct_format_
 end: 323
 _siunitx_table_direct_format_
 switch: 323
 _siunitx_table_direct_marker: 323
 _siunitx_table_direct_marker_
 end: 323
 _siunitx_table_direct_marker_
 switch: 323
 _siunitx_table_direct_none: .. 323
 _siunitx_table_direct_none_
 end: 323
 _siunitx_table_fil: .. 231, 245,
 252, 343, 382, 392, 406, 451, 460,
 495, 529, 550, 564, 585, 598, 619, 691
 \l_siunitx_table_fixed_width_
 bool 176, 191
 \l_siunitx_table_format_tl 265,
 269, 274, 281, 290, 292, 293, 296, 508
 _siunitx_table_generate_
 model:n 272, 285
 _siunitx_table_generate_
 model:nnnnnn 273, 285
 _siunitx_table_generate_model_
 S:nw 285
 \l_siunitx_table_integer_box ...
 229, 238, 242, 249,
 253, 344, 365, 384, 410, 475, 480,
 534, 547, 556, 561, 576, 578, 580,
 584, 592, 594, 599, 605, 606, 608, 616
 \l_siunitx_table_model_tl
 270, 272, 281, 301, 371, 516
 \l_siunitx_table_number_tl
 101, 111, 113, 118
 _siunitx_table_print:nnn . 116, 432
 _siunitx_table_print_format:nnn
 432
 _siunitx_table_print_format:nnnnn
 507, 539
 _siunitx_table_print_format_
 after:N 432
 _siunitx_table_print_format_
 auxi:w 520, 541
 _siunitx_table_print_format_
 auxii:w 568, 570
 _siunitx_table_print_format_
 auxiii:w 624, 626
 _siunitx_table_print_format_
 auxiv:w 636, 638
 _siunitx_table_print_format_
 auxv:w 642, 646
 _siunitx_table_print_format_
 auxvi:w 643, 650
 _siunitx_table_print_format_
 auxvii:w 649, 658, 660
 _siunitx_table_print_format_
 box:Nn 432
 _siunitx_table_print_marker:nnn
 432
 _siunitx_table_print_marker:w 432
 _siunitx_table_print_marker_
 auxi:w 432
 _siunitx_table_print_marker_
 auxii:w 432
 _siunitx_table_print_marker_
 auxiii:w 432
 _siunitx_table_print_none:nnn 432
 _siunitx_table_print_text:n ...
 114, 223, 329
 _siunitx_table_skip:n
 171, 195, 197, 209, 211
 _siunitx_table_split:nNNN
 108, 122, 268
 _siunitx_table_split_group:NNNn
 122
 _siunitx_table_split_loop:NNN 122
 _siunitx_table_split_tidy:N ...
 128, 129, 160
 _siunitx_table_split_tidy:Nn . 160
 _siunitx_table_split_token:NNNN
 122
 \l_siunitx_table_text_bool
 6, 9, 16, 225
 \l_siunitx_table_tmp_box 3, 339,
 342, 378, 381, 383, 385, 485, 493,
 522, 524, 544, 548, 560, 573, 581,
 595, 614, 629, 633, 654, 655, 656,
 665, 666, 667, 687, 692, 697, 704, 709
 \l_siunitx_table_tmp_dim
 ... 3, 85, 604, 615, 655, 666, 696, 708
 \l_siunitx_table_tmp_tl 3,
 370, 373, 464, 465, 466, 467, 469,
 498, 511, 513, 514, 518, 521, 718, 719
 _siunitx_tmp:w 120, 122

\l__siunitx_tmp_tl	__siunitx_unit_format_output: ..
..... 26, 27, 37, <u>38</u> , 38, 41, 499, <u>638</u>
42, 44, 45, 61, 62, 64, 65, 81, 82, 90, 91	__siunitx_unit_format_output_-
\l__siunitx_unit_autofrac_bool ..	aux: <u>638</u>
416, 423, 430, 437, 444, 451, <u>470</u> , 717	__siunitx_unit_format_output_-
\l__siunitx_unit_bracket_bool ...	aux:nn <u>638</u>
..... 468, 492, 513, 557, 641, 662	__siunitx_unit_format_output_-
\l__siunitx_unit_bracket_close_-	denominator: <u>638</u>
tl <u>403</u> , 517, 599	__siunitx_unit_format_parsed: ..
\l__siunitx_unit_bracket_open_tl 128, <u>482</u>
..... <u>403</u> , 515, 596	__siunitx_unit_format_parsed_-
\l__siunitx_unit_current_tl	aux:n <u>482</u>
..... 478, 493, 552, 554,	__siunitx_unit_format_power: .. <u>521</u>
581, 589, 627, 630, 636, 688, 696, 699	__siunitx_unit_format_power_-
\l__siunitx_unit_denominator_-	aux:wTF <u>521</u>
bracket_bool <u>403</u> , 660	__siunitx_unit_format_power_-
\l__siunitx_unit_denominator_tl .	negative: <u>521</u>
480, 485, 661, 706, 747, 756, 765, 772	__siunitx_unit_format_power_-
\l__siunitx_unit_font_bool	negative_aux:w <u>521</u>
..... 469, 494, 694, 701	__siunitx_unit_format_power_-
\l__siunitx_unit_forbid_literal_-	positive: <u>521</u>
bool 131, <u>403</u>	__siunitx_unit_format_power_-
__siunitx_unit_format:nNN <u>105</u>	superscript: <u>521</u>
__siunitx_unit_format_aux: ... <u>105</u>	__siunitx_unit_format_prefix: . <u>559</u>
__siunitx_unit_format_bracket:N	__siunitx_unit_format_prefix_-
..... 511, 554, 756	power: <u>559</u>
__siunitx_unit_format_finalise:	__siunitx_unit_format_prefix_-
..... 501, <u>704</u>	symbol: <u>559</u>
__siunitx_unit_format_finalise_-	__siunitx_unit_format_qualifier:
autofrac: <u>704</u> <u>582</u>
__siunitx_unit_format_finalise_-	__siunitx_unit_format_qualifier_-
fraction: 722, 728, 741	bracket: <u>582</u>
__siunitx_unit_format_finalise_-	__siunitx_unit_format_qualifier_-
fractional: <u>704</u>	combine: <u>582</u>
__siunitx_unit_format_finalise_-	__siunitx_unit_format_qualifier_-
power: <u>704</u>	phrase: <u>582</u>
__siunitx_unit_format_finalise_-	__siunitx_unit_format_qualifier_-
symbol: 721, 731, 750	subscript: <u>582</u>
__siunitx_unit_format_font: ...	__siunitx_unit_format_special: <u>625</u>
..... 523, 593, 605, 615, 640, <u>692</u>	__siunitx_unit_format_unit: .. <u>633</u>
__siunitx_unit_format_literal:n	\l__siunitx_unit_formatted_tl ...
..... 133, 136, <u>150</u> <u>104</u> , <u>112</u> , 120, 140, 167, 175,
__siunitx_unit_format_literal_-	176, 222, 225, 227, 486, 669, 670,
auxi:w <u>150</u>	715, 716, 730, 732, 736, 737, 738,
__siunitx_unit_format_literal_-	743, 746, 752, 754, 761, 764, 768, 770
auxii:n 183, 187	\l__siunitx_unit_fraction_-
__siunitx_unit_format_literal_-	function_tl <u>403</u> , 745
auxii:w <u>150</u>	__siunitx_unit_if_symbolic:n ... 11
__siunitx_unit_format_literal_-	__siunitx_unit_if_symbolic:nTF .
auxiii:w <u>150</u> 11, 73, 124
__siunitx_unit_format_literal_-	__siunitx_unit_literal_power:nN
auxiv:w <u>150</u> 42, 92, <u>148</u>
__siunitx_unit_format_literal_-	__siunitx_unit_literal_special:nN
auxv:w <u>150</u> 83, 86, <u>149</u>

<code>\c_siunitx_unit_math_subscript_-</code>	<code>\l_siunitx_unit_prefix_power_-</code>
<code>tl</code> 7 , 162 ,	<code>bool</code> 107 , 112 , 476 , 561
206 , 207 , 210 , 211 , 215 , 216 , 218 ,	<code>\l_siunitx_unit_prefixes_-</code>
219 , 242 , 618 , 871 , 877 , 888 , 894 , 901	<code>forward_prop</code> 48 , 567
<code>_siunitx_unit_non_latin:n</code>	<code>\l_siunitx_unit_prefixes_-</code>
..... 776 , 813 , 841 , 908	<code>reverse_prop</code> 48
<code>_siunitx_unit_non_latin:nmmn</code> . 776	<code>\l_siunitx_unit_product_tl</code>
<code>\l_siunitx_unit_numerator_bool</code> 97 , 184 , 653 , 664 , 771
..... 116 , 474 , 495 , 535 , 645	<code>\l_siunitx_unit_qualifier_mode_-</code>
<code>_siunitx_unit_parse:n</code> 126 , 260	<code>tl</code> 403 , 475 , 587
<code>_siunitx_unit_parse_add:nmmn</code> ..	<code>\l_siunitx_unit_qualifier_-</code>
272 , 289 , 303 , 321 , 331 , 338 , 343 , 357	<code>phrase_tl</code> 403 , 608
<code>\l_siunitx_unit_parse_bool</code> 122 , 403	<code>\l_siunitx_unit_separator_tl</code> .. 150
<code>_siunitx_unit_parse_finalise:</code> .	<code>_siunitx_unit_set_symbolic:Nmmn</code>
..... 270 , 393 22 , 41 , 44 , 50 , 64 , 70 , 79
<code>_siunitx_unit_parse_finalise:n</code>	<code>_siunitx_unit_set_symbolic:Nmmn</code>
..... 269 , 362 22
<code>_siunitx_unit_parse_per:</code> .. 81 , 348	<code>_siunitx_unit_set_symbolic:Npmn</code>
<code>_siunitx_unit_parse_power:nmmn</code> 22 , 82 , 85 , 88 , 91 , 94
..... 43 , 46 , 93 , 96 , 286	<code>\l_siunitx_unit_sticky_per_bool</code>
<code>_siunitx_unit_parse_prefix:Nn</code> 108 , 253 , 350
..... 52 , 286	<code>\l_siunitx_unit_test_bool</code>
<code>_siunitx_unit_parse_qualifier:nmmn</code> 10 , 14 , 31 , 265
..... 66 , 90 , 286	<code>\l_siunitx_unit_tmp_fp</code> 4 , 108
<code>_siunitx_unit_parse_special:n</code> .	<code>\l_siunitx_unit_tmp_int</code> . 4 , 288 , 290
..... 84 , 87 , 286	<code>\l_siunitx_unit_tmp_tl</code>
<code>_siunitx_unit_parse_unit:Nn</code> 75 , 335 4 , 15 , 17 , 156 , 157 , 159 , 161 ,
<code>\l_siunitx_unit_parsed_prop</code> ...	165 , 166 , 168 , 171 , 274 , 276 , 283 ,
..... 105 , 127 ,	294 , 300 , 317 , 319 , 364 , 365 , 368 ,
257 , 262 , 275 , 282 , 300 , 319 , 365 ,	369 , 372 , 380 , 384 , 389 , 397 , 399 ,
367 , 371 , 379 , 383 , 388 , 399 , 507 , 572	505 , 508 , 570 , 573 , 574 , 576 , 730 , 735
<code>\l_siunitx_unit_parsing_bool</code> ...	<code>\l_siunitx_unit_total_int</code>
..... 9 , 33 , 155 , 263 481 , 484 , 490
<code>\l_siunitx_unit_part_tl</code> 373 , 375 ,	<code>\l_siunitx_unit_two_part_bool</code> ..
376 , 377 , 384 , 478 , 508 , 525 , 538 ,	419 , 426 , 433 , 440 , 447 , 454 , 470 , 657
541 , 543 , 555 , 568 , 576 , 581 , 589 ,	skip commands:
594 , 598 , 606 , 610 , 616 , 621 , 629 , 636	<code>\skip_horizontal:n</code> 173 , 527
<code>\l_siunitx_unit_per_bool</code>	<code>\c_zero_skip</code> 174
..... 108 , 257 , 264 , 341 , 352	<code>\sp</code> 114
<code>\l_siunitx_unit_per_symbol_bool</code>	<code>\SplitArgument</code> 68
..... 417 , 424 ,	<code>\square</code> 94 , 928
431 , 438 , 445 , 452 , 470 , 668 , 674 , 720	<code>\squared</code> 95 , 880 , 928
<code>\l_siunitx_unit_per_symbol_tl</code> ..	<code>\steradian</code> 93 , 840
..... 403 , 755	<code>sticky-per</code> 97
<code>\l_siunitx_unit_position_int</code> ...	<code>\stilb</code> 94 , 917
..... 112 , 257 , 266 , 268 , 288 , 295 ,	<code>\stokes</code> 94 , 917
306 , 318 , 322 , 332 , 337 , 339 , 344 ,	str commands:
358 , 398 , 484 , 488 , 490 , 496 , 506 , 571	<code>\str_case_x:nmmTF</code> 146
<code>\l_siunitx_unit_powers_positive_-</code>	<code>\str_if_eq:nmmTF</code> . 53 , 71 , 127 , 167 ,
<code>bool</code> 418 ,	199 , 218 , 238 , 299 , 543 , 635 , 657 ,
425 , 432 , 439 , 446 , 453 , 470 , 536 , 708	674 , 946 , 1020 , 1066 , 1341 , 1391 , 1617
<code>\l_siunitx_unit_prefix_fp</code>	<code>\str_if_eq_p:nmm</code> 285 ,
..... 121 , 142 , 477 , 487 , 575	441 , 504 , 616 , 1379 , 1381 , 1539 , 1542
	<code>\str_if_eq_x:nmmTF</code> 1195 , 1239

<code>\str_if_eq_x_p:nn</code>	618	145, 167, 221, 267, 279, 289, 315, 367, 476, 485, 486, 493, 543, 544, 564
sys commands:		
<code>\sys_if_engine luatex_p:</code>		<code>\tl_const:Nn</code>
..... 6, 25, 89, 112, 777		7, 330
<code>\sys_if_engine xetex_p:</code>		<code>\tl_count:n</code>
..... 7, 26, 90, 113, 778		142, 149, 619, 655, 665, 667, 812, 813, 1031, 1164, 1167, 1174, 1184, 1210, 1219, 1224, 1241, 1405, 1416, 1483
T		
<code>\tablenum</code>	94	<code>\tl_head:n</code>
<code>\tabularnewline</code>	54, 56, 75, 77	181, 235, 309, 911, 929, 1082, 1195, 1469
<code>\tera</code>	13, 58, 93, 817	<code>\tl_head:w</code>
<code>\tesla</code>	93, 840	768, 787
<code>\TeV</code>	43, 127	<code>\tl_if_blank:nTF</code>
TeX and L ^A T _E X 2 _ε commands:		... 3, 17, 135, 151, 172, 179, 206, 232, 238, 240, 266, 288, 307, 480, 551, 641, 663, 675, 697, 709, 718, 765, 776, 784, 866, 976, 1108, 1149, 1216, 1329, 1335, 1389, 1466, 1557
<code>\@ifpackagelater</code>	4	<code>\tl_if_blank_p:n</code>
<code>\@ifpackageloaded</code> 7, 23, 66, 74, 135, 202		... 4, 106, 110, 185, 186, 1191, 1380
<code>\@maybe@unskip</code>	72	<code>\tl_if_empty:N</code> 75, 94, 98, 101, 102, 102, 113, 118, 123, 139, 147, 152, 162, 168, 222, 226, 227, 230, 270, 278, 293, 349, 468, 521, 563, 604, 686, 706, 715, 761, 1314, 1343, 1577
<code>\@temptokena</code>	125, 127	<code>\tl_if_empty:nTF</code>
<code>\AtBeginDocument</code>	59	530
<code>\f@family</code>	146	<code>\tl_if_empty_p:N</code>
<code>\f@series</code>	113	429, 661, 669
<code>\m@th</code>	297, 331	<code>\tl_if_eq:NNTF</code>
<code>\math@version</code>	127	139
<code>\NC@do</code>	120, 121	<code>\tl_if_in:NnTF</code>
<code>\NC@find</code>	130	5, 57, 116, 216, 275, 306, 381, 387, 399, 402, 409, 414, 484, 501, 516, 553
<code>\NC@list</code>	4, 121, 122	<code>\tl_if_novalue:nTF</code>
<code>\protected@edef</code> 15, 49, 86, 103, 106, 165		108, 111
<code>\sf@size</code>	310	<code>\tl_map_inline:Nn</code>
<code>\tab@setcr</code>	73	236, 346, 387
<code>\z@</code>	310	<code>\tl_map_inline:nn</code>
tex commands:		55
<code>\tex_cr:D</code>	31	<code>\tl_new:N</code>
<code>\tex_hfil:D</code>	231	3, 3, 4, 5, 6, 7, 8, 9, 13, 23, 29, 34, 35, 38, 67, 68, 69, 70, 71, 72, 73, 74, 80, 81, 82, 83, 101, 102, 103, 104, 222, 252, 279, 280, 281, 282, 283, 284, 342, 343, 475, 478, 479, 480, 600, 601, 1246, 1569
<code>\tex_kern:D</code>	174	<code>\tl_put_right:Nn</code>
<code>\tex_the:D</code>	122	... 14, 46, 58, 60, 81, 140, 141, 150, 153, 154, 157, 308, 378, 389, 431, 443, 470, 486, 506, 522, 589, 635
<code>\text</code>	56, 131, 213	<code>\tl_replace_all:Nnn</code>
text-family-to-math	58	... 3, 6, 11, 11, 103, 157, 159, 161, 185, 188, 238, 253, 327, 343
text-weight-to-math	58	<code>\tl_reverse:n</code> ..
<code>\textcolor</code>	56, 57, 57, 58, 77, 86, 87, 90, 95, 1361	704, 1003, 1004, 1010
<code>\textminus</code>	56, 58, 241	<code>\tl_set:Nn</code>
<code>\textmu</code>	16, 21	21, 24, 26, 32, 48, 54, 98, 103, 104, 114, 118, 120, 134, 136, 138, 139, 139, 154, 156, 156, 168, 175, 179, 183, 196, 205, 218, 219, 220, 222, 227, 229, 234, 250, 257, 263, 265, 265, 273, 274, 283, 286, 287, 292, 294, 298, 301, 317, 319, 320, 334,
<code>\textpm</code>	56, 237	
<code>\textsubscript</code>	56, 259, 290	
<code>\textsuperscript</code>	56, 264	
<code>\texttimes</code>	127, 132	
<code>\the</code>	127	
<code>\THz</code>	8, 126	
tight-spacing	15	
<code>\times</code>	138, 145, 1642	
tl commands:		
<code>\c_empty_tl</code>	48, 49, 1441	
<code>\tl_clear:N</code> ... 26, 80, 85, 94, 119, 120, 121, 124, 125, 126, 142, 144,		

335, 351, 364, 369, 370, 375, 397, 430, 462, 466, 469, 505, 513, 514, 524, 538, 552, 555, 559, 560, 562, 570, 574, 594, 606, 616, 627, 628, 637, 670, 683, 696, 716, 730, 732, 743, 752, 768, 793, 819, 829, 837, 1232	\upOmega 98, 99
\tl_set_eq:NN 47, 69, 92, 152, 184, 263, 277, 460, 577, 581, 593, 763	\upshape 56, 57, 220
\tl_tail:n 769, 788	\us 84, 126
\tl_use:N 82, 191, 207, 229	use commands:
\l_tmpa_tl 90, 91	\use:N 28, 79, 226, 309, 332, 336, 408, 433, 509, 531, 584, 642, 715, 741, 1022, 1407, 1413, 1469
token commands:	\use:n 41, 63, 67, 76, 105, 132, 166, 167, 168, 176, 193, 199, 241, 250, 311, 1393
\l_peek_token 270	\use_i:nnnn 115
\token_if_eq_charcode_p:NN 557	\use_i_delimit_by_q_recursion_- stop:nw 1050, 1095, 1619, 1621
\token_if_eq_meaning:NNTF 97	\use_i_delimit_by_q_stop:nw 98
\token_to_str:N 29, 119, 160, 162, 279, 312, 326, 344, 943, 946	\use_ii:nn 1199
\tonne 93, 851	\use_iv:nnnn 107, 111
\tothe 91, 95	\use_none:n 480, 1084, 1451
track-explicit-plus 15	\use_none:nn 1455
\ttdefault 150	\uV 21, 127
	\uW 43, 127
	V
U	\V 21, 127
\uA 2, 126	\volt 21, 22, 23, 24, 25, 26, 93, 840
\uG 35, 126	
\uJ 43, 127	W
\uL 27, 127	\W 43, 127
\uI 27, 127	\watt 43, 44, 45, 46, 47, 48, 59, 93, 840
\uM 60, 126	\weber 93, 840
\uMol 14, 127	
uncertainty-separator 15	Y
\unit 68, 132, 352	\yocto 93, 806
unit-close-bracket 97	\yotta 93, 817
unit-color 58	
unit-mode 58	Z
unit-open-bracket 97	\zepto 93, 806
\unskip 53, 74	\zetta 93, 817