

siunitx – A comprehensive (SI) units package*

Joseph Wright[†]

Released 2021-04-16

Contents

I	siunitx – Overall set up	1
1	siunitx implementation	1
1.1	Initial set up	1
1.2	Safety checks	1
1.3	Provide a kernel command	2
1.4	Top-level scratch space	2
1.5	Load time options	2
1.6	Option handling	3
1.7	User interfaces	3
1.7.1	Preamble commands	3
1.7.2	Document commands	3
1.8	“Glue” commands	6
1.9	Table column	7
1.10	Document commands in bookmarks	8
II	siunitx-angle – Formatting angles	11
1	Formatting angles	11
1.1	Key-value options	11
2	siunitx-angle implementation	12
III	siunitx-compound – Compound numbers and quantities	19
1	siunitx-compound implementation	21
1.1	General mechanism	21
1.2	Lists	29
1.3	Products	30
1.4	Ranges	31
1.5	Standard settings for module options	32

*This file describes v3.0.0-beta, last revised 2021-04-16.

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

IV	siunitx-locale – Localisation	34
1	siunitx-locale implementation	34
1.1	Locales	34
1.2	Localisation	35
V	siunitx-number – Parsing and formatting numbers	36
1	Formatting numbers	36
1.1	Key-value options	38
2	siunitx-number implementation	40
2.1	Initial set-up	41
2.2	Main formatting routine	41
2.3	Parsing numbers	42
2.4	Processing numbers	56
2.5	Number modification	74
2.6	Outputting parsed numbers	74
2.7	Miscellaneous tools	82
2.8	Messages	84
2.9	Standard settings for module options	84
VI	siunitx-print – Printing material with font control	86
1	Printing quantities	86
1.1	Key-value options	86
2	siunitx-print implementation	88
2.1	Initial set up	89
2.2	Printing routines	90
2.3	Standard settings for module options	97
VII	siunitx-quantity – Quantities	99
1	siunitx-quantity implementation	99
1.1	Initial set-up	100
1.2	Main formatting routine	100
1.3	Standard settings for module options	103
1.4	Adjustments to units	103
VIII	siunitx-symbol – Symbol-related settings	105
1	siunitx-symbol implementation	105
1.1	Bookmark definitions	108
IX	siunitx-table – Formatting numbers in tables	110

1	Numbers in tables	110
1.1	Key-value options	110
2	siunitx-table implementation	111
2.1	Interface functions	111
2.2	Collecting tokens	112
2.3	Separating collected material	114
2.4	Printing numbers in cells: spacing	115
2.5	Printing just text	117
2.6	Number alignment: core ideas	117
2.7	Directly printing without collection	120
2.8	Printing numbers in cells: main functions	122
2.9	Standard settings for module options	129
X	siunitx-unit – Parsing and formatting units	130
1	Formatting units	130
2	Defining symbolic units	132
3	Per-unit options	133
4	Units in (PDF) strings	133
5	Pre-defined symbolic unit components	133
5.1	Key-value options	136
6	siunitx-unit implementation	138
6.1	Initial set up	138
6.2	Defining symbolic unit	139
6.3	Applying unit options	141
6.4	Non-standard symbolic units	142
6.5	Main formatting routine	143
6.6	Formatting literal units	145
6.7	(PDF) String creation	148
6.8	Parsing symbolic units	148
6.9	Formatting parsed units	153
6.10	Non-Latin character support	165
6.11	Pre-defined unit components	166
6.12	Messages	168
6.13	Standard settings for module options	169
XI	siunitx-abbreviations – Abbreviations	170
1	siunitx-abbreviation implementation	172
XII	siunitx-binary – Binary units	176

1	siunitx-binary implementation	176
XIII	siunitx-command – Units as document command	177
1	Creating units as document commands	177
1.1	Key–value options	177
2	siunitx-command implementation	177
2.1	Options	178
2.2	Creation of unit document commands	178
2.3	Standard settings for module options	180
XIV	siunitx-emulation – Emulation	181
1	siunitx-emulation implementation	181
1.1	Load-time option	182
1.2	Angle options	182
1.3	Combination functions options	182
1.4	Command options	183
1.5	Print options	183
1.6	Symbol options	184
1.7	Number options	184
1.7.1	Table options	186
1.8	Unit options	189
1.9	Quantity units	190
1.10	Preamble commands	191
1.11	Document commands	191
	Index	193

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

```
3 <*init>
    Set up a couple of commands in recent-ish LATEX 2ε releases.
4 \providecommand\DeclareRelease[3]{}
5 \providecommand\DeclareCurrentRelease[2]{}
    Allow rollback to version 2: if we need to, version 1 could eventually be added here
    too.
```

```
6 \DeclareRelease{2}{2021-04-09}{siunitx-v2.sty}
7 \DeclareRelease{v2}{2021-04-09}{siunitx-v2.sty}
8 \DeclareRelease{2.8d}{2021-04-09}{siunitx-v2.sty}
9 \DeclareRelease{v2.8d}{2021-04-09}{siunitx-v2.sty}
10 \DeclareCurrentRelease{3.0.0-beta}{2021-04-16}
```

```
    Load only the essential support (expl3) “up-front”, and only if required.
11 \@ifundefined{ExplFileName}
12   {\RequirePackage{expl3}}
13   {}
```

Make sure that the version of l3kernel in use is sufficiently new. We use \ExplFileName as \ifpackagelater doesn’t work for pre-loaded expl3 in the absence of the package.

```
14 \@ifl@t@r\ExplFileName{2018-06-01}
15   {}
16   {%
17     \PackageError{siunitx}{Support package expl3 too old}
18     {%
19       You need to update your installation of the bundles ‘l3kernel’ and
20       ‘l3packages’. \MessageBreak
21       Loading~siunitx~will~abort!%
22     }%
23     \endinput
24   }%
```

Identify the package and give the over all version information.

```
25 \ProvidesExplPackage {siunitx} {2021-04-16} {3.0.0-beta}
26   {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.

```

27 \msg_new:nnnn { siunitx } { incompatible-package }
28   { Package~'#1'~incompatible. }
29   { The~#1~package~and~siunitx~are~incompatible. }
30 \cs_new_protected:Npn \__siunitx_load_check:n #1
31   {
32     \@ifpackageloaded {#1}
33       { \msg_error:nxx { siunitx } { incompatible-package } {#1} }
34       { }
35   }
36 \clist_map_function:nN
37   { SIunits , sistyle , unitsdef , fancyunits }
38   \__siunitx_load_check:n
39 \AtBeginDocument
40   {
41     \clist_map_function:nN { SIunits , sistyle }
42     \__siunitx_load_check:n
43   }

```

(End definition for __siunitx_load_check:.)

1.3 Provide a kernel command

`\IfFormatAtLeastTF` Not present in older kernels: use the L^AT_EX 2_ε mechanism as this is correct for this case.

```

44 \providecommand \IfFormatAtLeastTF { \@ifl@t@r \fmtversion }

```

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

1.4 Top-level scratch space

`\l__siunitx_tmp_tl` Scratch space for the interfaces.

```

45 \tl_new:N \l__siunitx_tmp_tl

```

(End definition for \l__siunitx_tmp_tl.)

```

46 \</init>

```

```

47 \<*options>

```

1.5 Load time options

`\l__siunitx_column_type_tl`

```

48 \keys_define:nn { siunitx }
49   {
50     table-column-type .tl_set:N =
51       \l__siunitx_column_type_tl
52   }
53 \keys_set:nn { siunitx }
54   {
55     table-column-type = S
56   }

```

(End definition for \l__siunitx_column_type_tl.)

1.6 Option handling

```

57 \RequirePackage { l3keys2e }
58 \ProcessKeysOptions { siunitx }
59 \</options>

```

1.7 User interfaces

```

60 \<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 re-arranged by DocStrip but there is no advantage.

User level interfaces are all created by xparse

```

61 \IfFormatAtLeastTF { 2020-10-01 }
62 { }
63 { \RequirePackage { xparse } }

```

1.7.1 Preamble commands

<pre> \DeclareSIPower \DeclareSIPrefix \DeclareSIQualifier \DeclareSIUnit </pre>	<pre> Pass data to the code layer. 64 \NewDocumentCommand \DeclareSIPower { +m +m m } 65 { 66 \siunitx_declare_power:Nnn #1 #2 {#3} 67 } 68 \NewDocumentCommand \DeclareSIPrefix { +m m m } 69 { 70 \siunitx_declare_prefix:Nnn #1 {#2} {#3} 71 } 72 \NewDocumentCommand \DeclareSIQualifier { +m m m } 73 { 74 \siunitx_declare_qualifier:Nn #1 {#2} 75 } 76 \NewDocumentCommand \DeclareSIUnit { o +m m m } 77 { 78 \IfNoValueTF {#1} 79 { \siunitx_declare_unit:Nn #2 {#3} } 80 { \siunitx_declare_unit:Nnn #2 {#3} {#1} } 81 } </pre>
--	---

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

1.7.2 Document commands

```

\qty
82 \@ifpackageloaded { physics }
83 {
84   \msg_new:nnn { siunitx } { physics-pkg }
85   {
86     Detected~the~"physics"~package: \\\
87     Omitting~definition~of~\token_to_str:N \qty.
88   }
89   \msg_warning:nn { siunitx } { physics-pkg }
90   \use_none:nnnn
91 }

```

```

92 { }
93 \NewDocumentCommand \qty { 0 { } m } > { \TrimSpaces } m }
94 {
95   \mode_leave_vertical:
96   \group_begin:
97     \siunitx_unit_options_apply:n {#3}
98     \keys_set:nn { siunitx } {#1}
99     \siunitx_quantity:nn {#2} {#3}
100   \group_end:
101 }

```

(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
\num formatting, print the result.

```

\unit 102 \NewDocumentCommand \ang { 0 { } } > { \SplitArgument { 2 } { ; } } m }
103 {
104   \mode_leave_vertical:
105   \group_begin:
106     \keys_set:nn { siunitx } {#1}
107     \_siunitx_angle:nnn #2
108   \group_end:
109 }
110 \NewDocumentCommand \num { 0 { } m }
111 {
112   \mode_leave_vertical:
113   \group_begin:
114     \keys_set:nn { siunitx } {#1}
115     \siunitx_number_format:nN {#2} \l__siunitx_tmp_tl
116     \siunitx_print:nV { number } \l__siunitx_tmp_tl
117   \group_end:
118 }
119 \NewDocumentCommand \unit { 0 { } } > { \TrimSpaces } m }
120 {
121   \mode_leave_vertical:
122   \group_begin:
123     \siunitx_unit_options_apply:n {#2}
124     \keys_set:nn { siunitx } {#1}
125     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
126     \siunitx_print:nV { unit } \l__siunitx_tmp_tl
127   \group_end:
128 }

```

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

\qtylist Interfaces for compound values.
\numlist 129 \NewDocumentCommand \qtylist
\qtyproduct 130 { 0 { } } > { \SplitList { ; } } m > { \TrimSpaces } m }
\numproduct 131 {
\qtyrange 132 \mode_leave_vertical:
133 \group_begin:
134 \siunitx_unit_options_apply:n {#3}
135 \keys_set:nn { siunitx } {#1}
136 \siunitx_quantity_list:nn {#2} {#3}


```

137   \group_end:
138 }
139 \NewDocumentCommand \numlist { O { } } > { \SplitList { ; } } m }
140 {
141   \mode_leave_vertical:
142   \group_begin:
143     \keys_set:nn { siunitx } {#1}
144     \siunitx_number_list:nn {#2}
145   \group_end:
146 }
147 \NewDocumentCommand \qtyproduct
148 { O { } } > { \SplitList { x } } m > { \TrimSpaces } m }
149 {
150   \mode_leave_vertical:
151   \group_begin:
152     \siunitx_unit_options_apply:n {#3}
153     \keys_set:nn { siunitx } {#1}
154     \siunitx_quantity_product:nn {#2} {#3}
155   \group_end:
156 }
157 \NewDocumentCommand \numproduct
158 { O { } } > { \SplitList { x } } m > { \TrimSpaces } m }
159 {
160   \mode_leave_vertical:
161   \group_begin:
162     \keys_set:nn { siunitx } {#1}
163     \siunitx_number_product:n {#2}
164   \group_end:
165 }
166 \NewDocumentCommand \qtyrange { O { } } m m > { \TrimSpaces } m }
167 {
168   \mode_leave_vertical:
169   \group_begin:
170     \siunitx_unit_options_apply:n {#4}
171     \keys_set:nn { siunitx } {#1}
172     \siunitx_quantity_range:nnn {#2} {#3} {#4}
173   \group_end:
174 }
175 \NewDocumentCommand \numrange { O { } } m m }
176 {
177   \mode_leave_vertical:
178   \group_begin:
179     \keys_set:nn { siunitx } {#1}
180     \siunitx_number_range:nn {#2} {#3}
181   \group_end:
182 }

```

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

\complexnum Interfaces for complex numbers.

```

\complexqty 183 \NewDocumentCommand \complexnum { O { } } m }
184 {
185   \mode_leave_vertical:
186   \group_begin:

```

```

187     \keys_set:nn { siunitx } {#1}
188     \siunitx_complex_number:n {#2} \l__siunitx_tmp_tl
189   \group_end:
190 }
191 \NewDocumentCommand \complexqty { 0 { } m m }
192 {
193   \mode_leave_vertical:
194   \group_begin:
195     \siunitx_unit_options_apply:n {#3}
196     \keys_set:nn { siunitx } {#1}
197     \siunitx_complex_quantity:nn {#2} {#3}
198   \group_end:
199 }

```

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

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

```

200 \NewDocumentCommand \tablenum { 0 { } m }
201 {
202   \mode_leave_vertical:
203   \group_begin:
204     \keys_set:nn { siunitx } {#1}
205     \siunitx_cell_begin:w
206       \ignorespaces #2
207     \siunitx_cell_end:
208   \group_end:
209 }

```

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

`\sisetup` A very thin wrapper.

```

210 \NewDocumentCommand \sisetup { m }
211 { \keys_set:nn { siunitx } {#1} }

```

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

1.8 “Glue” commands

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

```

212 \cs_new_protected:Npn \__siunitx_angle:nnn #1#2#3
213 {
214   \tl_if_novalue:nTF {#2}
215     { \siunitx_angle:n {#1} }
216     {
217       \tl_if_novalue:nTF {#3}
218         { \siunitx_angle:nnn {#1} {#2} { } }
219         { \siunitx_angle:nnn {#1} {#2} {#3} }
220     }
221 }

```

(End definition for `__siunitx_angle:nnn`.)

1.9 Table column

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

```
222 \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.

```
223 \cs_new_protected:Npn \_siunitx_declare_column:Nnn #1#2#3
224 {
225   \cs_if_exist:cT { NC@find@ #1 }
226   {
227     \cs_undefine:c { NC@find@ #1 }
228     \msg_warning:nnn { siunitx } { column-overwritten } {#1}
229   }
230   \newcolumnntype {#1} { }
231   \cs_set_protected:Npn \_siunitx_tmp:w \NC@do ##1##2 \NC@do #1
232   { \NC@list { \NC@do ##1 \NC@do #1 ##2 } }
233   \exp_after:wN \_siunitx_tmp:w \the \NC@list
234   \exp_args:NNc \renewcommand * { NC@rewrite@ #1 } [ 1 ] [ ]
235   {
236     \@temptokena \expandafter
237     {
238       \the \@temptokena
239       > {#2} c < {#3}
240     }
241     \NC@find
242   }
243 }
244 \msg_new:nnn { siunitx } { column-overwritten }
245 { Tabular~column~type~"#1"~overwritten~with~siunitx~definition. }
```

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

```
246 \AtBeginDocument
247 {
248   \@ifpackageloaded { mdwtab }
249   {
250     \cs_set_protected:Npn \_siunitx_declare_column:Nnn #1#2#3
251     {
252       \cs_if_exist:cT { NC@find@ #1 }
253       {
254         \cs_undefine:c { NC@find@ #1 }
255         \msg_warning:nnn { siunitx } { column-overwritten } {#1}
256       }
257       \newcolumnntype {#1} [ 1 ] [ ]
258       { > {#2} c < {#3} }
259     }
260   }
261 }
262 \tl_map_inline:Nn \l__siunitx_column_type_tl
263 {
```

```

264     \__siunitx_declare_column:Nnn #1
265     {
266         \keys_set:nn { siunitx } {##1}
267         \siunitx_cell_begin:w
268     }
269     { \siunitx_cell_end: }
270 }
271 }

```

(End definition for __siunitx_declare_column:Nnn.)

1.10 Document commands in bookmarks

In bookmarks, the siunitx document commands need to produce simple strings that represent their input as far as possible.

__siunitx_bookmark_cmd:Nn

To keep things fast, expandable versions of the document command are created only once. As here we are at the top-level for internal names, we can use the various parts of siunitx-compound that would otherwise be inaccessible.

```

272 \cs_new_protected:Npn \__siunitx_bookmark_cmd:Nnn #1#2#3
273 {
274     \exp_args:Nc \DeclareExpandableDocumentCommand
275     { \cs_to_str:N #1 \c_space_tl ( pdfstring ~ context ) }
276     {#2} {#3}
277 }
278 \__siunitx_bookmark_cmd:Nnn \qty { o m m } { #2 ~ #3 }
279 \__siunitx_bookmark_cmd:Nnn \ang { m } { \__siunitx_angle:n {#1} }
280 \__siunitx_bookmark_cmd:Nnn \num { o m } { #2 }
281 \__siunitx_bookmark_cmd:Nnn \unit { o m } { #2 }
282 \__siunitx_bookmark_cmd:Nnn \numlist { o m }
283 {
284     \__siunitx_list_use:nnVVV {#2} { }
285     \l_siunitx_list_separator_pair_tl
286     \l_siunitx_list_separator_tl
287     \l_siunitx_list_separator_final_tl
288 }
289 \__siunitx_bookmark_cmd:Nnn \qtylist { o m m }
290 {
291     \__siunitx_list_use:nnVVV {#2} {#3}
292     \l_siunitx_list_separator_pair_tl
293     \l_siunitx_list_separator_tl
294     \l_siunitx_list_separator_final_tl
295 }
296 \__siunitx_bookmark_cmd:Nnn \numproduct { o m } { }
297 \__siunitx_bookmark_cmd:Nnn \qtyproduct { o m m } { }
298 \__siunitx_bookmark_cmd:Nnn \numrange { o m m }
299 { #2 \tl_use:N \l_siunitx_range_phrase_tl #3 }
300 \__siunitx_bookmark_cmd:Nnn \qtyrange { o m m m }
301 { #2 ~ #4 \tl_use:N \l_siunitx_range_phrase_tl #3 ~ #4 }

```

(End definition for __siunitx_bookmark_cmd:Nn.)

\c__siunitx_bookmark_seq

Commands usable in bookmarks

```

302 \seq_const_from_clist:Nn \c__siunitx_bookmark_seq

```

```

303 {
304   \ang , \qty , \num , \unit ,
305   \numlist , \qtylist ,
306   \numrange , \qtyrange
307 }

(End definition for \c__siunitx_bookmark_seq.)
Activate the document commands here: the unit macros are handled in siunitx-final.

308 \AtBeginDocument
309 {
310   \@ifpackageloaded { hyperref }
311   {
312     \pdfstringdefDisableCommands
313     {
314       \seq_map_inline:Nn \c__siunitx_bookmark_seq
315       {
316         \cs_set_eq:Nc #1
317         { \cs_to_str:N #1 \c_space_tl ( pdfstring ~ context ) }
318       }
319     }
320     \pdfstringdefDisableCommands
321     {
322       \siunitx_unit_pdfstring_context:
323       \cs_if_exist:NT \FB@fg { \def \fg { \FB@fg } }
324     }
325   }
326   { }
327 }

```

_siunitx_angle:n Expandable splitting of the angle: simply enough, also outputs the
_siunitx_angle:w

```

328 \cs_new:Npn \_siunitx_angle:n #1
329 { \_siunitx_angle:w #1 ; ; ; \q_stop }
330 \cs_new:Npn \_siunitx_angle:w #1 ; #2 ; #3 ; #4 \q_stop
331 {
332   \tl_if_blank:nF {#1}
333   { #1 \degree }
334   \tl_if_blank:nF {#2}
335   {
336     \tl_if_blank:nF {#1} { \c_space_tl }
337     #2 \arcminute
338   }
339   \tl_if_blank:nF {#3}
340   {
341     \tl_if_blank:nF {#1#2} { \c_space_tl }
342     #3 \arcsecond
343   }
344 }

```

(End definition for _siunitx_angle:n and _siunitx_angle:w.)

_siunitx_list_use:nnnnn Copies of the ideas in the l3clist module but using ; as a list separator. The functions
_siunitx_list_use:nnVVV have to be extended to allow for a unit argument.

```

\_siunitx_list_use_aux:nnnnn 345 \cs_new:Npn \_siunitx_list_use:nnnnn #1#2#3#4#5
\_siunitx_list_use_auxi:w 346 {
\_siunitx_list_use_auxii:nnw
\_siunitx_list_use_auxiii:nnw
\_siunitx_list_count:n
\_siunitx_list_count:w

```

```

347 \tl_if_blank:nTF {#2}
348 { \__siunitx_list_use_aux:nnnnn {#1} { } }
349 { \__siunitx_list_use_aux:nnnnn {#1} { ~ #2 } }
350 {#3} {#4} {#5}
351 }
352 \cs_generate_variant:Nn \__siunitx_list_use:nnnnn { nnVVV }
353 \cs_new:Npn \__siunitx_list_use_aux:nnnnn #1#2#3#4#5
354 {
355 \int_case:nnF { \__siunitx_list_count:n {#1} }
356 {
357 { 0 } { }
358 { 1 } { \__siunitx_list_use_auxi:nw {#2} #1 ; ; { } }
359 { 2 } { \__siunitx_list_use_auxi:nw {#2} #1 ; {#3} }
360 }
361 {
362 \__siunitx_list_use_auxii:nnw {#2} { } #1 ;
363 \q_mark ; { \__siunitx_list_use_auxii:nnw {#2} {#4} }
364 \q_mark ; { \__siunitx_list_use_auxiii:nnw {#2} {#5} }
365 \q_stop { }
366 }
367 }
368 \cs_new:Npn \__siunitx_list_use_auxi:nw #1#2 ; #3 ; #4
369 { #2 #1 #4 #3 \tl_if_blank:nF {#3} {#1} }
370 \cs_new:Npn \__siunitx_list_use_auxii:nnw
371 #1#2#3 ; #4 ; #5 ; #6 \q_mark ; #7#8 \q_stop #9
372 { #7 {#4} ; {#5} ; #6 \q_mark ; {#7} #8 \q_stop { #9 #2 #3 #1 } }
373 \cs_new:Npn \__siunitx_list_use_auxiii:nnw #1#2#3 ; #4 \q_stop #5
374 { #5 #2 #3 #1 }
375 \cs_new:Npx \__siunitx_list_count:n #1
376 {
377 \exp_not:N \int_eval:n
378 {
379 0
380 \exp_not:N \__siunitx_list_count:w \c_space_tl
381 #1 \exp_not:n { ; \q_recursion_tail ; \q_recursion_stop }
382 }
383 }
384 \cs_new:Npx \__siunitx_list_count:w #1 ;
385 {
386 \exp_not:n { \exp_args:Nf \quark_if_recursion_tail_stop:n } {#1}
387 \exp_not:N \tl_if_blank:nF {#1} { + 1 }
388 \exp_not:N \__siunitx_list_count:w \c_space_tl
389 }

```

(End definition for __siunitx_list_use:nnnnn and others.)

```

390 </interfaces>
391 </package>

```

Part II

siunitx-angle – Formatting angles

1 Formatting angles

`\siunitx_angle:n`
`\siunitx_angle:nnn`

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

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

1.1 Key-value options

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

`angle-mode`

`angle-mode = ⟨choice⟩`

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

`arc-separator`

`arc-separator = ⟨separator⟩`

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

`arc-separator-over-decimal`

`arc-separator-over-decimal = true|false`

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

`fill-arc-degrees`

`fill-arc-degrees = true|false`

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

`fill-arc-minutes`

`fill-arc-minutes = true|false`

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

`fill-arc-seconds`

`fill-arc-seconds = true|false`

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

`number-angle-product`

`number-angle-product = $\langle separator \rangle$`

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

2 siunitx-angle implementation

Start the DocStrip guards.

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

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

```
2  $\langle @@=siunitx\_angle \rangle$ 
```

`\l__siunitx_angle_tmp_bool` Scratch space.

```
3 \bool_new:N \l__siunitx_angle_tmp_bool
4 \dim_new:N \l__siunitx_angle_tmp_dim
5 \tl_new:N \l__siunitx_angle_tmp_tl
```

(End definition for `\l__siunitx_angle_tmp_bool`, `\l__siunitx_angle_tmp_dim`, and `\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

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



```

37 \bool_new:N \l__siunitx_angle_force_arc_bool
38 \bool_new:N \l__siunitx_angle_force_decimal_bool

```

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

`\siunitx_angle: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.

`\siunitx_angle:nnn`

```

39 \cs_new_protected:Npn \siunitx_angle:n #1
40 {
41   \bool_if:NTF \l__siunitx_angle_force_arc_bool
42   { \exp_args:Ne \__siunitx_angle_arc_convert:n { \fp_eval:n {#1} } }
43   {
44     \siunitx_number_parse:nN {#1} \l__siunitx_angle_degrees_tl
45     \tl_set:Nx \l__siunitx_angle_degrees_tl
46     { \siunitx_number_output:NN \l__siunitx_angle_degrees_tl \q_nil }
47     \__siunitx_angle_arc_print:VVV
48     \l__siunitx_angle_degrees_tl
49     \c_empty_tl
50     \c_empty_tl
51   }
52 }
53 \cs_new_protected:Npn \siunitx_angle:nnn #1#2#3
54 {
55   \bool_if:NTF \l__siunitx_angle_force_decimal_bool
56   {
57     \exp_args:Ne \siunitx_angle:n
58     { \fp_eval:n { #1 + (#2) / 60 + (#3) / 3600 } }
59   }
60   { \__siunitx_angle_arc_sign:nnn {#1} {#2} {#3} }
61 }
62 \cs_new_protected:Npn \__siunitx_angle_arc_convert:n #1
63 {
64   \use:x
65   {
66     \siunitx_angle:nnn
67     { \fp_eval:n { trunc(#1,0) } }
68     { \fp_eval:n { trunc((#1 - trunc(#1,0)) * 60,0) } }
69     {
70       \fp_eval:n
71       {
72         (
73           (#1 - trunc(#1,0)) * 60
74           - trunc((#1 - trunc(#1,0)) * 60,0)
75         )
76         * 60
77       }
78     }
79   }
80 }

```

(End definition for `\siunitx_angle:n`, `\siunitx_angle:nnn`, and `__siunitx_angle_arc_convert:n`. These functions are documented on page 11.)

```

\l__siunitx_angle_degrees_tl Space for formatting parsed numbers.
\l__siunitx_angle_minutes_tl 81 \tl_new:N \l__siunitx_angle_degrees_tl
\l__siunitx_angle_seconds_tl 82 \tl_new:N \l__siunitx_angle_minutes_tl
83 \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”.
84 \tl_new:N \l__siunitx_angle_sign_tl

(End definition for \l__siunitx_angle_sign_tl.)

\__siunitx_angle_arc_sign:nnn To get the sign in the right place whilst dealing with zero filling means doing some
\__siunitx_angle_arc_sign:nn shuffling. That means doing processing of each number manually.
\__siunitx_angle_extract_sign:nnnnnnnn 85 \cs_new_protected:Npn \__siunitx_angle_arc_sign:nnn #1#2#3
\__siunitx_angle_sign:nnnnnnnn 86 {
87   \group_begin:
88   \keys_set:nn { siunitx }
89   {
90     input-close-uncertainty = ,
91     input-exponent-markers = ,
92     input-open-uncertainty = ,
93     input-uncertainty-signs =
94   }
95   \tl_clear:N \l__siunitx_angle_sign_tl
96   \__siunitx_angle_arc_sign:nn {#1} { degrees }
97   \__siunitx_angle_arc_sign:nn {#2} { minutes }
98   \__siunitx_angle_arc_sign:nn {#3} { seconds }
99   \tl_if_empty:NF \l__siunitx_angle_sign_tl
100   {
101     \clist_map_inline:nn { degrees , minutes , seconds }
102     {
103       \tl_if_empty:cF { l__siunitx_angle_ ##1 _tl }
104       {
105         \tl_set:cx { l__siunitx_angle_ ##1 _tl }
106         {
107           { }
108           { \exp_not:V \l__siunitx_angle_sign_tl }
109           \exp_after:wN \exp_after:wN \exp_after:wN
110             \__siunitx_angle_sign:nnnnnnnn
111             \cs:w l__siunitx_angle_ ##1 _tl \cs_end:
112         }
113         \clist_map_break:
114       }
115     }
116   }
117   \clist_map_inline:nn { degrees , minutes , seconds }
118   {
119     \tl_if_empty:cF { l__siunitx_angle_ ##1 _tl }
120     {
121       \tl_set:cx { l__siunitx_angle_ ##1 _tl }
122       {
123         \exp_args:Nc \siunitx_number_output:NN

```

```

124         { l__siunitx_angle_ ##1 _t1 } \q_nil
125     }
126 }
127 }
128 \__siunitx_angle_arc_print:VVV
129 \l__siunitx_angle_degrees_t1
130 \l__siunitx_angle_minutes_t1
131 \l__siunitx_angle_seconds_t1
132 \group_end:
133 }
134 \cs_new_protected:Npn \__siunitx_angle_arc_sign:nn #1#2
135 {
136     \tl_if_blank:nTF {#1}
137     {
138         \bool_if:cTF { l__siunitx_angle_fill_ #2 _bool }
139         {
140             \tl_set:cn { l__siunitx_angle_ #2 _t1 }
141             { { } { } { 0 } { } { } { } { 0 } }
142         }
143         { \tl_clear:c { l__siunitx_angle_ #2 _t1 } }
144     }
145     {
146         \siunitx_number_parse:nN {#1} \l__siunitx_angle_tmp_t1
147         \exp_after:wN \__siunitx_angle_extract_sign:nnnnnnnn \l__siunitx_angle_tmp_t1 {#2}
148     }
149 }
150 \cs_new_protected:Npn \__siunitx_angle_extract_sign:nnnnnnnn #1#2#3#4#5#6#7#8
151 {
152     \tl_if_blank:nTF {#2}
153     { \tl_set_eq:cn { l__siunitx_angle_ #8 _t1 } \l__siunitx_angle_tmp_t1 }
154     {
155         \tl_set:cn { l__siunitx_angle_ #8 _t1 }
156         { {#1} { } {#3} {#4} {#5} {#6} {#7} }
157         \tl_set:Nn \l__siunitx_angle_sign_t1 {#2}
158         \keys_set:nn { siunitx }
159         { input-comparators = , input-signs = }
160     }
161 }
162 \cs_new:Npn \__siunitx_angle_sign:nnnnnnnn #1#2#3#4#5#6#7
163 { \exp_not:n { {#3} {#4} {#5} {#6} {#7} } }

```

(End definition for __siunitx_angle_arc_sign:nnn and others.)

\l__siunitx_angle_marker_box
\l__siunitx_angle_unit_box

For “astronomy style” angles.

```

164 \box_new:N \l__siunitx_angle_marker_box
165 \box_new:N \l__siunitx_angle_unit_box

```

(End definition for \l__siunitx_angle_marker_box and \l__siunitx_angle_unit_box.)

__siunitx_angle_arc_print:nnn
__siunitx_angle_arc_print:VVV
__siunitx_angle_arc_print_auxi:nnn
__siunitx_angle_arc_print_auxii:w
__siunitx_angle_arc_print_auxiii:n
__siunitx_angle_arc_print_auxiv:NN
__siunitx_angle_arc_print_auxv:w
__siunitx_angle_arc_print_auxvi:n

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 The need to handle astronomy-style formatting means that the number has to be decomposed into parts.

```

166 \cs_new_protected:Npn \__siunitx_angle_arc_print:nnn #1#2#3
167 {

```

```

168 \__siunitx_angle_arc_print_auxi:nnn {#1} { \degree } {#2#3}
169 \__siunitx_angle_arc_print_auxi:nnn {#2} { \arcminute } {#3}
170 \__siunitx_angle_arc_print_auxi:nnn {#3} { \arcsecond } { }
171 }
172 \cs_generate_variant:Nn \__siunitx_angle_arc_print:nnn { VVV }
173 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxi:nnn #1#2#3
174 {
175   \tl_if_blank:nF {#1}
176   {
177     \bool_if:NTF \l__siunitx_angle_astronomy_bool
178     { \__siunitx_angle_arc_print_auxii:nw {#2} #1 \q_stop }
179     {
180       \__siunitx_angle_arc_print_auxv:w #1 \q_stop
181       \__siunitx_angle_arc_print_auxvi:n {#2}
182     }
183     \tl_if_blank:nF {#3}
184     {
185       \nobreak
186       \l__siunitx_angle_separator_tl
187     }
188   }
189 }
190 % \end{macrocode}
191 % To align the two parts of the astronomy-style marker, we need to allow
192 % for the |\scriptspace|.
193 % \begin{macrocode}
194 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxii:nw
195 #1#2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
196 {
197   \mode_if_math:TF
198   { \bool_set_true:N \l__siunitx_angle_tmp_bool }
199   { \bool_set_false:N \l__siunitx_angle_tmp_bool }
200   \siunitx_print:nn { number } {#2#3#4}
201   \tl_if_blank:nTF {#6}
202   { \__siunitx_angle_arc_print_auxvi:n {#1} }
203   {
204     \hbox_set:Nn \l__siunitx_angle_marker_box
205     {
206       \__siunitx_angle_arc_print_auxiii:n
207       { \siunitx_print:nn { number } {#5} }
208     }
209     \hbox_set:Nn \l__siunitx_angle_unit_box
210     {
211       \__siunitx_angle_arc_print_auxiii:n
212       {
213         \siunitx_unit_format:nN {#1} \l__siunitx_angle_tmp_tl
214         \siunitx_print:nV { unit } \l__siunitx_angle_tmp_tl
215         \skip_horizontal:n { -\scriptspace }
216       }
217     }
218     \dim_compare:nNnTF { \box_wd:N \l__siunitx_angle_marker_box } >
219     { \box_wd:N \l__siunitx_angle_unit_box }
220     {
221       \__siunitx_angle_arc_print_auxiv:NN

```

```

222         \l__siunitx_angle_marker_box
223         \l__siunitx_angle_unit_box
224     }
225     {
226         \__siunitx_angle_arc_print_auxiv:NN
227         \l__siunitx_angle_unit_box
228         \l__siunitx_angle_marker_box
229     }
230     \hbox_set_to_wd:Nnn \l__siunitx_angle_marker_box
231     \l__siunitx_angle_tmp_dim
232     {
233         \hbox_overlap_right:n
234         { \box_use_drop:N \l__siunitx_angle_marker_box }
235         \hbox_overlap_right:n
236         { \box_use_drop:N \l__siunitx_angle_unit_box }
237         \tex_hfil:D
238     }
239     \box_use:N \l__siunitx_angle_marker_box
240     \skip_horizontal:N \scriptspace
241     \siunitx_print:nn { number } {#6}
242 }
243 }
244 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxiii:n #1
245 {
246     \bool_if:NTF \l__siunitx_angle_tmp_bool
247     { \ensuremath }
248     { \use:n }
249     {#1}
250 }
251 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxiv:NN #1#2
252 {
253     \dim_set:Nn \l__siunitx_angle_tmp_dim { \box_wd:N #1 }
254     \hbox_set_to_wd:Nnn #2
255     \l__siunitx_angle_tmp_dim
256     {
257         \tex_hss:D
258         \hbox_unpack_drop:N #2
259         \tex_hss:D
260     }
261 }
262 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxv:w
263 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_stop
264 { \siunitx_print:nn { number } {#1#2#3#4#5} }
265 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxvi:n #1
266 {
267     \nobreak
268     \l__siunitx_angle_product_tl
269     \siunitx_unit_format:nN {#1} \l__siunitx_angle_tmp_tl
270     \siunitx_print:nV { unit } \l__siunitx_angle_tmp_tl
271 }

```

(End definition for __siunitx_angle_arc_print:nnn and others.)

```

272 \keys_set:nn { siunitx }
273 {

```

```

274     angle-mode                = input ,
275     angle-symbol-over-decimal = false ,
276     arc-separator             =      ,
277     fill-arc-degrees          = false ,
278     fill-arc-minutes          = false ,
279     fill-arc-seconds          = false ,
280     number-angle-product      =
281   }
282 </package>

```

Part III

siunitx-compound – Compound numbers and quantities

<code>\siunitx_compound_number:n</code>	<code>\siunitx_compound_number:n {<entries>}</code>
---	---

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

<code>\siunitx_compound_quantity:nn</code>	<code>\siunitx_compound_quantity:nn {<entries>} {<unit>}</code>
--	---

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

<code>\siunitx_number_list:nn</code>	<code>\siunitx_number_list:nn {<entries>}</code>
--------------------------------------	--

Prints the list of numbers in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$.

<code>\siunitx_quantity_list:nn</code>	<code>\siunitx_quantity_list:nn {<entries>} {<unit>}</code>
--	---

Prints the list of quantities in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$.

<code>\siunitx_number_product:n</code>	<code>\siunitx_number_product:n {<entries>}</code>
--	--

Prints the series of numbers in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$.

<code>\siunitx_quantity_product:nn</code>	<code>\siunitx_number_product:n {<entries>} {<unit>}</code>
---	---

Prints the series of quantities in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$.

<code>\siunitx_number_range:nn</code>	<code>\siunitx_number_range:nn {<start>} {<end>}</code>
---------------------------------------	---

Prints the range of numbers from the $\langle start \rangle$ to the $\langle end \rangle$.

<code>\siunitx_quantity_range:nnn</code>	<code>\siunitx_number_range:nn {<start>} {<end>} {<unit>}</code>
--	--

Prints the range of quantities from the $\langle start \rangle$ to the $\langle end \rangle$.

<code>\l_siunitx_list_separator_pair_tl</code>
<code>\l_siunitx_list_separator_tl</code>
<code>\l_siunitx_list_separator_final_tl</code>

Separators for lists of numbers and quantities.

<u>\l_siunitx_range_phrase_tl</u>	Phrase (or similar) used between limits of a range.
<u>compound-exponents</u>	compound-exponents = combine combine-bracket individual
<u>compound-final-separator</u>	compound-final-separator = $\langle text \rangle$
<u>compound-pair-separator</u>	compound-pair-separator = $\langle text \rangle$
<u>compound-separator</u>	compound-separator = $\langle text \rangle$
<u>compound-separator-mode</u>	compound-separator-mode = number text
<u>compound-units</u>	compound-units = bracket repeat single
<u>list-exponents</u>	list-exponents = combine combine-bracket individual
<u>list-final-separator</u>	list-final-separator = $\langle text \rangle$
<u>list-pair-separator</u>	list-pair-separator = $\langle text \rangle$
<u>list-separator</u>	list-separator = $\langle text \rangle$
<u>list-units</u>	list-units = bracket repeat single
<u>product-exponents</u>	product-exponents = combine combine-bracket individual
<u>product-mode</u>	product-mode = phrase choice
<u>product-phrase</u>	product-phrase = $\langle text \rangle$
<u>product-symbol</u>	product-symbol = $\langle symbol \rangle$
<u>range-exponents</u>	range-exponents = combine combine-bracket individual

<code>range-phrase</code>	<code>range-phrase = <text></code>
---------------------------	--

<code>range-units</code>	<code>range-units = bracket repeat single</code>
--------------------------	--

Start the DocStrip guards.

1 `<*package>`

1 siunitx-compound implementation

2 `\cs_generate_variant:Nn \keys_set:nn { nx }`

1.1 General mechanism

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

3 `<@@=siunitx_compound>`

Typesetting lists, ranges and products of numbers or quantities has shared features which mean they are best handled using a common mechanism. The aim therefore is to abstract out enough of the process such that output-specific aspects can be left to separate processes.

Scratch space.

4 `\fp_new:N \l__siunitx_compound_tmp_fp`
5 `\seq_new:N \l__siunitx_compound_tmp_seq`
6 `\tl_new:N \l__siunitx_compound_tmp_tl`

(End definition for `\l__siunitx_compound_tmp_fp`, `\l__siunitx_compound_tmp_seq`, and `\l__siunitx_compound_tmp_tl`.)

The first number in the list in internal format.

7 `\tl_new:N \l__siunitx_compound_first_tl`

(End definition for `\l__siunitx_compound_first_tl`.)

For storing the combined exponent, if present.

8 `\tl_new:N \l__siunitx_compound_exp_tl`

(End definition for `\l__siunitx_compound_exp_tl`.)

Data on the end-of-list.

9 `\tl_new:N \l__siunitx_compound_start_tl`
10 `\tl_new:N \l__siunitx_compound_end_tl`

(End definition for `\l__siunitx_compound_start_tl` and `\l__siunitx_compound_end_tl`.)

Data on the length-of-list.

11 `\int_new:N \l__siunitx_compound_count_int`

(End definition for `\l__siunitx_compound_count_int`.)

`\l__siunitx_compound_unit_tl`

12 `\tl_new:N \l__siunitx_compound_unit_tl`

(End definition for \l__siunitx_compound_unit_tl.)

Purely internal for the present.

```
\l__siunitx_compound_bracket_close_tl
\l__siunitx_compound_bracket_open_tl
13 \tl_new:N \l__siunitx_compound_bracket_close_tl
14 \tl_new:N \l__siunitx_compound_bracket_open_tl
15 \tl_set:Nn \l__siunitx_compound_bracket_open_tl { ( }
16 \tl_set:Nn \l__siunitx_compound_bracket_close_tl { ) }
```

(End definition for \l__siunitx_compound_bracket_close_tl and \l__siunitx_compound_bracket_open_tl.)

List options.

```
\l__siunitx_compound_separator_final_tl
\l__siunitx_compound_separator_pair_tl
\l__siunitx_compound_separator_tl
\l__siunitx_compound_separator_text_bool
\l__siunitx_compound_exp_bracket_bool
\l__siunitx_compound_exp_combine_bool
\l__siunitx_compound_unit_bracket_bool
\l__siunitx_compound_unit_repeat_bool
17 \bool_new:N \l__siunitx_compound_exp_bracket_bool
18 \bool_new:N \l__siunitx_compound_exp_combine_bool
19 \bool_new:N \l__siunitx_compound_separator_text_bool
20 \bool_new:N \l__siunitx_compound_unit_bracket_bool
21 \bool_new:N \l__siunitx_compound_unit_power_bool
22 \bool_new:N \l__siunitx_compound_unit_repeat_bool
23 \keys_define:nn { siunitx }
24 {
25   compound-exponents .choice: ,
26   compound-exponents / combine .code:n =
27   {
28     \bool_set_false:N \l__siunitx_compound_exp_bracket_bool
29     \bool_set_true:N \l__siunitx_compound_exp_combine_bool
30   } ,
31   compound-exponents / combine-bracket .code:n =
32   {
33     \bool_set_true:N \l__siunitx_compound_exp_bracket_bool
34     \bool_set_true:N \l__siunitx_compound_exp_combine_bool
35   } ,
36   compound-exponents / individual .code:n =
37   {
38     \bool_set_false:N \l__siunitx_compound_exp_bracket_bool
39     \bool_set_false:N \l__siunitx_compound_exp_combine_bool
40   } ,
41   compound-final-separator .tl_set:N =
42     \l__siunitx_compound_separator_final_tl ,
43   compound-pair-separator .tl_set:N =
44     \l__siunitx_compound_separator_pair_tl ,
45   compound-separator .tl_set:N =
46     \l__siunitx_compound_separator_tl ,
47   compound-separator-mode .choice: ,
48   compound-separator-mode / number .code:n =
49     { \bool_set_false:N \l__siunitx_compound_separator_text_bool } ,
50   compound-separator-mode / text .code:n =
51     { \bool_set_true:N \l__siunitx_compound_separator_text_bool } ,
52   compound-units .choice: ,
53   compound-units / bracket .code:n =
54   {
55     \bool_set_true:N \l__siunitx_compound_unit_bracket_bool
56     \bool_set_false:N \l__siunitx_compound_unit_power_bool
57     \bool_set_false:N \l__siunitx_compound_unit_repeat_bool
58   } ,
```

```

59 compound-units / bracket-power .code:n =
60 {
61     \bool_set_true:N \l__siunitx_compound_unit_bracket_bool
62     \bool_set_true:N \l__siunitx_compound_unit_power_bool
63     \bool_set_false:N \l__siunitx_compound_unit_repeat_bool
64 } ,
65 compound-units / power .code:n =
66 {
67     \bool_set_false:N \l__siunitx_compound_unit_bracket_bool
68     \bool_set_true:N \l__siunitx_compound_unit_power_bool
69     \bool_set_false:N \l__siunitx_compound_unit_repeat_bool
70 } ,
71 compound-units / repeat .code:n =
72 {
73     \bool_set_false:N \l__siunitx_compound_unit_bracket_bool
74     \bool_set_false:N \l__siunitx_compound_unit_power_bool
75     \bool_set_true:N \l__siunitx_compound_unit_repeat_bool
76 } ,
77 compound-units / single .code:n =
78 {
79     \bool_set_false:N \l__siunitx_compound_unit_bracket_bool
80     \bool_set_false:N \l__siunitx_compound_unit_power_bool
81     \bool_set_false:N \l__siunitx_compound_unit_repeat_bool
82 }
83 }

```

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

```

\siunitx_compound_number:n
\__siunitx_compound_format:n
\__siunitx_compound_format:nnn

```

Printing a generic set starts with the question of whether we want to extract exponents. If we do, then there is the work to do with extraction. Either way, the printing is handed off to a common function. We do a quick count up-front to avoid excess work when there is not actually a list.

```

84 \cs_new_protected:Npn \siunitx_compound_number:n #1
85 {
86     \group_begin:
87     \__siunitx_compound_format:nn {#1} { }
88     \__siunitx_compound_print:N \__siunitx_compound_print_number:n
89     \group_end:
90 }
91 \cs_new_protected:Npn \__siunitx_compound_format:nn #1#2
92 {
93     \seq_clear:N \l__siunitx_compound_tmp_seq
94     \bool_if:NTF \l_siunitx_number_parse_bool
95     {
96         \exp_args:Nxx \__siunitx_compound_format:nnn
97         { \tl_head:n {#1} }
98         { \tl_tail:n {#1} }
99         {#2}
100     }
101     { \tl_map_function:nN {#1} \__siunitx_compound_unparsed:n }
102 }

```

Formatting at a low level needs to know about units and numbers: we have to exchange data between the two. Most of the business of handling the units is left to a dedicated auxiliary.

```

103 \cs_new_protected:Npn \__siunitx_compound_format:nnn #1#2#3
104 {
105   \siunitx_number_parse:nN {#1} \l__siunitx_compound_tmp_tl
106   \tl_if_blank:nTF {#3}
107     { \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
108       { \__siunitx_compound_format_units:nn {#2} {#3} }
109     \bool_lazy_and:nnTF
110       { \l__siunitx_compound_exp_combine_bool }
111       { \int_compare_p:nNn { \tl_count:n {#2} } > 0 }
112       { \__siunitx_compound_extract_exponents: }
113       {
114         \tl_set:Nx \l__siunitx_compound_tmp_tl
115         { \siunitx_number_output:N \l__siunitx_compound_first_tl }
116         \seq_put_right:NV \l__siunitx_compound_tmp_seq \l__siunitx_compound_tmp_tl
117       }
118     \tl_map_function:nN {#2} \__siunitx_compound_parsed:n
119   }

```

(End definition for `\siunitx_compound_number:n`, `__siunitx_compound_format:n`, and `__siunitx_compound_format:nnn`. This function is documented on page 19.)

`__siunitx_compound_extract_exponents:N`
`__siunitx_compound_extract_exponents_auxi:w`
`\siunitx_compound_extract_exponents_auxii:nw`
`__siunitx_compound_extract_exponents_auxiii:nnnnnnn`

Extracting exponents means dealing with the first entry as a special case. After that, apply fixed processing to all other entries, tidying up using the number formatter.

```

120 \cs_new_protected:Npn \__siunitx_compound_extract_exponents:
121 {
122   \tl_set:Nx \l__siunitx_compound_tmp_tl
123   { \siunitx_number_output:NN \l__siunitx_compound_first_tl \q_nil }
124   \exp_after:wN \__siunitx_compound_extract_exponents_auxi:w
125   \l__siunitx_compound_tmp_tl \q_stop
126 }
127 \cs_new_protected:Npn \__siunitx_compound_extract_exponents_auxi:w
128 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil #8
129 \q_nil #9 \q_stop
130 {
131   \__siunitx_compound_extract_exponents_auxii:nw {#1#2#3#4#5#6#7#8} #9 \q_stop
132 }
133 \cs_new_protected:Npn \__siunitx_compound_extract_exponents_auxii:nw
134 #1#2 \q_nil #3 \q_nil #4 \q_stop
135 {
136   \seq_put_right:Nn \l__siunitx_compound_tmp_seq { #1#2 }
137   \tl_set:Nn \l__siunitx_compound_exp_tl { #3#4 }
138   \exp_after:wN \__siunitx_compound_extract_exponents_auxiii:nnnnnnn
139   \l__siunitx_compound_first_tl
140 }
141 \cs_new_protected:Npn \__siunitx_compound_extract_exponents_auxiii:nnnnnnn
142 #1#2#3#4#5#6#7
143 {
144   \keys_set:nn { siunitx }
145   {
146     drop-exponent = true ,
147     exponent-mode = fixed ,
148     fixed-exponent = #6#7
149   }
150 }

```

(End definition for `_siunitx_compound_extract_exponents:N` and others.)

`_siunitx_compound_parsed:n` The simple cases for parsing (or not) all entries.
`_siunitx_compound_unparsed:n`

```

151 \cs_new_protected:Npn \_siunitx_compound_parsed:n #1
152 {
153   \siunitx_number_format:nN {#1} \l__siunitx_compound_tmp_tl
154   \seq_put_right:NV \l__siunitx_compound_tmp_seq \l__siunitx_compound_tmp_tl
155 }
156 \cs_new_protected:Npn \_siunitx_compound_unparsed:n #1
157 {
158   \seq_put_right:Nn \l__siunitx_compound_tmp_seq { \ensuremath {#1} }
159 }

```

(End definition for `_siunitx_compound_parsed:n` and `_siunitx_compound_unparsed:n`.)

`_siunitx_compound_format_units:nn` Actually formatting the units is much the same as is done in the quantities module, except that we have to cover the multiplication cases too: gets a bit repetitive. Notice that when combining exponents, there is no adjustment to the original exponent: we purely need to extract it.

```

160 \cs_new_protected:Npn \_siunitx_compound_format_units:nn #1#2
161 {
162   \bool_if:NTF \l__siunitx_compound_unit_power_bool
163   {
164     \use:c { \_siunitx_compound_format_ \l_siunitx_quantity_prefix_mode_tl :nn }
165     {#2} { \tl_count:n {#1} + 1 }
166   }
167   {
168     \use:c { \_siunitx_compound_format_ \l_siunitx_quantity_prefix_mode_tl :n } {#2}
169   }
170 }
171 \cs_new_protected:cpx { \_siunitx_compound_format_combine-exponent:n } #1
172 {
173   \exp_not:c { \_siunitx_compound_format_combine-exponent_aux:n }
174   {
175     \exp_not:N \siunitx_unit_format_combine_exponent:nnN
176     {#1}
177   }
178 }
179 \cs_new_protected:cpx { \_siunitx_compound_format_combine-exponent:nn } #1#2
180 {
181   \exp_not:c { \_siunitx_compound_format_combine-exponent_aux:n }
182   {
183     \exp_not:N \siunitx_unit_format_multiply_combine_exponent:nnnN
184     {#1} {#2}
185   }
186 }
187 \cs_new_protected:cpn { \_siunitx_compound_format_combine-exponent_aux:n } #1
188 {
189   \bool_set_true:N \l__siunitx_compound_exp_combine_bool
190   \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
191   \exp_args:NV \_siunitx_compound_extract_exp:nN
192   \l__siunitx_compound_first_tl \l__siunitx_compound_tmp_fp
193   #1 \l__siunitx_compound_tmp_fp \l__siunitx_compound_unit_tl
194 }

```

```

195 \cs_new_protected:cpx { __siunitx_compound_format_extract-exponent:n } #1
196 {
197   \exp_not:c { __siunitx_compound_format_extract-exponent_aux:n }
198   { \exp_not:N \siunitx_unit_format_extract_prefixes:nnN {#1} }
199 }
200 \cs_new_protected:cpx { __siunitx_compound_format_extract-exponent:nn } #1#2
201 {
202   \exp_not:c { __siunitx_compound_format_extract-exponent_aux:n }
203   {
204     \exp_not:N \siunitx_unit_format_multiply_extract_prefixes:nnNN
205     {#1} {#2}
206   }
207 }
208 \cs_new_protected:cpn { __siunitx_compound_format_extract-exponent_aux:n } #1
209 {
210   #1 \l__siunitx_compound_unit_tl \l__siunitx_compound_tmp_fp
211   \tl_set:Nx \l__siunitx_compound_tmp_tl
212   { \siunitx_number_adjust_exponent:Nn \l__siunitx_compound_tmp_tl \l__siunitx_compound_
213     \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
214     \bool_set_true:N \l__siunitx_compound_exp_combine_bool
215   }
216 \cs_new_protected:Npn \__siunitx_compound_format_input:n #1
217 {
218   \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
219   \siunitx_unit_format:nN {#1} \l__siunitx_compound_unit_tl
220 }
221 \cs_new_protected:Npn \__siunitx_compound_format_input:nn #1#2
222 {
223   \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
224   \siunitx_unit_format_multiply:nnN {#1} {#2} \l__siunitx_compound_unit_tl
225 }
226 \cs_new_protected:Npn \__siunitx_compound_extract_exp:nN #1#2
227 { \__siunitx_compound_extract_exp:nnnnnnN #1 #2 }
228 \cs_new_protected:Npn \__siunitx_compound_extract_exp:nnnnnnN #1#2#3#4#5#6#7#8
229 { \fp_set:Nn #8 {#6#7} }

```

(End definition for `__siunitx_compound_format_units:nn` and others.)

`\siunitx_compound_quantity:nn` For quantities, life is more complex as there are interactions between the options for exponents and units.

```

230 \cs_new_protected:Npn \siunitx_compound_quantity:nn #1#2
231 {
232   \group_begin:
233   \bool_if:NT \l__siunitx_compound_unit_bracket_bool
234   { \bool_set_true:N \l__siunitx_compound_exp_bracket_bool }
235   \bool_if:NT \l__siunitx_compound_unit_repeat_bool
236   { \bool_set_false:N \l__siunitx_compound_exp_combine_bool }
237   \bool_lazy_or:nnT
238   { \l__siunitx_compound_unit_bracket_bool }
239   { ! \l__siunitx_compound_unit_repeat_bool }
240   { \bool_set_false:N \l__siunitx_number_bracket_ambiguous_bool }
241   \__siunitx_compound_format:nn {#1} {#2}
242   \str_if_eq:VnT \l__siunitx_quantity_prefix_mode_tl { combine-exponent }
243   { \tl_clear:N \l__siunitx_compound_exp_tl }

```

```

244 \bool_if:NTF \l__siunitx_compound_unit_repeat_bool
245 { \__siunitx_compound_print:N \__siunitx_compound_print_quantity:n }
246 {
247   \bool_lazy_and:nnTF
248   { \l__siunitx_compound_unit_bracket_bool }
249   { \tl_if_empty_p:N \l__siunitx_compound_exp_tl }
250   {
251     \siunitx_print:nV { number } \l__siunitx_compound_bracket_open_tl
252     \__siunitx_compound_print:N \__siunitx_compound_print_number:n
253     \siunitx_print:nV { number } \l__siunitx_compound_bracket_close_tl
254   }
255   { \__siunitx_compound_print:N \__siunitx_compound_print_number:n }
256   \__siunitx_compound_print_quantity:n { }
257 }
258 \group_end:
259 }

```

(End definition for \siunitx_compound_quantity:nn. This function is documented on page 19.)

```

\__siunitx_compound_print:N
  \__siunitx_compound_print:nnN
  \__siunitx_compound_print:xxN
  \__siunitx_compound_print:nnnN
  \__siunitx_compound_print_aux:n
  \__siunitx_compound_print_aux:nn
  \__siunitx_compound_print_number:n

```

We now need to know how many entries there are: the reason we don't use \seq_use:Nnnn is that we want to be able to insert \siunitx_print:nn in a controlled way.

```

260 \cs_new_protected:Npn \__siunitx_compound_print:N #1
261 {
262   \bool_lazy_and:nnTF
263   { \l__siunitx_compound_exp_bracket_bool }
264   { ! \tl_if_empty_p:N \l__siunitx_compound_exp_tl }
265   {
266     \__siunitx_compound_print:xxN
267     { \exp_not:V \l__siunitx_compound_bracket_open_tl }
268     {
269       \exp_not:V \l__siunitx_compound_bracket_close_tl
270       \exp_not:V \l__siunitx_compound_exp_tl
271     }
272     #1
273   }
274   { \__siunitx_compound_print:xxN { } { \exp_not:V \l__siunitx_compound_exp_tl } #1 }
275 }
276 \cs_new_protected:Npn \__siunitx_compound_print:nnN #1#2#3
277 {
278   \exp_args:Nx \__siunitx_compound_print:nnnN
279   { \seq_count:N \l__siunitx_compound_tmp_seq } {#1} {#2} #3
280 }
281 \cs_generate_variant:Nn \__siunitx_compound_print:nnN { xx }

```

A rather long auxiliary as we want a way to have the brackets/exponent available. The actual flow is simple enough: see how many entries there are and print as required. To keep everything generic, we have some slightly tricky saving of data to allow everything to go to the mapping.

```

282 \cs_new_protected:Npn \__siunitx_compound_print:nnnN #1#2#3#4
283 {
284   \int_case:nnF {#1}
285   {
286     { 0 } { }
287     { 1 }

```

```

288     {
289         #4
290         { \seq_item:Nn \l__siunitx_compound_tmp_seq { 1 } }
291     }
292 { 2 }
293 {
294     #4
295     {
296         \exp_not:n {#2}
297         \seq_item:Nn \l__siunitx_compound_tmp_seq { 1 }
298     }
299     \__siunitx_compound_print_separator:V \l__siunitx_compound_separator_pair_tl
300     #4
301     {
302         \seq_item:Nn \l__siunitx_compound_tmp_seq { 2 }
303         \exp_not:n {#3}
304     }
305 }
306 }
307 {
308     \int_set:Nn \l__siunitx_compound_count_int {#1}
309     \tl_set:Nn \l__siunitx_compound_start_tl {#2}
310     \tl_set:Nn \l__siunitx_compound_end_tl {#3}
311     \cs_set_eq:NN \__siunitx_compound_print_aux:n #4
312     \seq_map_indexed_function:NN
313         \l__siunitx_compound_tmp_seq
314         \__siunitx_compound_print_aux:nn
315 }
316 }
317 \cs_new_protected:Npn \__siunitx_compound_print_aux:n #1 { }
318 \cs_new_protected:Npn \__siunitx_compound_print_aux:nn #1#2
319 {
320     \int_case:nnF {#1}
321     {
322         { 1 }
323         {
324             \__siunitx_compound_print_aux:n
325             {
326                 \exp_not:V \l__siunitx_compound_start_tl
327                 \exp_not:n {#2}
328             }
329             \__siunitx_compound_print_separator:V \l__siunitx_compound_separator_tl
330         }
331         { \l__siunitx_compound_count_int - 1 }
332         {
333             \__siunitx_compound_print_aux:n { \exp_not:n {#2} }
334             \__siunitx_compound_print_separator:V \l__siunitx_compound_separator_final_tl
335         }
336         { \l__siunitx_compound_count_int }
337         {
338             \__siunitx_compound_print_aux:n
339             {
340                 \exp_not:n {#2}
341                 \exp_not:V \l__siunitx_compound_end_tl

```



```

342     }
343   }
344 }
345 {
346   \__siunitx_compound_print_aux:n { \exp_not:n {#2} }
347   \__siunitx_compound_print_separator:V \l__siunitx_compound_separator_tl
348 }
349 }
350 \cs_new_protected:Npn \__siunitx_compound_print_number:n #1
351 { \siunitx_print:nx { number } {#1} }
352 \cs_new_protected:Npn \__siunitx_compound_print_quantity:n #1
353 { \siunitx_quantity_print:nV {#1} \l__siunitx_compound_unit_tl }
354 \cs_new_protected:Npn \__siunitx_compound_print_separator:n #1
355 {
356   \bool_if:NTF \l__siunitx_compound_separator_text_bool
357   { \text {#1} }
358   { \siunitx_print:nn { number } {#1} }
359 }
360 \cs_generate_variant:Nn \__siunitx_compound_print_separator:n { V }

```

(End definition for `__siunitx_compound_print:N` and others.)

1.2 Lists

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

```

361 <@@=siunitx_list>

```

```

\l_siunitx_list_separator_tl
\l_siunitx_list_separator_final_tl
\l_siunitx_list_separator_pair_tl
\l__siunitx_list_exp_tl
\l__siunitx_list_units_tl

```

Options for products.

```

362 \tl_new:N \l__siunitx_list_exp_tl
363 \tl_new:N \l__siunitx_list_units_tl
364 \keys_define:nn { siunitx }
365 {
366   list-exponents .choices:nn =
367   { combine , combine-bracket , individual }
368   { \tl_set_eq:NN \l__siunitx_list_exp_tl \l_keys_choice_tl } ,
369   list-final-separator .tl_set:N = \l_siunitx_list_separator_final_tl ,
370   list-pair-separator .tl_set:N = \l_siunitx_list_separator_pair_tl ,
371   list-separator .tl_set:N = \l_siunitx_list_separator_tl ,
372   list-units .choices:nn =
373   { bracket , repeat , single }
374   { \tl_set_eq:NN \l__siunitx_list_units_tl \l_keys_choice_tl }
375 }

```

(End definition for `\l_siunitx_list_separator_tl` and others. These variables are documented on page 19.)

Simply recover the settings and use as a list.

```

\siunitx_number_list:nn
\siunitx_quantity_list:nn
\__siunitx_list_aux:

```

```

376 \cs_new_protected:Npn \siunitx_number_list:nn #1
377 {
378   \group_begin:
379   \__siunitx_list_aux:
380   \siunitx_compound_number:n {#1}
381   \group_end:

```

```

382 }
383 \cs_new_protected:Npn \siunitx_quantity_list:nn #1#2
384 {
385   \group_begin:
386     \__siunitx_list_aux:
387     \siunitx_compound_quantity:nn {#1} {#2}
388   \group_end:
389 }
390 \cs_new_protected:Npn \__siunitx_list_aux:
391 {
392   \keys_set:nx { siunitx }
393   {
394     compound-exponents      = \l__siunitx_list_exp_tl ,
395     compound-final-separator =
396       { \exp_not:V \l_siunitx_list_separator_final_tl } ,
397     compound-pair-separator =
398       { \exp_not:V \l_siunitx_list_separator_pair_tl } ,
399     compound-separator      =
400       { \exp_not:V \l_siunitx_list_separator_tl } ,
401     compound-separator-mode = text ,
402     compound-units          = \l__siunitx_list_units_tl
403   }
404 }

```

(End definition for `\siunitx_number_list:nn`, `\siunitx_quantity_list:nn`, and `__siunitx_list_aux:`. These functions are documented on page 19.)

1.3 Products

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

```

405 <@@=siunitx_product>

```

Options for products.

```

\l__siunitx_product_exp_tl
\l_siunitx_product_phrase_bool
\l_siunitx_product_phrase_tl
\l_siunitx_product_symbol_tl
\l__siunitx_product_units_tl
406 \tl_new:N \l__siunitx_product_exp_tl
407 \bool_new:N \l_siunitx_product_phrase_bool
408 \tl_new:N \l__siunitx_product_units_tl
409 \keys_define:nn { siunitx }
410 {
411   product-exponents .choices:nn =
412     { combine , combine-bracket , individual }
413     { \tl_set_eq:NN \l__siunitx_product_exp_tl \l_keys_choice_tl } ,
414   product-mode .choice: ,
415   product-mode / phrase .code:n =
416     { \bool_set_true:N \l_siunitx_product_phrase_bool } ,
417   product-mode / symbol .code:n =
418     { \bool_set_false:N \l_siunitx_product_phrase_bool } ,
419   product-phrase .tl_set:N = \l_siunitx_product_phrase_tl ,
420   product-symbol .tl_set:N = \l_siunitx_product_symbol_tl ,
421   product-units .choices:nn =
422     { bracket , bracket-power , power , repeat , single }
423     { \tl_set_eq:NN \l__siunitx_product_units_tl \l_keys_choice_tl }
424 }

```

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

Simply recover the settings and use as a list.

```

\siunitx_number_product:n
\siunitx_quantity_product:nn
  \__siunitx_product_aux:
  \__siunitx_product_aux:n
  \__siunitx_product_aux:x
425 \cs_new_protected:Npn \siunitx_number_product:n #1
426 {
427   \group_begin:
428     \__siunitx_product_aux:
429     \siunitx_compound_number:n {#1}
430   \group_end:
431 }
432 \cs_new_protected:Npn \siunitx_quantity_product:nn #1#2
433 {
434   \group_begin:
435     \__siunitx_product_aux:
436     \siunitx_compound_quantity:nn {#1} {#2}
437   \group_end:
438 }
439 \cs_new_protected:Npn \__siunitx_product_aux:
440 {
441   \bool_if:NTF \l__siunitx_product_phrase_bool
442     { \__siunitx_product_aux:x { \exp_not:V \l__siunitx_product_phrase_tl } }
443     { \__siunitx_product_aux:x { { } \exp_not:V \l__siunitx_product_symbol_tl { } } }
444 }
445 \cs_new_protected:Npn \__siunitx_product_aux:n #1
446 {
447   \keys_set:nx { siunitx }
448   {
449     compound-exponents      = \l__siunitx_product_exp_tl ,
450     compound-final-separator = { \exp_not:n {#1} } ,
451     compound-pair-separator = { \exp_not:n {#1} } ,
452     compound-separator      = { \exp_not:n {#1} } ,
453     compound-separator-mode =
454       \bool_if:NTF \l__siunitx_product_phrase_bool { text } { number } ,
455     compound-units          = \l__siunitx_product_units_tl
456   }
457 }
458 \cs_generate_variant:Nn \__siunitx_product_aux:n { x }

```

(End definition for `\siunitx_number_product:n` and others. These functions are documented on page 19.)

1.4 Ranges

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

```

459 <@@=siunitx_range>

```

Options for products.

```

\l__siunitx_range_exp_tl
\l__siunitx_range_phrase_tl
\l__siunitx_range_units_tl
460 \tl_new:N \l__siunitx_range_exp_tl
461 \tl_new:N \l__siunitx_range_units_tl
462 \keys_define:nn { siunitx }
463 {
464   range-exponents .choices:nn =
465     { combine , combine-bracket , individual }

```

```

466     { \tl_set_eq:NN \l__siunitx_range_exp_tl \l_keys_choice_tl } ,
467     range-phrase .tl_set:N = \l_siunitx_range_phrase_tl ,
468     range-units .choices:nn =
469     { bracket , repeat , single }
470     { \tl_set_eq:NN \l__siunitx_range_units_tl \l_keys_choice_tl }
471 }

```

(End definition for `\l__siunitx_range_exp_tl`, `\l_siunitx_range_phrase_tl`, and `\l__siunitx_range_units_tl`. This variable is documented on page 20.)

```

\siunitx_number_range:nn
\siunitx_quantity_range:nnn
  \__siunitx_range_aux:

```

Simply recover the settings and use as a list.

```

472 \cs_new_protected:Npn \siunitx_number_range:nn #1#2
473 {
474   \group_begin:
475     \__siunitx_range_aux:
476     \siunitx_compound_number:n { {#1} {#2} }
477   \group_end:
478 }
479 \cs_new_protected:Npn \siunitx_quantity_range:nnn #1#2#3
480 {
481   \group_begin:
482     \__siunitx_range_aux:
483     \siunitx_compound_quantity:nn { {#1} {#2} } {#3}
484   \group_end:
485 }
486 \cs_new_protected:Npn \__siunitx_range_aux:
487 {
488   \keys_set:nx { siunitx }
489   {
490     compound-exponents      = \l__siunitx_range_exp_tl ,
491     compound-pair-separator = { \exp_not:V \l_siunitx_range_phrase_tl } ,
492     compound-separator-mode = text ,
493     compound-units          = \l__siunitx_range_units_tl
494   }
495 }

```

(End definition for `\siunitx_number_range:nn`, `\siunitx_quantity_range:nnn`, and `__siunitx_range_aux:`. These functions are documented on page 19.)

1.5 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.

```

496 \keys_set:nn { siunitx }
497 {
498   compound-exponents      = individual ,
499   compound-final-separator = { ~ and ~ } ,
500   compound-pair-separator = { ~ and ~ } ,
501   compound-separator      = { , ~ } ,
502   compound-separator-mode = text ,
503   compound-units          = repeat ,
504   list-exponents          = individual ,
505   list-final-separator    = { ~ and ~ } ,
506   list-pair-separator     = { ~ and ~ } ,

```

```

507     list-separator      = { , ~ }      ,
508     list-units           = repeat        ,
509     product-exponents   = individual    ,
510     product-mode         = symbol        ,
511     product-phrase       = { ~ by ~ }   ,
512     product-symbol       = \times        ,
513     product-units        = repeat        ,
514     range-exponents      = individual    ,
515     range-phrase         = { ~ to ~ }   ,
516     range-units          = repeat
517   }
518 </package>

```

Part IV

siunitx-locale – Localisation

1 siunitx-locale 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_locale>
```

1.1 Locales

The basics for defining locales are easy: these are just meta keys.

```
3 \keys_define:nn { siunitx }
4 {
5   locale .choice: ,
6   locale / DE .meta:n =
7     {
8       exponent-product      = \cdot ,
9       inter-unit-product    = \, ,
10      output-decimal-marker = { , }
11    } ,
12   locale / FR .meta:n =
13     {
14       exponent-product      = \times ,
15       inter-unit-product    = \, ,
16       output-decimal-marker = { , }
17     } ,
18   locale / UK .meta:n =
19     {
20       exponent-product      = \times ,
21       inter-unit-product    = \, ,
22       output-decimal-marker = .
23     } ,
24   locale / US .meta:n =
25     {
26       exponent-product      = \times ,
27       inter-unit-product    = \, ,
28       output-decimal-marker = .
29     } ,
30   locale / ZA .meta:n =
31     {
32       exponent-product      = \times ,
33       inter-unit-product    = \cdot ,
34       output-decimal-marker = { , }
35     }
36 }
```

1.2 Localisation

Localisation makes use of the `translator` package. This only happens if it is available, and is transparent to the user.

```
37 \file_if_exist:nT { translations.sty }
38 {
39   \RequirePackage { translations }
40   \DeclareTranslation { English } { to~(numerical~range) } { to }
41   \DeclareTranslation { French } { to~(numerical~range) } { á }
42   \DeclareTranslation { German } { to~(numerical~range) } { bis }
43   \DeclareTranslation { Spanish } { to~(numerical~range) } { a }
44   \keys_set:nn { siunitx }
45   {
46     list-final-separator = { ~ \GetTranslation { and } ~ } ,
47     list-pair-separator  = { ~ \GetTranslation { and } ~ } ,
48     range-phrase         = { ~ \GetTranslation { to~(numerical~range) } ~ }
49   }
50 }
```

Part V

siunitx-number – Parsing and formatting numbers

1 Formatting numbers

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

`\siunitx_number_parse:nN` $\{\langle number \rangle\}$ $\langle tl\ var \rangle$

Parses the *number* and stores the resulting internal representation in the $\langle tl\ var \rangle$. The parsing is influenced by the various key–value settings for numerical input. The $\langle number \rangle$ 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 $\langle tl\ var \rangle$.

The structure of a valid number is:

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

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

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

`\siunitx_number_process:NN`

`\siunitx_number_process:N` $\langle tl\ var1 \rangle$ $\langle tl\ var2 \rangle$

Applies a set of number processing operations to the $\langle internal\ number \rangle$ stored in the $\langle tl\ var1 \rangle$, *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 $\langle tl\ var2 \rangle$.

<code>\siunitx_number_output:N</code>	☆	<code>\siunitx_number_output:N</code>	$\langle number \rangle$
<code>\siunitx_number_output:n</code>	☆	<code>\siunitx_number_output:NN</code>	$\langle number \rangle$ $\langle marker \rangle$
<code>\siunitx_number_output:NN</code>	☆		
<code>\siunitx_number_output:nN</code>	☆		

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.*

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

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

<code>\siunitx_number_format:nN</code>	<code>\siunitx_number_format:nN</code>	$\{\langle number \rangle\}$	$\langle tl\ var \rangle$
--	--	------------------------------	---------------------------

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

<code>\siunitx_number_adjust_exponent:Nn</code>	★	<code>\siunitx_number_adjust_exponent:Nn</code>	$\langle number \rangle$	$\{\langle fp\ expr \rangle\}$
<code>\siunitx_number_adjust_exponent:nn</code>	★			

Adjusts the exponent of the $\langle number \rangle$ (in internal format) by the $\langle fp\ expr \rangle$ and leaves the result in the input stream.

<code>\siunitx_number_normalize_symbols:N</code>	<code>\siunitx_number_normalize_symbols:N</code>	$\langle tl\ var \rangle$
--	--	---------------------------

Replaces all multi-token signs and comparators in the $\langle tl\ var \rangle$ with their single-token equivalents. Replaces any active hyphen tokens with non-active versions.

<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.

`\l_siunitx_bracket_ambiguous_bool`

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

`\l_siunitx_number_parse_bool`

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

`\l_siunitx_number_comparator_tl`

`\l_siunitx_number_exponent_tl`

`\l_siunitx_number_sign_tl`

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

`\l_siunitx_number_input_decimal_tl`

`\l_siunitx_number_output_decimal_tl`

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

1.1 Key-value options

The options defined by this submodule are available within the `\keys{siunitx}` tree.

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

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

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

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

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

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

`exponent-base` `exponent-base = <base>`

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

`exponent-product` `exponent-product = <symbol>`

`expression` `expression = <expression>`

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

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

2 siunitx-number implementation

Start the DocStrip guards.

¹ `$\langle *package \rangle$`

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

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

2.1 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`.)

2.2 Main formatting routine

`\l__siunitx_number_outputted_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_outputted_tl

```

(End definition for `\l__siunitx_number_outputted_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 38.)

`\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 38.)

`\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_outputted_tl
22     { \siunitx_number_output:N \l__siunitx_number_parsed_tl }
23   }
24   { \tl_set:Nn \l__siunitx_number_outputted_tl { \ensuremath {#1} } } }
25 \exp_args:NNNV \group_end:
26 \tl_set:Nn #2 \l__siunitx_number_outputted_tl
27 }

```

(End definition for `\siunitx_number_format:nN`. This function is documented on page 37.)

2.3 Parsing numbers

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

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

In this release, 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 all components must be given in braces. *All* of the components must be present in a stored number (*i.e.* at the end of parsing). The number must have at least one digit for both the $\langle\textit{integer}\rangle$ and $\langle\textit{exponent}\rangle$ parts.

A non-empty $\langle\textit{uncertainty}\rangle$ must contain one leading brace group containing an identifier, then zero or more brace groups which contain the uncertainty data. In this release, the known uncertainty types are

- **S**: A symmetrical statistical uncertainty made up of a single value. These are stored as uncertainty in significant digits, with no radix point in the stored value.

`\l_siunitx_number_input_decimal_tl`

The input decimal markers(s).

28 `\tl_new:N \l_siunitx_number_input_decimal_tl`

(End definition for `\l_siunitx_number_input_decimal_tl`. This variable is documented on page 38.)

`\l_siunitx_number_expression_bool`

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

`\l_siunitx_number_input_uncert_close_tl`

29 `\keys_define:nn { siunitx }`

30 `{`

31 `evaluate-expression .bool_set:N =`

32 `\l_siunitx_number_expression_bool ,`

33 `expression .code:n =`

34 `\cs_set:Npn _siunitx_number_expression:n ##1 {#1} ,`

35 `input-close-uncertainty .tl_set:N =`

36 `\l_siunitx_number_input_uncert_close_tl ,`

37 `input-comparators .tl_set:N =`

38 `\l_siunitx_number_input_comparator_tl ,`

39 `input-decimal-markers .tl_set:N =`

40 `\l_siunitx_number_input_decimal_tl ,`

41 `input-digits .tl_set:N =`

42 `\l_siunitx_number_input_digit_tl ,`

43 `input-exponent-markers .tl_set:N =`

44 `\l_siunitx_number_input_exponent_tl ,`

45 `input-ignore .tl_set:N =`

46 `\l_siunitx_number_input_ignore_tl ,`

```

47   input-open-uncertainty .tl_set:N =
48     \l__siunitx_number_input_uncert_open_tl ,
49   input-signs .tl_set:N =
50     \l__siunitx_number_input_sign_tl ,
51   input-uncertainty-signs .code:n =
52     {
53       \tl_set:Nn \l__siunitx_number_input_uncert_sign_tl {#1}
54       \tl_map_inline:nn {#1}
55       {
56         \tl_if_in:NnF \l__siunitx_number_input_sign_tl {##1}
57         { \tl_put_right:Nn \l__siunitx_number_input_sign_tl {##1} }
58       }
59     } ,
60   parse-numbers .bool_set:N =
61     \l__siunitx_number_parse_bool ,
62   retain-explicit-plus .bool_set:N =
63     \l__siunitx_number_explicit_plus_bool ,
64   retain-zero-uncertainty .bool_set:N =
65     \l__siunitx_number_zero_uncert_bool
66   }
67 \cs_new:Npn \__siunitx_number_expression:n #1 { }
68 \tl_new:N \l__siunitx_number_input_uncert_sign_tl

```

(End definition for \l__siunitx_number_expression_bool and others. These variables are documented on page ??.)

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

```
69 \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.

```
70 \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.

```
71 \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.

```
72 \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.

```
73 \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.

```
74 \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.

```
75 \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.

```
76 \bool_new:N \l__siunitx_number_validate_bool
```

(End definition for \l__siunitx_number_validate_bool.)

\siunitx_number_normalize_symbols:N 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.

```
77 \cs_new_protected:Npn \siunitx_number_normalize_symbols:N #1
78 {
79   \__siunitx_number_normalize_minus:N #1
80   \exp_after:wN \__siunitx_number_normalize_aux:NnN \exp_after:wN #1
81   \c__siunitx_number_normalize_tl
82   { ? } \q_recursion_tail
83   \q_recursion_stop
84 }
85 \cs_set_protected:Npn \__siunitx_number_normalize_aux:NnN #1#2#3
86 {
87   \quark_if_recursion_tail_stop:N #3
88   \tl_replace_all:Nnn #1 {#2} {#3}
89   \__siunitx_number_normalize_aux:NnN #1
90 }
91 \tl_const:Nn \c__siunitx_number_normalize_tl
92 {
93   { -+ } \mp
94   { +- } \pm
95   { << } \ll
96   { <= } \le
97   { >> } \gg
98   { >= } \ge
99 }
100 \group_begin:
101   \char_set_catcode_active:N \-
102   \cs_new_protected:Npx \__siunitx_number_normalize_minus:N #1
103   {
104     \tl_replace_all:Nnn #1
105     { \exp_not:N - } { \token_to_str:N - }
106   }
107 \group_end:
```

(End definition for \siunitx_number_normalize_symbols:N and others. This function is documented on page 37.)

`\siunitx_number_parse:nN`
`\siunitx_number_parse:VN`
`__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.

```

108 \cs_new_protected:Npn \siunitx_number_parse:nN #1#2
109 {
110   \bool_if:NTF \l_siunitx_number_parse_bool
111     { \__siunitx_number_parse:nN {#1} #2 }
112     { \tl_clear:N #2 }
113 }
114 \cs_generate_variant:Nn \siunitx_number_parse:nN { V }
115 \cs_new_protected:Npn \__siunitx_number_parse:nN #1#2
116 {
117   \group_begin:
118   \tl_clear:N \l__siunitx_number_parsed_tl
119   \protected@edef \l__siunitx_number_arg_tl
120     {
121       \bool_if:NTF \l__siunitx_number_expression_bool
122         { \fp_eval:n { \__siunitx_number_expression:n {#1} } }
123         {#1}
124     }
125   \tl_set_eq:NN \l__siunitx_number_input_tl \l__siunitx_number_arg_tl
126   \siunitx_number_normalize_symbols:N \l__siunitx_number_arg_tl
127   \tl_if_empty:NF \l__siunitx_number_arg_tl
128     { \__siunitx_number_parse_comparator: }
129   \__siunitx_number_parse_check:
130   \exp_args:NNNV \group_end:
131   \tl_set:Nn #2 \l__siunitx_number_parsed_tl
132 }

```

(End definition for `\siunitx_number_parse:nN` and `__siunitx_number_parse:nN`. This function is documented on page 36.)

`__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

```

133 \cs_new_protected:Npn \__siunitx_number_parse_check:
134 {
135   \tl_if_empty:NF \l__siunitx_number_flex_tl
136   {
137     \bool_lazy_and:nnTF
138       {
139         \tl_if_blank_p:f
140         { \exp_after:wN \use_iv:nnnn \l__siunitx_number_parsed_tl }
141       }
142       {
143         \tl_if_blank_p:f
144         { \exp_after:wN \use_iv:nnnn \l__siunitx_number_flex_tl }
145       }
146       {
147         \tl_set:Nx \l__siunitx_number_tmp_tl
148           { \exp_after:wN \use_i:nnnn \l__siunitx_number_flex_tl }
149         \tl_if_in:NVTF \l__siunitx_number_input_uncert_sign_tl
150           \l__siunitx_number_tmp_tl

```

```

151         { \_siunitx_number_parse_combine_uncert: }
152         { \tl_clear:N \l__siunitx_number_parsed_tl }
153     }
154     { \tl_clear:N \l__siunitx_number_parsed_tl }
155 }
156 \tl_if_empty:NTF \l__siunitx_number_parsed_tl
157 {
158     \bool_if:NF \l__siunitx_number_validate_bool
159     {
160         \msg_error:nxx { siunitx } { invalid-number }
161         { \exp_not:V \l__siunitx_number_input_tl }
162     }
163 }
164 { \_siunitx_number_parse_finalise: }
165 }

```

(End definition for _siunitx_number_parse_check:.)

_siunitx_number_parse_combine_uncert:
x_number_parse_combine_uncert_auxi:nnnnnnnn
itx_number_parse_combine_uncert_auxii:nnnnn
itx_number_parse_combine_uncert_auxiii:fnnnn
x_number_parse_combine_uncert_auxiii:nnnnnn
x_number_parse_combine_uncert_auxiii:fnnnn
_siunitx_number_parse_combine_uncert_auxiv: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.

```

166 \cs_new_protected:Npn \_siunitx_number_parse_combine_uncert:
167 {
168     \exp_after:wN \exp_after:wN \exp_after:wN
169     \_siunitx_number_parse_combine_uncert_auxi:nnnnnnnn
170     \exp_after:wN \l__siunitx_number_parsed_tl \l__siunitx_number_flex_tl
171 }

```

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.

```

172 \cs_new_protected:Npn
173 \_siunitx_number_parse_combine_uncert_auxi:nnnnnnnn #1#2#3#4#5#6#7#8
174 {
175     \int_compare:nNnTF { \tl_count:n {#6} } > { \tl_count:n {#2} }
176     {
177         \tl_clear:N \l__siunitx_number_parsed_tl
178         \tl_clear:N \l__siunitx_number_flex_tl
179     }
180     {
181         \_siunitx_number_parse_combine_uncert_auxii:fnnnn
182         { \int_eval:n { \tl_count:n {#3} - \tl_count:n {#7} } }
183         {#2} {#3} {#6} {#7}
184     }
185 }
186 \cs_new_protected:Npn
187 \_siunitx_number_parse_combine_uncert_auxii:nnnnn #1
188 {
189     \_siunitx_number_parse_combine_uncert_auxiii:fnnnnn

```

```

190     { \prg_replicate:nn { \int_abs:n {#1} } { 0 } }
191     {#1}
192   }
193 \cs_generate_variant:Nn \__siunitx_number_parse_combine_uncert_auxii:nnnnn { f }
194 \cs_new_protected:Npn
195   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn #1#2#3#4#5#6
196   {
197     \int_compare:nNnTF {#2} > 0
198     {
199       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
200       {#3} {#4} {#5} { #6 #1 }
201     }
202     {
203       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
204       {#3} { #4 #1 } {#5} {#6}
205     }
206   }
207 \cs_generate_variant:Nn
208   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn { f }
209 \cs_new_protected:Npn
210   \__siunitx_number_parse_combine_uncert_auxiv:nnnn #1#2#3#4
211   {
212     \tl_set:Nx \l__siunitx_number_parsed_tl
213     {
214       { \tl_head:V \l__siunitx_number_parsed_tl }
215       { \exp_not:n {#1} }
216     }
217     {
218       \bool_lazy_and:nnTF
219       { \tl_if_blank_p:n {#2} }
220       { ! \tl_if_blank_p:n {#4} }
221       { 0 }
222       { \exp_not:n {#2} }
223     }
224     {
225       \__siunitx_number_parse_combine_uncert_auxv:w #3#4
226       \q_recursion_tail \q_recursion_stop
227     }
228   }

```

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

```

229 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxv:w #1
230   {
231     \quark_if_recursion_tail_stop_do:Nn #1
232     {
233       \bool_if:NT \l__siunitx_number_zero_uncert_bool
234       { { S } { 0 } }
235     }
236     \str_if_eq:nnTF {#1} { 0 }
237     { \__siunitx_number_parse_combine_uncert_auxv:w }
238     { \__siunitx_number_parse_combine_uncert_auxvi:w #1 }
239   }
240 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxvi:w

```

```

241   #1 \q_recursion_tail \q_recursion_stop
242   { { S } { \exp_not:n {#1} } } }

```

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

`_siunitx_number_parse_comparator:`
`_siunitx_number_parse_comparator_aux:Nw`

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

```

243 \cs_new_protected:Npn \_siunitx_number_parse_comparator:
244 {
245   \exp_after:wN \_siunitx_number_parse_comparator_aux:Nw
246   \l_siunitx_number_arg_tl \q_stop
247 }
248 \cs_new_protected:Npn \_siunitx_number_parse_comparator_aux:Nw #1#2 \q_stop
249 {
250   \tl_if_in:NnTF \l_siunitx_number_input_comparator_tl {#1}
251   {
252     \tl_set:Nn \l_siunitx_number_comparator_tl {#1}
253     \tl_set:Nn \l_siunitx_number_arg_tl {#2}
254   }
255   { \tl_clear:N \l_siunitx_number_comparator_tl }
256   \tl_if_empty:NF \l_siunitx_number_arg_tl
257   { \_siunitx_number_parse_sign: }
258 }

```

(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.

```

259 \cs_new_protected:Npn \_siunitx_number_parse_exponent:
260 {
261   \tl_if_empty:NnTF \l_siunitx_number_input_exponent_tl
262   {
263     \tl_set:Nn \l_siunitx_number_exponent_tl { { } 0 }
264     \tl_if_empty:NF \l_siunitx_number_parsed_tl
265     { \_siunitx_number_parse_loop: }
266   }
267   {
268     \tl_set:Nx \l_siunitx_number_tmp_tl
269     { \tl_head:V \l_siunitx_number_input_exponent_tl }
270     \tl_map_inline:Nn \l_siunitx_number_input_exponent_tl
271     {
272       \tl_replace_all:NnV \l_siunitx_number_arg_tl
273       {##1} \l_siunitx_number_tmp_tl
274     }

```

```

275 \use:x
276 {
277   \cs_set_protected:Npn
278     \exp_not:N \__siunitx_number_parse_exponent_auxi:w
279     #####1 \exp_not:V \l__siunitx_number_tmp_tl
280     #####2 \exp_not:V \l__siunitx_number_tmp_tl
281     #####3 \exp_not:N \q_stop
282 }
283 { \__siunitx_number_parse_exponent_auxii:nn {##1} {##2} }
284 \use:x
285 {
286   \__siunitx_number_parse_exponent_auxi:w
287   \exp_not:V \l__siunitx_number_arg_tl
288   \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_nil
289   \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_stop
290 }
291 }
292 }
293 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxi:w { }
294 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxii:nn #1#2
295 {
296   \quark_if_nil:nTF {#2}
297   { \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 } }
298   {
299     \tl_set:Nn \l__siunitx_number_arg_tl {#1}
300     \tl_if_blank:nTF {#2}
301     { \tl_clear:N \l__siunitx_number_parsed_tl }
302     { \__siunitx_number_parse_exponent_auxiii:Nw #2 \q_stop }
303   }
304   \tl_if_empty:NF \l__siunitx_number_parsed_tl
305   { \__siunitx_number_parse_loop: }
306 }
307 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxiii:Nw #1#2 \q_stop
308 {
309   \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#1}
310   { \__siunitx_number_parse_exponent_auxiv:nn {#1} {#2} }
311   { \__siunitx_number_parse_exponent_auxiv:nn { } {#1#2} }
312   \tl_if_empty:NT \l__siunitx_number_exponent_tl
313   { \tl_clear:N \l__siunitx_number_parsed_tl }
314 }
315 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxiv:nn #1#2
316 {
317   \bool_lazy_or:nnTF
318   { \l__siunitx_number_explicit_plus_bool }
319   { ! \str_if_eq_p:nn {#1} { + } }
320   { \tl_set:Nn \l__siunitx_number_exponent_tl { {#1} } }
321   { \tl_set:Nn \l__siunitx_number_exponent_tl { { } } }
322   \tl_if_blank:nTF {#2}
323   { \tl_clear:N \l__siunitx_number_parsed_tl }
324   {
325     \__siunitx_number_parse_exponent_zero_test:N #2
326     \q_recursion_tail \q_recursion_stop
327   }
328 }

```

```

329 \cs_new_protected:Npn \__siunitx_number_parse_exponent_zero_test:N #1
330 {
331   \quark_if_recursion_tail_stop_do:Nn #1
332   { \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 } }
333   \str_if_eq:nnTF {#1} { 0 }
334   { \__siunitx_number_parse_exponent_zero_test:N }
335   { \__siunitx_number_parse_exponent_check:N #1 }
336 }
337 \cs_new_protected:Npn \__siunitx_number_parse_exponent_check:N #1
338 {
339   \quark_if_recursion_tail_stop:N #1
340   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#1}
341   {
342     \tl_put_right:Nn \l__siunitx_number_exponent_tl {#1}
343     \__siunitx_number_parse_exponent_check:N
344   }
345   { \__siunitx_number_parse_exponent_cleanup:wN }
346 }
347 \cs_new_protected:Npn \__siunitx_number_parse_exponent_cleanup:wN
348 #1 \q_recursion_stop
349 { \tl_clear:N \l__siunitx_number_parsed_tl }

```

(End definition for __siunitx_number_parse_exponent: and others.)

__siunitx_number_parse_finalise:
__siunitx_number_parse_finalise:nw

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

```

350 \cs_new_protected:Npn \__siunitx_number_parse_finalise:
351 {
352   \tl_if_empty:NF \l__siunitx_number_parsed_tl
353   {
354     \tl_set:Nx \l__siunitx_number_parsed_tl
355     {
356       { \exp_not:V \l__siunitx_number_comparator_tl }
357       \exp_not:V \l__siunitx_number_parsed_tl
358       \exp_after:wN \__siunitx_number_parse_finalise:nw
359       \l__siunitx_number_exponent_tl \q_stop
360     }
361   }
362 }
363 \cs_new:Npn \__siunitx_number_parse_finalise:nw #1#2 \q_stop
364 {
365   { \exp_not:n {#1} }
366   { \exp_not:n {#2} }
367 }

```

(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:NNNN
__siunitx_number_parse_loop_main_end:NN
__siunitx_number_parse_loop_main_digit:NNNN
__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:NNNN
__siunitx_number_parse_loop_after_uncert:N
__siunitx_number_parse_loop_root_swap:NNwNN
__siunitx_number_parse_loop_break:wN

```

368 \cs_new_protected:Npn \__siunitx_number_parse_loop:

```

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.

```

369 {
370   \tl_clear:N \l__siunitx_number_partial_tl
371   \exp_after:wN \__siunitx_number_parse_loop_first:NNN
372   \exp_after:wN \l__siunitx_number_parsed_tl \exp_after:wN \c_true_bool
373   \l__siunitx_number_arg_tl
374   \q_recursion_tail \q_recursion_stop
375 }

```

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.

```

376 \cs_new_protected:Npn \__siunitx_number_parse_loop_first:NNN #1#2#3
377 {
378   \quark_if_recursion_tail_stop_do:Nn #3
379   {
380     \bool_if:NTF #2
381     { \tl_put_right:Nn #1 { { 1 } { } { } } }
382     { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
383   }
384   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#3}
385   {
386     \__siunitx_number_parse_loop_main:NNNNN
387     #1 \c_true_bool \c_false_bool #2 #3
388   }
389   {
390     \tl_if_in:NnTF \l__siunitx_number_input_decimal_tl {#3}
391     {
392       \tl_put_right:Nn #1 { { 0 } }
393       \__siunitx_number_parse_loop_after_decimal:NNN #1 #2
394     }
395     { \__siunitx_number_parse_loop_break:wN }
396   }
397 }

```

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.

```

398 \cs_new_protected:Npn \__siunitx_number_parse_loop_main:NNNNN #1#2#3#4#5
399 {
400   \quark_if_recursion_tail_stop_do:Nn #5
401   { \__siunitx_number_parse_loop_main_end:NN #1#2 }
402   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#5}
403   { \__siunitx_number_parse_loop_main_digit:NNNNN #1#2#3#4#5 }
404   {
405     \tl_if_in:NnTF \l__siunitx_number_input_decimal_tl {#5}
406     {
407       \bool_if:NTF #2
408       { \__siunitx_number_parse_loop_main_decimal:NN #1 #4 }
409       { \__siunitx_number_parse_loop_break:wN }
410     }
411     {
412       \tl_if_in:NnTF \l__siunitx_number_input_uncert_open_tl {#5}
413       { \__siunitx_number_parse_loop_main_uncert:NNN #1#2 #4 }
414       {
415         \bool_if:NTF #4
416         {
417           \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#5}
418           {
419             \__siunitx_number_parse_loop_main_sign:NNN
420             #1#2 #5
421           }
422           { \__siunitx_number_parse_loop_break:wN }
423         }
424         { \__siunitx_number_parse_loop_break:wN }
425       }
426     }
427   }
428 }

```

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.

```

429 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_end:NN #1#2
430 {
431   \bool_lazy_and:nnT
432   {#2} { \tl_if_empty_p:N \l__siunitx_number_partial_tl }
433   { \tl_set:Nn \l__siunitx_number_partial_tl { 0 } }
434   \tl_put_right:Nx #1
435   {
436     { \exp_not:V \l__siunitx_number_partial_tl }
437     \bool_if:NT #2 { { } }
438     { }

```



```

439     }
440 }

```

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.

```

441 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_digit:NNNNN #1#2#3#4#5
442 {
443   \bool_lazy_or:nnTF
444     {#3} { ! \str_if_eq_p:nn {#5} { 0 } }
445   {
446     \tl_put_right:Nn \l__siunitx_number_partial_tl {#5}
447     \__siunitx_number_parse_loop_main:NNNNN #1 #2 \c_true_bool #4
448   }
449   { \__siunitx_number_parse_loop_main:NNNNN #1 #2 \c_false_bool #4 }
450 }

```

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.

```

451 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_decimal:NN #1#2
452 {
453   \__siunitx_number_parse_loop_main_store:NNN #1 \c_false_bool \c_false_bool
454   \__siunitx_number_parse_loop_after_decimal:NNN #1 #2
455 }

```

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.

```

456 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_uncert:NNN #1#2#3
457 {
458   \__siunitx_number_parse_loop_main_store:NNN #1 #2 \c_false_bool
459   \__siunitx_number_parse_loop_uncert:NNNNN
460   #1 \c_true_bool \c_false_bool #3
461 }

```

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

```

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

```

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

```

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

```

```

474     {
475         { \exp_not:V \l__siunitx_number_partial_tl }
476         \bool_if:NT #2 { { } }
477         \bool_if:NT #3 { { } }
478     }
479     \tl_clear:N \l__siunitx_number_partial_tl
480 }

```

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

```

481 \cs_new_protected:Npn \__siunitx_number_parse_loop_after_decimal:NNN #1#2#3
482 {
483     \tl_if_blank:FTF { \exp_after:wN \use_none:n #1 }
484     {
485         \quark_if_recursion_tail_stop_do:Nn #3
486         { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
487         \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#1}
488         {
489             \tl_put_right:Nn \l__siunitx_number_partial_tl {#3}
490             \__siunitx_number_parse_loop_main:NNNNN
491             #1 \c_false_bool \c_true_bool #2
492         }
493         { \__siunitx_number_parse_loop_break:wN }
494     }
495     {
496         \__siunitx_number_parse_loop_main:NNNNN
497         #1 \c_false_bool \c_true_bool #2 #3
498     }
499 }

```

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

```

500 \cs_new_protected:Npn \__siunitx_number_parse_loop_uncert:NNNNN #1#2#3#4#5
501 {
502     \quark_if_recursion_tail_stop_do:Nn #5
503     { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
504     \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#5}
505     {
506         \bool_lazy_or:nnTF
507         {#3} { ! \str_if_eq_p:nn {#5} { 0 } }
508         {
509             \tl_put_right:Nn \l__siunitx_number_partial_tl {#5}
510             \__siunitx_number_parse_loop_uncert:NNNNN
511             #1 \c_false_bool \c_true_bool #4
512         }
513         {
514             \__siunitx_number_parse_loop_uncert:NNNNN
515             #1 \c_false_bool \c_false_bool #4
516         }
517     }
518 }

```

```

519 \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#5}
520 {
521   \bool_if:NTF #2
522   { \__siunitx_number_parse_loop_break:wN }
523   {
524     \tl_if_empty:NTF \l__siunitx_number_partial_tl
525     {
526       \tl_put_right:Nx #1
527       {
528         {
529           \bool_if:NT \l__siunitx_number_zero_uncert_bool
530           { { S } { 0 } }
531         }
532       }
533     }
534     {
535       \tl_set:Nx \l__siunitx_number_partial_tl
536       { { S } { \exp_not:V \l__siunitx_number_partial_tl } }
537       \__siunitx_number_parse_loop_main_store:NNN #1
538       \c_false_bool \c_false_bool
539     }
540     \__siunitx_number_parse_loop_after_uncert:N
541   }
542 }
543 { \__siunitx_number_parse_loop_break:wN }
544 }
545 }

```

No further tokens are allowed after an uncertainty in parenthesis.

```

546 \cs_new_protected:Npn \__siunitx_number_parse_loop_after_uncert:N #1
547 {
548   \quark_if_recursion_tail_stop:N #1
549   \__siunitx_number_parse_loop_break:wN
550 }

```

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.

```

551 \cs_new_protected:Npn \__siunitx_number_parse_loop_break:wN
552 #1 \q_recursion_stop
553 {
554   \tl_clear:N \l__siunitx_number_flex_tl
555   \tl_clear:N \l__siunitx_number_parsed_tl
556 }

```

(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.

```

557 \cs_new_protected:Npn \__siunitx_number_parse_sign:
558 {
559   \exp_after:wN \__siunitx_number_parse_sign_aux:Nw
560   \l__siunitx_number_arg_tl \q_stop
561 }
562 \cs_new_protected:Npn \__siunitx_number_parse_sign_aux:Nw #1#2 \q_stop

```

```

563 {
564   \tl_if_in:NnTF \l_siunitx_number_input_sign_tl {#1}
565   {
566     \tl_set:Nn \l__siunitx_number_arg_tl {#2}
567     \bool_lazy_and:nnTF
568     { \token_if_eq_charcode_p:NN #1 + }
569     { ! \l__siunitx_number_explicit_plus_bool }
570     { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
571     { \tl_set:Nn \l__siunitx_number_parsed_tl { {#1} } }
572   }
573   { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
574   \tl_if_empty:NTF \l__siunitx_number_arg_tl
575   { \tl_clear:N \l__siunitx_number_parsed_tl }
576   { \__siunitx_number_parse_exponent: }
577 }

```

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

2.4 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_mode_tl
\l__siunitx_number_round_pad_bool
\l__siunitx_number_round_precision_int
578 \keys_define:nn { siunitx }
579 {
580   drop-exponent .bool_set:N =
581     \l__siunitx_number_drop_exponent_bool ,
582   drop-uncertainty .bool_set:N =
583     \l__siunitx_number_drop_uncertainty_bool ,
584   drop-zero-decimal .bool_set:N =
585     \l__siunitx_number_drop_zero_decimal_bool ,
586   exponent-mode .choices:nn =
587     { engineering , fixed , input , scientific }
588     { \tl_set_eq:NN \l__siunitx_number_exponent_mode_tl \l_keys_choice_tl } ,
589   fixed-exponent .int_set:N =
590     \l__siunitx_number_exponent_fixed_int ,
591   minimum-decimal-digits .int_set:N =
592     \l__siunitx_number_min_decimal_int ,
593   minimum-integer-digits .int_set:N =
594     \l__siunitx_number_min_integer_int ,
595   round-half .choice: ,
596   round-half / even .code:n =
597     { \bool_set_true:N \l__siunitx_number_round_half_even_bool } ,
598   round-half / up .code:n =
599     { \bool_set_false:N \l__siunitx_number_round_half_even_bool } ,
600   round-minimum .code:n =
601     { \__siunitx_number_set_round_min:n {#1} } ,
602   round-mode .choices:nn =
603     { figures , none , places , uncertainty }
604     { \tl_set_eq:NN \l__siunitx_number_round_mode_tl \l_keys_choice_tl } ,
605   round-pad .bool_set:N =
606     \l__siunitx_number_round_pad_bool ,
607   round-precision .int_set:N =
608     \l__siunitx_number_round_precision_int ,
609 }
610 \bool_new:N \l__siunitx_number_round_half_even_bool

```

```

611 \tl_new:N \l__siunitx_number_exponent_mode_tl
612 \tl_new:N \l__siunitx_number_round_mode_tl

(End definition for \l__siunitx_number_drop_exponent_bool and others.)

```

\l__siunitx_number_round_min_tl For storing the minimum for rounding.

```

613 \tl_new:N \l__siunitx_number_round_min_tl

(End definition for \l__siunitx_number_round_min_tl.)

```

__siunitx_number_set_round_min:n For setting the rounding minimum, the aim is to do as much of the work now as possible. That's mainly a question of checking if there are any significant digits in the mantissa given.

```

614 \cs_new_protected:Npn \__siunitx_number_set_round_min:n #1
615 {
616   \siunitx_number_parse:nN {#1} \l__siunitx_number_tmp_tl
617   \exp_after:wN \__siunitx_number_set_round_min:nnnnnnn \l__siunitx_number_tmp_tl
618 }
619 \cs_new:Npn \__siunitx_number_set_round_min:nnnnnnn #1#2#3#4#5#6#7
620 {
621   \tl_set:Nx \l__siunitx_number_round_min_tl
622   {
623     \bool_lazy_and:nnF
624     { \str_if_eq_p:nn {#3} { 0 } }
625     {
626       \str_if_eq_p:ee
627       { \exp_not:n {#4} }
628       { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
629     }
630     { \exp_not:n { {#3} {#4} } }
631   }
632 }

```

(End definition for __siunitx_number_set_round_min:n and __siunitx_number_set_round_min:nnnnnnn.)

\siunitx_number_process:NN A top-level interface for the processing tools.

```

\__siunitx_number_process:nnnnnnnNN
633 \cs_new_protected:Npn \siunitx_number_process:NN #1#2
634 {
635   \tl_if_empty:NTF #1
636   { \tl_clear:N #2 }
637   {
638     \__siunitx_number_drop_uncertainty:NN #1 #2
639     \exp_after:wN \__siunitx_number_process:nnnnnnnNN #2 #2 #2
640     \__siunitx_number_drop_exponent:NN #2 #2
641     \__siunitx_number_zero_decimal:NN #2 #2
642     \__siunitx_number_digits:NN #2 #2
643   }
644 }
645 \cs_new_protected:Npn \__siunitx_number_process:nnnnnnnNN #1#2#3#4#5#6#7#8#9
646 {
647   \bool_lazy_and:nnF
648   { \str_if_eq_p:nn {#3} { 0 } }
649   {
650     \str_if_eq_p:ee
651     { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }

```

```

652     }
653     {
654         \_siunitx_number_exponent:NN #8 #9
655         \_siunitx_number_round:NN #9 #9
656     }
657 }

```

(End definition for \siunitx_number_process:NN and _siunitx_number_process:nnnnnnnNN. This function is documented on page 36.)

```

\_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_fixed:nnnnnnnn
siunitx_number_exponent_scientific:nnnnnnnn
\_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_shift_up_aux:nnn
\_siunitx_number_exponent_shift_up_aux:fnw
\_siunitx_number_exponent_shift_up_aux:ffn
\_siunitx_number_exponent_shift_uncert:nw
siunitx_number_exponent_shift_uncert_S:nnnn
siunitx_number_exponent_shift_uncert_S:fnnn
\_siunitx_number_exponent_uncert:n
\_siunitx_number_exponent_finalise:n
itx_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:nnNw
siunitx_number_exponent_engineering_uncert:nn
tx_number_exponent_engineering_uncert_S:nnn

```

```

658 \cs_new_protected:Npn \_siunitx_number_exponent:NN #1#2
659 {
660     \tl_set:Nx #2
661     {
662         \cs:w
663         \_siunitx_number_exponent_ \l__siunitx_number_exponent_mode_tl :nnnnnnnn
664         \exp_after:wN
665         \cs_end: #1
666     }
667     \str_if_eq:VnT \l__siunitx_number_exponent_mode_tl { engineering }
668     {
669         \tl_set:Nx #2
670         { \exp_after:wN \_siunitx_number_exponent_engineering_aux:nnnnnnn #2 }
671     }
672 }
673 \cs_new:Npn \_siunitx_number_exponent_fixed:nnnnnnnn #1#2#3#4#5#6#7
674 {
675     \exp_args:Nf \_siunitx_number_exponent_fixed:nnnnnnnn
676     { \int_eval:n { \l__siunitx_number_exponent_fixed_int - (#6#7) } }
677     {#1} {#2} {#3} {#4} {#5} {#6} {#7}
678 }
679 \cs_new:Npn \_siunitx_number_exponent_fixed:nnnnnnnn #1#2#3#4#5#6#7#8
680 {
681     \exp_not:n { {#2} {#3} }
682     \_siunitx_number_exponent_shift:nnn {#1} {#4} {#5}
683     \_siunitx_number_exponent_uncert:n {#6}
684     \exp_not:n { {#7} } { \int_use:N \l__siunitx_number_exponent_fixed_int }
685 }
686 \cs_new:Npn \_siunitx_number_exponent_input:nnnnnnnn #1#2#3#4#5#6#7
687 { \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.

```

688 \cs_new:Npn \_siunitx_number_exponent_scientific:nnnnnnnn #1#2#3#4#5#6#7
689 {
690     \exp_args:Nf \_siunitx_number_exponent_scientific:nnnnnnnn
691     { \int_eval:n { \tl_count:n {#3} } }
692     {#1} {#2} {#3} {#4} {#5} {#6} {#7}
693 }

```

```

694 \cs_new:Npn \__siunitx_number_exponent_scientific:nnnnnnnn #1#2#3#4#5#6#7#8
695 {
696   \exp_not:n { {#2} {#3} }
697   \int_compare:nNnTF {#1} = 1
698   {
699     \str_if_eq:nnTF {#4} { 0 }
700     {
701       \__siunitx_number_exponent_scientific:nnnw
702       { 0 } {#6} { #7#8 } #5 \q_stop
703     }
704     { \exp_not:n { {#4} {#5} {#6} {#7} {#8} } }
705   }
706   {
707     \__siunitx_number_exponent_shift:nnn { #1 - 1 } {#4} {#5}
708     \__siunitx_number_exponent_uncert:n {#6}
709     \__siunitx_number_exponent_finalise:n { #1 + #7#8 - 1 }
710   }
711 }
712 \cs_new_eq:NN \__siunitx_number_exponent_engineering:nnnnnnnn
713 \__siunitx_number_exponent_scientific:nnnnnnnn
714 \cs_new:Npn \__siunitx_number_exponent_scientific:nnnw #1#2#3#4#5 \q_stop
715 {
716   \str_if_eq:nnTF {#4} { 0 }
717   {
718     \__siunitx_number_exponent_scientific:nnnw
719     { #1 - 1 } {#2} {#3} #5 \q_stop
720   }
721   {
722     \exp_not:n { {#4} {#5} {#2} }
723     \__siunitx_number_exponent_finalise:n { #1 + #3 - 1 }
724   }
725 }

```

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

```

726 \cs_new:Npn \__siunitx_number_exponent_shift:nnn #1#2#3
727 {
728   \int_compare:nNnTF {#1} > 0
729   { \__siunitx_number_exponent_shift_down:nnnw {#1} {#3} { } #2 \q_stop }
730   {
731     \int_compare:nNnTF {#1} < 0
732     { \__siunitx_number_exponent_shift_up:nnn {#1} {#2} {#3} }
733     { {#2} {#3} }
734   }
735 }
736 \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.

```

737 \cs_new:Npn \__siunitx_number_exponent_shift_down:nnnw #1#2#3#4#5 \q_stop
738 {
739   \tl_if_blank:nTF {#5}
740   { \__siunitx_number_exponent_shift_down:nnn {#1} { #4 #3 } {#2} }
741   { \__siunitx_number_exponent_shift_down:nnnw {#1} {#2} { #4 #3 } #5 \q_stop }

```

```

742 }
743 \cs_new:Npn \__siunitx_number_exponent_shift_down:nnn #1#2#3
744 {
745   \int_compare:nNnTF {#1} = 0
746     { { \tl_reverse:n {#2} } \exp_not:n { {#3} } }
747     { \__siunitx_number_exponent_shift_down:nw {#1} #2 \q_stop {#3} }
748 }
749 \cs_new:Npn \__siunitx_number_exponent_shift_down:nw #1#2#3 \q_stop #4
750 {
751   \tl_if_blank:nTF {#3}
752     { \__siunitx_number_exponent_shift_down:nnn { #1 - 1 } { 0 } { #2#4 } }
753     { \__siunitx_number_exponent_shift_down:nnn { #1 - 1 } {#3} { #2#4 } }
754 }

```

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. We also need to deal with leading zeros: these cannot accumulate.

```

755 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnn #1#2#3
756 {
757   \tl_if_blank:nTF {#3}
758   {
759     \__siunitx_number_exponent_shift_up_aux:ffn
760     { \int_eval:n { #1 + 1 } }
761     { \str_if_eq:nnF {#2} { 0 } {#2} 0 }
762     { }
763     \__siunitx_number_exponent_shift_uncert:nw { 1 }
764   }
765   { \__siunitx_number_exponent_shift_up:nnw {#1} {#2} #3 \q_stop }
766 }
767 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnw #1#2#3#4 \q_stop
768 {
769   \__siunitx_number_exponent_shift_up_aux:ffn
770   { \int_eval:n { #1 + 1 } }
771   { \str_if_eq:nnF {#2} { 0 } {#2} #3 }
772   {#4}
773 }
774 \cs_new:Npn \__siunitx_number_exponent_shift_up_aux:nnn #1#2#3
775 {
776   \int_compare:nNnTF {#1} = 0
777     { \exp_not:n { {#2} {#3} } }
778     {
779       \tl_if_blank:nTF {#3}
780       {
781         {
782           \exp_not:n {#2}
783           \prg_replicate:nn { \int_abs:n {#1} } { 0 }
784         }
785         { }
786         \__siunitx_number_exponent_shift_uncert:nw { \int_abs:n {#1} }
787       }
788       { \__siunitx_number_exponent_shift_up:nnn {#1} {#2} {#3} }
789     }
790 }
791 \cs_generate_variant:Nn \__siunitx_number_exponent_shift_up_aux:nnn { f , ff }

```


If the shift has put digits into the integer part, we have to adjust the uncertainty accordingly. First, we grab the data, then adjust by the number of places that have been transferred.

```

792 \cs_new:Npn \__siunitx_number_exponent_shift_uncert:nw
793   #1#2 \__siunitx_number_exponent_uncert:n #3
794   {
795     \tl_if_blank:nTF {#3}
796     {
797       #2
798       \__siunitx_number_exponent_uncert:n { }
799     }
800     {
801       \str_if_eq:nnTF {#3} { 0 }
802       {
803         #2
804         \__siunitx_number_exponent_uncert:n { { S } { 0 } }
805       }
806       {
807         \use:c { \__siunitx_number_exponent_shift_uncert_ \use_i:nn #3 :fnnn }
808         { \prg_replicate:nn {#1} { 0 } }
809         {#2}
810         #3
811       }
812     }
813   }
814 \cs_new:Npn \__siunitx_number_exponent_shift_uncert_S:nnnn #1#2#3#4
815   {
816     #2
817     \__siunitx_number_exponent_uncert:n { { S } { #4#1 } }
818   }
819 \cs_generate_variant:Nn \__siunitx_number_exponent_shift_uncert_S:nnnn { f }
820 \cs_new:Npn \__siunitx_number_exponent_uncert:n #1 { { \exp_not:n {#1} } }

```

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

```

821 \cs_new:Npn \__siunitx_number_exponent_finalise:n #1
822   {
823     \int_compare:nNnTF {#1} < 0
824     { { - } }
825     { { } }
826     { \int_abs:n {#1} }
827   }

```

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.

```

828 \cs_new:Npn \__siunitx_number_exponent_engineering_aux:nnnnnnn #1#2#3#4#5#6#7
829   {
830     \exp_not:n { {#1} {#2} }
831     \use:c
832     {
833       \__siunitx_number_exponent_engineering_
834       \int_compare:nNnTF {#6#7} < 0
835       {

```

```

836         \int_case:nnF { \int_mod:nn { #7 } { 3 } }
837         {
838             { 1 } { 2 }
839             { 2 } { 1 }
840         }
841         { 0 }
842     }
843     { \int_mod:nn {#7} { 3 } }
844     :nnnn
845 }
846     {#3} {#4} {#5} {#6#7}
847 }
848 \cs_new:cpn { __siunitx_number_exponent_engineering_0:nnnn } #1#2#3#4
849 {
850     \exp_not:n { {#1} {#2} {#3} }
851     \__siunitx_number_exponent_finalise:n {#4}
852 }
853 \cs_new:cpn { __siunitx_number_exponent_engineering_1:nnnn } #1#2#3#4
854 {
855     \tl_if_blank:nTF {#2}
856     {
857         { \exp_not:n { #1 0 } } { }
858         { \__siunitx_number_exponent_engineering_uncert:nn {#3} { 0 } }
859     }
860     {
861         { \exp_not:n {#1} \exp_not:o { \tl_head:w #2 \q_stop } }
862         { \exp_not:f { \tl_tail:n {#2} } }
863         { \exp_not:n {#3} }
864     }
865     \__siunitx_number_exponent_finalise:n { #4 - 1 }
866 }
867 \cs_new:cpn { __siunitx_number_exponent_engineering_2:nnnn } #1#2#3#4
868 {
869     \tl_if_blank:nTF {#2}
870     {
871         { \exp_not:n { #1 00 } } { }
872         { \__siunitx_number_exponent_engineering_uncert:nn {#3} { 00 } }
873     }
874     { \__siunitx_number_exponent_engineering:nnNw {#1} {#3} #2 \q_stop }
875     \__siunitx_number_exponent_finalise:n { #4 - 2 }
876 }
877 \cs_new:Npn \__siunitx_number_exponent_engineering:nnNw #1#2#3#4 \q_stop
878 {
879     \tl_if_blank:nTF {#4}
880     {
881         { \exp_not:n { #1#3 0 } } { }
882         { { \__siunitx_number_exponent_engineering_uncert:nn {#2} { 0 } } }
883     }
884     {
885         { \exp_not:n {#1#3} \exp_not:o { \tl_head:w #4 \q_stop } }
886         { \exp_not:f { \tl_tail:n {#4} } }
887         { \exp_not:n {#2} }
888     }
889 }

```

```

890 \cs_new:Npn \__siunitx_number_exponent_engineering_uncert:nn #1#2
891 {
892   \tl_if_blank:nF {#1}
893   {
894     \use:c { __siunitx_number_exponent_engineering_uncert_ \use_i:nn #1 :nnn }
895     #1 {#2}
896   }
897 }
898 \cs_new:Npn \__siunitx_number_exponent_engineering_uncert_S:nnn #1#2#3
899 {
900   { S }
901   {
902     \exp_not:n {#2}
903     \str_if_eq:nnF {#2} { 0 } {#3}
904   }
905 }

```

(End definition for __siunitx_number_exponent:NN and others.)

```

\__siunitx_number_digits:NN
  \__siunitx_number_digits:nnnnnnn
\__siunitx_number_digits:Nn

```

Forcing a minimum number of digits in each part is quite easy. As the common case is that we don't do anything here, there is no real need to optimise the calculation (normally also numbers have only a few digits).

```

906 \cs_new_protected:Npn \__siunitx_number_digits:NN #1#2
907 {
908   \tl_set:Nx #2
909   { \exp_after:wN \__siunitx_number_digits:nnnnnnn #1 }
910 }
911 \cs_new:Npn \__siunitx_number_digits:nnnnnnn #1#2#3#4#5#6#7
912 {
913   \exp_not:n { {#1} {#2} }
914   {
915     \__siunitx_number_digits:Nn \l__siunitx_number_min_integer_int {#3}
916     \exp_not:n {#3}
917   }
918   {
919     \exp_not:n {#4}
920     \__siunitx_number_digits:Nn \l__siunitx_number_min_decimal_int {#4}
921   }
922   \exp_not:n { {#5} {#6} {#7} }
923 }
924 \cs_new:Npn \__siunitx_number_digits:Nn #1#2
925 {
926   \int_compare:nNnT
927     { #1 - \tl_count:n {#2} } > 0
928     { \prg_replicate:nn { #1 - \tl_count:n {#2} } { 0 } }
929 }

```

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

```

\__siunitx_number_drop_exponent:NN
\__siunitx_number_drop_exponent:nnnnnnn

```

Simple stripping of the exponent.

```

930 \cs_new_protected:Npn \__siunitx_number_drop_exponent:NN #1#2
931 {
932   \bool_if:NT \l__siunitx_number_drop_exponent_bool

```

```

933     {
934         \tl_set:Nx #2
935         { \exp_after:wN \__siunitx_number_drop_exponent:nnnnnnn #1 }
936     }
937 }
938 \cs_new:Npn \__siunitx_number_drop_exponent:nnnnnnn #1#2#3#4#5#6#7
939 { \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.

```

940 \cs_new_protected:Npn \__siunitx_number_drop_uncertainty:NN #1#2
941 {
942     \bool_if:NTF \l__siunitx_number_drop_uncertainty_bool
943     {
944         \tl_set:Nx #2
945         { \exp_after:wN \__siunitx_number_drop_uncertainty:nnnnnnn #1 }
946     }
947     { \tl_set_eq:NN #2 #1 }
948 }
949 }
950 \cs_new:Npn \__siunitx_number_drop_uncertainty:nnnnnnn #1#2#3#4#5#6#7
951 { \exp_not:n { {#1} {#2} {#3} {#4} { } {#6} {#7} } }

```

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

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

```

952 \cs_new_protected:Npn \__siunitx_number_round:NN #1#2
953 {
954     \tl_set:Nx #2
955     {
956         \cs:w
957         __siunitx_number_round_ \l__siunitx_number_round_mode_tl :nnnnnnn
958         \exp_after:wN
959         \cs_end: #1
960     }
961 }
962 \cs_new:Npn \__siunitx_number_round_none:nnnnnnn #1#2#3#4#5#6#7
963 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }

```

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

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.

```

964 \cs_new:Npn \__siunitx_number_round:nnn #1#2#3
965 {
966     \__siunitx_number_round_auxi:nnnN {#1} {#2} { }
967     #3 \q_recursion_tail \q_recursion_stop
968 }
969 \cs_generate_variant:Nn \__siunitx_number_round:nnn { f }
970 \cs_new:Npn \__siunitx_number_round_auxi:nnnN #1#2#3#4

```

```

971 {
972   \quark_if_recursion_tail_stop_do:Nn #4
973   {
974     \__siunitx_number_round_auxii:nnnN {#1} {#3} { } #2
975     \q_recursion_tail \q_recursion_stop
976   }
977   \__siunitx_number_round_auxi:nnnN {#1} {#2} {#4#3}
978 }
979 \cs_new:Npn \__siunitx_number_round_auxii:nnnN #1#2#3#4
980 {
981   \quark_if_recursion_tail_stop_do:Nn #4
982   {
983     \tl_if_blank:nTF {#2}
984     {
985       \__siunitx_number_round_auxiv:nnnN {#1} { } { } #3
986       \q_recursion_tail \q_recursion_stop
987     }
988     {
989       \__siunitx_number_round_auxiii:nnnN {#1} {#3} { } #2
990       \q_recursion_tail \q_recursion_stop
991     }
992   }
993   \__siunitx_number_round_auxii:nnnN {#1} {#2} {#4#3}
994 }

```

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.

```

995 \cs_new:Npn \__siunitx_number_round_auxiii:nnnN #1#2#3#4
996 {
997   \quark_if_recursion_tail_stop_do:Nn #4
998   {
999     \__siunitx_number_round_auxiv:nnnN {#1} { } {#3} #2
1000     \q_recursion_tail \q_recursion_stop
1001   }
1002   \int_compare:nNnTF {#1} > 0
1003   {
1004     \exp_args:Nf \__siunitx_number_round_auxiii:nnnN
1005     { \int_eval:n { #1 - 1 } } {#2} { #4#3 }
1006   }
1007   { \__siunitx_number_round_auxv:nnN {#3} {#2} #4 }
1008 }
1009 \cs_new:Npn \__siunitx_number_round_auxiv:nnnN #1#2#3#4
1010 {
1011   \quark_if_recursion_tail_stop_do:Nn #4
1012   { { 0 } { } }
1013   \int_compare:nNnTF {#1} > 0
1014   {
1015     \exp_args:Nf \__siunitx_number_round_auxiv:nnnN
1016     { \int_eval:n { #1 - 1 } } { #2 0 } { #4#3 }
1017   }
1018   { \__siunitx_number_round_auxvi:nnnN {#3} {#2} #4 }

```

1019 }

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.

```

1020 \cs_new:Npn \__siunitx_number_round_auxv:nnN #1#2#3
1021 {
1022   \quark_if_recursion_tail_stop_do:Nn #3
1023   {
1024     \__siunitx_number_round_auxvi:nnN
1025     {#1} { } #2 \q_recursion_tail \q_recursion_stop
1026   }
1027   \bool_lazy_or:nnTF
1028   { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
1029   {
1030     \bool_lazy_all_p:n
1031     {
1032       { \l__siunitx_number_round_half_even_bool }
1033       { \int_if_odd_p:n {#3} }
1034       { \__siunitx_number_round_if_half_p:n {#1} }
1035     }
1036   }
1037   { \__siunitx_number_round_final_decimal:nnw }
1038   { \__siunitx_number_round_auxvii:nnN }
1039   {#2} { } #3
1040 }
1041 \cs_new:Npn \__siunitx_number_round_auxvi:nnnN #1#2#3
1042 {
1043   \quark_if_recursion_tail_stop_do:Nn #3
1044   { { 0 } { } }
1045   \bool_lazy_or:nnTF
1046   { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
1047   {
1048     \bool_lazy_all_p:n
1049     {
1050       { \l__siunitx_number_round_half_even_bool }
1051       { \int_if_odd_p:n {#3} }
1052       { \__siunitx_number_round_if_half_p:n {#1} }
1053     }
1054   }
1055   { \__siunitx_number_round_final_integer:nnw }
1056   { \__siunitx_number_round_auxviii:nnN }
1057   { } {#2} #3
1058 }

```

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.

```

1059 \cs_new:Npn \__siunitx_number_round_auxvii:nnN #1#2#3
1060 {
1061   \quark_if_recursion_tail_stop_do:Nn #3
1062   {
1063     \str_if_eq:nnTF {#1} { 0 }

```

```

1064         {
1065             \_siunitx_number_round_final_output:ff
1066             { 1 }
1067             { \_siunitx_number_round_truncate:n {#2} }
1068         }
1069         {
1070             \_siunitx_number_round_auxviii:nnN {#2} { } #1
1071             \q_recursion_tail \q_recursion_stop
1072         }
1073     }
1074     \int_compare:nNnTF {#3} = 9
1075     { \_siunitx_number_round_auxvii:nnN {#1} { 0 #2 } }
1076     {
1077         \int_compare:nNnTF {#3} = 0
1078         {
1079             \_siunitx_number_round_final_decimal:nnw
1080             {#1} { 1 \_siunitx_number_round_truncate:n {#2} }
1081         }
1082         {
1083             \_siunitx_number_round_final:fn
1084             { \int_eval:n { #3 + 1 } }
1085             { \_siunitx_number_round_final_decimal:nnw {#1} {#2} }
1086         }
1087     }
1088 }
1089 \cs_new:Npn \_siunitx_number_round_auxviii:nnN #1#2#3
1090 {
1091     \quark_if_recursion_tail_stop_do:Nn #3
1092     {
1093         \tl_if_blank:nTF {#1}
1094         {
1095             \_siunitx_number_round_final_shift:ff
1096             {
1097                 \exp_last_unbraced:Nf 1
1098                 { \_siunitx_number_round_truncate_direct:n {#2} } 0
1099             }
1100             { }
1101         }
1102         {
1103             \_siunitx_number_round_final_shift:ff
1104             { 1 #2 }
1105             { \_siunitx_number_round_truncate:n {#1} }
1106         }
1107     }
1108     \int_compare:nNnTF {#3} = 9
1109     { \_siunitx_number_round_auxviii:nnN {#1} { 0 #2 } }
1110     {
1111         \_siunitx_number_round_final:fn
1112         { \int_eval:n { #3 + 1 } }
1113         { \_siunitx_number_round_final_integer:nnw {#1} {#2} }
1114     }
1115 }

```

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

```

1116 \cs_new:Npn \__siunitx_number_round_final_decimal:nnw
1117   #1#2#3 \q_recursion_tail \q_recursion_stop
1118   {
1119     \__siunitx_number_round_final_output:ff
1120     { \tl_reverse:n {#1} }
1121     { \tl_reverse:n {#3} #2 }
1122   }
1123 \cs_new:Npn \__siunitx_number_round_final_integer:nnw
1124   #1#2#3 \q_recursion_tail \q_recursion_stop
1125   {
1126     \__siunitx_number_round_final_output:ff
1127     { \tl_reverse:n {#3} #2 }
1128     {#1}
1129   }
1130 \cs_new:Npn \__siunitx_number_round_final_output:nn #1#2 { {#1} {#2} }
1131 \cs_generate_variant:Nn \__siunitx_number_round_final_output:nn { ff }
1132 \cs_new:Npn \__siunitx_number_round_final:nn #1#2
1133   { #2 #1 }
1134 \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.

```

1135 \cs_new:Npn \__siunitx_number_round_final_shift:nn #1#2
1136   {
1137     \str_if_eq:VnTF \l__siunitx_number_round_mode_tl { places }
1138     {
1139       \use:c
1140       { __siunitx_number_round_ \l__siunitx_number_exponent_mode_tl :nn }
1141       {#1} {#2}
1142     }
1143     { {#1} {#2} }
1144   }
1145 \cs_generate_variant:Nn \__siunitx_number_round_final_shift:nn { ff }
1146 \cs_new:Npn \__siunitx_number_round_engineering:nn #1#2
1147   {
1148     \int_compare:nNnTF { \tl_count:n {#1} } = 4
1149     {
1150       \__siunitx_number_round_engineering:NNNNn #1 {#2}
1151       { }
1152       \__siunitx_number_round_final_shift:Nw 3
1153     }
1154     { {#1} {#2} }
1155   }
1156 \cs_new:Npn \__siunitx_number_round_engineering:NNNNn #1#2#3#4#5
1157   {
1158     {#1}
1159     \exp_args:NV \__siunitx_number_round_engineering:nnN
1160     { \l__siunitx_number_round_precision_int } { }
1161     #2#3#4#5 \q_recursion_tail \q_recursion_stop
1162   }
1163 \cs_new:Npn \__siunitx_number_round_engineering:nnN #1#2#3

```



```

1164 {
1165   \quark_if_recursion_tail_stop_do:Nn #3 { {#2} }
1166   \int_compare:nNnTF {#1} = { 0 }
1167     { \use_i_delimit_by_q_recursion_stop:nw { {#2} } }
1168     { \__siunitx_number_round_engineering:nnN { #1 - 1 } { #2#3 } }
1169 }
1170 \cs_new:Npn \__siunitx_number_round_fixed:nn #1#2 { {#1} {#2} }
1171 \cs_new:Npn \__siunitx_number_round_input:nn #1#2 { {#1} {#2} }
1172 \cs_new:Npn \__siunitx_number_round_scientific:nn #1#2
1173 {
1174   \__siunitx_number_exponent_shift:nnf
1175     { 1 } {#1} { \__siunitx_number_round_truncate_direct:n {#2} }
1176   { }
1177   \__siunitx_number_round_final_shift:Nw 1
1178 }
1179 \cs_new:Npn \__siunitx_number_round_final_shift:Nw #1#2 \__siunitx_number_round_places_end:nn
1180 { \__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.

```

1181 \cs_new:Npn \__siunitx_number_round_truncate:n #1
1182 {
1183   \str_if_eq:VnTF \l__siunitx_number_round_mode_tl { figures }
1184     { \__siunitx_number_round_truncate_direct:n {#1} }
1185     { #1 }
1186 }
1187 \cs_new:Npn \__siunitx_number_round_truncate_direct:n #1
1188 {
1189   \__siunitx_number_round_truncate:nnN { } { }
1190   #1 \q_recursion_tail \q_recursion_stop
1191 }
1192 \cs_new:Npn \__siunitx_number_round_truncate:nnN #1#2#3
1193 {
1194   \quark_if_recursion_tail_stop_do:Nn #3 { #1 }
1195   \__siunitx_number_round_truncate:nnN {#1#2} {#3}
1196 }

```

(End definition for __siunitx_number_round:nnn and others.)

__siunitx_number_round_if_half_p:n
 __siunitx_number_round_if_half:N

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

```

1197 \prg_new_conditional:Npnn \__siunitx_number_round_if_half:n #1 { p }
1198 {
1199   \int_compare:nNnTF { \tl_head:n { #1 0 } } = 5
1200     {
1201       \exp_after:wN \__siunitx_number_round_if_half:N \use_none:n #1 0
1202       \q_recursion_tail \q_recursion_stop
1203     }
1204     { \prg_return_false: }
1205 }
1206 \cs_new:Npn \__siunitx_number_round_if_half:N #1
1207 {
1208   \quark_if_recursion_tail_stop_do:Nn #1

```

```

1209     { \prg_return_true: }
1210 \int_compare:nNnTF {#1} = 0
1211 { \__siunitx_number_round_if_half:N }
1212 { \use_i_delimit_by_q_recursion_stop:nw { \prg_return_false: } }
1213 }

```

(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.

```

1214 \cs_new:Npn \__siunitx_number_round_pad:nnn #1#2#3
1215 {
1216   {#2}
1217   {
1218     #3
1219     \bool_if:NT \__siunitx_number_round_pad_bool
1220     { \prg_replicate:nn {#1} { 0 } }
1221   }
1222 }

```

(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 uncertainty, and that the number of figures requested is positive: if not, the result is always fixed at zero.

__siunitx_number_round_figures_count:nnN
__siunitx_number_round_figures_count:nnnN

```

1223 \cs_new:Npn \__siunitx_number_round_figures:nnnnnnn #1#2#3#4#5#6#7
1224 {
1225   \tl_if_blank:nTF {#5}
1226   {
1227     \int_compare:nNnTF \__siunitx_number_round_precision_int > 0
1228     {
1229       \exp_not:n { {#1} {#2} }
1230       \__siunitx_number_round_figures_count:nnN {#3} {#4} #3#4
1231       \q_recursion_tail \q_recursion_stop
1232       \exp_not:n { { } {#6} {#7} }
1233     }
1234     { { } { } { } { 0 } { } { } { } { } { 0 } }
1235   }
1236   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1237 }

```

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

```

1238 \cs_new:Npn \__siunitx_number_round_figures_count:nnN #1#2#3
1239 {
1240   \quark_if_recursion_tail_stop_do:Nn #3
1241   { { } { } { 0 } { } { } { } { 0 } }
1242   \int_compare:nNnTF {#3} = 0
1243   { \__siunitx_number_round_figures_count:nnN {#1} {#2} }
1244   { \__siunitx_number_round_figures_count:nnnN { 1 } {#1} {#2} }
1245 }
1246 \cs_new:Npn \__siunitx_number_round_figures_count:nnnN #1#2#3#4
1247 {
1248   \quark_if_recursion_tail_stop_do:Nn #4

```

```

1249 {
1250   \int_compare:nNnTF {#1} > \l__siunitx_number_round_precision_int
1251   {
1252     \__siunitx_number_round:fnn
1253     { \int_eval:n { #1 - \l__siunitx_number_round_precision_int } }
1254     {#2} {#3}
1255   }
1256   {
1257     \__siunitx_number_round_pad:nnn
1258     { \l__siunitx_number_round_precision_int - (#1) } {#2} {#3}
1259   }
1260 }
1261 \exp_args:Nf \__siunitx_number_round_figures_count:nnnN
1262 { \int_eval:n { #1 + 1 } } {#2} {#3}
1263 }

```

(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
\__siunitx_number_round_places_finalise:n
\__siunitx_number_round_places_finalise:nnnnnnn
\__siunitx_number_round_places_finalise:nnnnn

```

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.

```

1264 \cs_new:Npn \__siunitx_number_round_places:nnnnnnn #1#2#3#4#5#6#7
1265 {
1266   \tl_if_blank:nTF {#5}
1267   {
1268     \exp_args:Ne \__siunitx_number_round_places_finalise:n
1269     {
1270       \exp_not:n { {#1} {#2} }
1271       \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
1272       { \__siunitx_number_round_places_decimal:nn }
1273       { \__siunitx_number_round_places_integer:nn }
1274       {#3} {#4}
1275       \__siunitx_number_round_places_end:nn {#6} {#7}
1276     }
1277   }
1278   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1279 }
1280 \cs_new:Npn \__siunitx_number_round_places_end:nn #1#2 { { } \exp_not:n { {#1} {#2} } }
1281 \cs_new:Npn \__siunitx_number_round_places_decimal:nn #1#2
1282 {
1283   \int_compare:nNnTF
1284   { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} } > 0
1285   {
1286     \__siunitx_number_round_pad:nnn
1287     { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} }
1288     {#1} {#2}
1289   }
1290   {
1291     \__siunitx_number_round:fnn
1292     {
1293       \int_eval:n
1294       { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }

```

```

1295     }
1296     {#1} {#2}
1297 }
1298 }
1299 \cs_new:Npn \__siunitx_number_round_places_integer:nn #1#2
1300 {
1301   \__siunitx_number_round:fnn
1302   {
1303     \int_eval:n
1304     { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }
1305   }
1306   {#1} {#2}
1307 }

```

To finalise rounding to places, we have to worry about a minimum value: that is basically a case of looking for value of zero and rearranging. We also need to worry about a “negative zero” arising.

```

1308 \cs_new:Npn \__siunitx_number_round_places_finalise:n #1
1309 { \__siunitx_number_round_places_finalise:nnnnnnn #1 }
1310 \cs_new:Npn \__siunitx_number_round_places_finalise:nnnnnnn #1#2#3#4#5#6#7
1311 {
1312   \bool_lazy_and:nnTF
1313   { \str_if_eq_p:nn {#3} { 0 } }
1314   {
1315     \str_if_eq_p:ee
1316     { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1317   }
1318   {
1319     \tl_if_empty:NTF \l__siunitx_number_round_min_tl
1320     {
1321       \exp_not:n { {#1} }
1322       { \str_if_eq:nnF {#2} { - } { \exp_not:n {#2} } }
1323       \exp_not:n { {#3} {#4} {#5} {#6} {#7} }
1324     }
1325     {
1326       \exp_after:wN \__siunitx_number_round_places_finalise:nnnnn
1327       \l__siunitx_number_round_min_tl {#2} {#6} {#7}
1328     }
1329   }
1330   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1331 }
1332 \cs_new:Npn \__siunitx_number_round_places_finalise:nnnnn #1#2#3#4#5
1333 {
1334   {
1335     \str_if_eq:nnTF {#3} { - }
1336     { > }
1337     { < }
1338   }
1339   \exp_not:n { {#3} {#1} {#2} { } {#4} {#5} }
1340 }

```

(End definition for __siunitx_number_round_places:nnnnnnn and others.)

__siunitx_number_round_uncertainty:nnnnnnn 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
 __siunitx_number_round_uncertainty:nnn
 __siunitx_number_round_uncertainty:nnnn

used for rounding.

```

1341 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnnnn #1#2#3#4#5#6#7
1342 {
1343   \bool_lazy_or:nnTF
1344     { \tl_if_blank_p:n {#5} }
1345     { ! \int_compare_p:nNn \l__siunitx_number_round_precision_int > 0 }
1346     { \exp_not:n { {#1} #2 {#3} {#4} { } #6 {#7} } }
1347     {
1348       \str_if_eq:eeTF { \tl_head:n {#5} } { S }
1349       {
1350         \exp_not:n { {#1} {#2} }
1351         \exp_args:Nnno \__siunitx_number_round_uncertainty:nnn
1352           {#3} {#4} { \use_ii:nn #5 }
1353         \exp_not:n { {#6} {#7} }
1354       }
1355       { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1356     }
1357 }

```

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.

```

1358 \cs_new:Npn \__siunitx_number_round_uncertainty:nnn #1#2#3
1359 {
1360   \exp_last_unbraced:Nf \__siunitx_number_round_uncertainty:nnnnn
1361   {
1362     \__siunitx_number_round:fnn
1363     { \tl_count:n {#3} - \l__siunitx_number_round_precision_int } { } {#3}
1364   }
1365   {#1} {#2} {#3}
1366 }
1367 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnn #1#2#3#4#5
1368 {
1369   \tl_if_blank:nTF {#1}
1370   {
1371     \__siunitx_number_round:fnn
1372     { \tl_count:n {#5} - \tl_count:n {#2} } {#3} {#4}
1373     { { S } {#2} }
1374   }
1375   {
1376     \__siunitx_number_round:fnn
1377     { \tl_count:n {#5} - \tl_count:n {#2} + 1 } {#3} {#4}
1378     { { S } { #1 \__siunitx_number_round_truncate_direct:n {#2} } }
1379   }
1380 }

```

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

Simple stripping of the decimal part if zero.

```

\__siunitx_number_zero_decimal:NN
\__siunitx_number_zero_decimal:nnnnnnn
1381 \cs_new_protected:Npn \__siunitx_number_zero_decimal:NN #1#2
1382 {
1383   \bool_if:NT \l__siunitx_number_drop_zero_decimal_bool
1384   {

```

```

1385         \tl_set:Nx #2
1386         { \exp_after:wN \__siunitx_number_zero_decimal:nnnnnnn #1 }
1387     }
1388 }
1389 \cs_new:Npn \__siunitx_number_zero_decimal:nnnnnnn #1#2#3#4#5#6#7
1390 {
1391     \exp_not:n { {#1} {#2} {#3} }
1392     \str_if_eq:eeTF
1393     { \exp_not:n {#4} }
1394     { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1395     { { } }
1396     { \exp_not:n { {#4} } }
1397     \exp_not:n { {#5} {#6} {#7} }
1398 }

```

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

2.5 Number modification

A simply case of breaking down and rebuilding the number.

```

\siunitx_number_adjust_exponent:nn \siunitx_number_adjust_exponent:Nn
\__siunitx_number_adjust_exp:nnnnnnnn \__siunitx_number_adjust_exp:nn
\__siunitx_number_adjust_exp:nn \__siunitx_number_adjust_exp:nNw
1399 \cs_new:Npn \siunitx_number_adjust_exponent:nn #1#2
1400 { \__siunitx_number_adjust_exp:nnnnnnnn #1 {#2} }
1401 \cs_new:Npn \siunitx_number_adjust_exponent:Nn #1#2
1402 {
1403     \tl_if_empty:NF #1
1404     { \exp_args:NV \siunitx_number_adjust_exponent:nn #1 {#2} }
1405 }
1406 \cs_new:Npn \__siunitx_number_adjust_exp:nnnnnnnn #1#2#3#4#5#6#7#8
1407 {
1408     \exp_not:n { {#1} {#2} {#3} {#4} {#5} }
1409     \exp_args:Ne \__siunitx_number_adjust_exp:nn { \fp_eval:n { #6#7 + #8 } } {#6}
1410 }
1411 \cs_new:Npn \__siunitx_number_adjust_exp:nn #1#2
1412 { \__siunitx_number_adjust_exp:nNw {#2} #1 \q_stop }
1413 \cs_new:Npn \__siunitx_number_adjust_exp:nNw #1#2#3 \q_stop
1414 {
1415     \token_if_eq_meaning:NNTF #2 -
1416     { { - } { \exp_not:n {#3} } }
1417     { { \str_if_eq:nnT {#1} { + } { + } } { \exp_not:n {#2#3} } }
1418 }

```

(End definition for \siunitx_number_adjust_exponent:nn and others. These functions are documented on page 37.)

2.6 Outputting parsed numbers

Purely internal for the present.

```

\l__siunitx_number_bracket_close_tl \l__siunitx_number_bracket_open_tl
1419 \tl_new:N \l__siunitx_number_bracket_close_tl
1420 \tl_new:N \l__siunitx_number_bracket_open_tl
1421 \tl_set:Nn \l__siunitx_number_bracket_open_tl { ( }
1422 \tl_set:Nn \l__siunitx_number_bracket_close_tl { ) }

```

(End definition for \l__siunitx_number_bracket_close_tl and \l__siunitx_number_bracket_open_tl.)

`\l_siunitx_number_bracket_ambiguous_bool`

1423 `\bool_new:N \l_siunitx_number_bracket_ambiguous_bool`

(End definition for `\l_siunitx_number_bracket_ambiguous_bool`. This variable is documented on page ??.)

`\l_siunitx_number_output_decimal_tl`

1424 `\tl_new:N \l_siunitx_number_output_decimal_tl`

(End definition for `\l_siunitx_number_output_decimal_tl`. This variable is documented on page 38.)

`\l_siunitx_number_bracket_negative_bool`

Keys producing tokens in the output.

1425 `\keys_define:nn { siunitx }`

1426 `{`

1427 `bracket-ambiguous-numbers .bool_set:N =`

1428 `\l_siunitx_number_bracket_ambiguous_bool ,`

1429 `bracket-negative-numbers .bool_set:N =`

1430 `\l_siunitx_number_bracket_negative_bool ,`

1431 `exponent-base .tl_set:N =`

1432 `\l_siunitx_number_exponent_base_tl ,`

1433 `exponent-product .tl_set:N =`

1434 `\l_siunitx_number_exponent_product_tl ,`

1435 `group-digits .choice: ,`

1436 `group-digits / all .code:n =`

1437 `{`

1438 `\bool_set_true:N \l_siunitx_number_group_decimal_bool`

1439 `\bool_set_true:N \l_siunitx_number_group_integer_bool`

1440 `}` ,

1441 `group-digits / decimal .code:n =`

1442 `{`

1443 `\bool_set_true:N \l_siunitx_number_group_decimal_bool`

1444 `\bool_set_false:N \l_siunitx_number_group_integer_bool`

1445 `}` ,

1446 `group-digits / integer .code:n =`

1447 `{`

1448 `\bool_set_false:N \l_siunitx_number_group_decimal_bool`

1449 `\bool_set_true:N \l_siunitx_number_group_integer_bool`

1450 `}` ,

1451 `group-digits / none .code:n =`

1452 `{`

1453 `\bool_set_false:N \l_siunitx_number_group_decimal_bool`

1454 `\bool_set_false:N \l_siunitx_number_group_integer_bool`

1455 `}` ,

1456 `group-digits .default:n = all ,`

1457 `group-minimum-digits .int_set:N =`

1458 `\l_siunitx_number_group_minimum_int ,`

1459 `group-separator .tl_set:N =`

1460 `\l_siunitx_number_group_separator_tl ,`

1461 `negative-color .tl_set:N =`

1462 `\l_siunitx_number_negative_color_tl ,`

1463 `output-close-uncertainty .tl_set:N =`

1464 `\l_siunitx_number_output_uncert_close_tl ,`

1465 `output-decimal-marker .tl_set:N =`

1466 `\l_siunitx_number_output_decimal_tl ,`

```

1467     output-open-uncertainty .tl_set:N =
1468     \l__siunitx_number_output_uncert_open_tl ,
1469     print-implicit-plus .bool_set:N =
1470     \l__siunitx_number_implicit_plus_bool ,
1471     print-unity-mantissa .bool_set:N =
1472     \l__siunitx_number_unity_mantissa_bool ,
1473     print-zero-exponent .bool_set:N =
1474     \l__siunitx_number_zero_exponent_bool ,
1475     separate-uncertainty .bool_set:N =
1476     \l__siunitx_number_uncert_separate_bool ,
1477     uncertainty-separator .tl_set:N =
1478     \l__siunitx_number_uncert_separator_tl ,
1479     tight-spacing .bool_set:N =
1480     \l__siunitx_number_tight_bool
1481   }
1482   \bool_new:N \l__siunitx_number_group_decimal_bool
1483   \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_output:N
\siunitx_number_output:n
\siunitx_number_output:NN
\siunitx_number_output:nN
1484 \cs_new:Npn \siunitx_number_output:N #1
1485   { \__siunitx_number_output:Nn #1 { } }
1486 \cs_new:Npn \siunitx_number_output:n #1
1487   { \__siunitx_number_output:nn #1 { } }
1488 \cs_new:Npn \siunitx_number_output:NN #1#2
1489   { \__siunitx_number_output:Nn #1 {#2} }
1490 \cs_new:Npn \siunitx_number_output:nN #1#2
1491   { \__siunitx_number_output:nn #1 {#2} }
1492 \cs_new:Npn \__siunitx_number_output:Nn #1#2
1493   {
1494     \tl_if_empty:NF #1
1495     { \exp_after:wN \__siunitx_number_output:nnnnnnn #1 {#2} }
1496   }
1497 \cs_new:Npn \__siunitx_number_output:nn #1#2
1498   {
1499     \tl_if_empty:NF {#1}
1500     { \__siunitx_number_output:nnnnnnn #1 {#2} }
1501   }
1502 \cs_new:Npn \__siunitx_number_output:nnnnnnn #1#2#3#4#5#6#7#8
1503   {
1504     \__siunitx_number_output_comparator:nn {#1} {#8}
1505     \__siunitx_number_output_bracket:nn {#5} {#7}
1506     \__siunitx_number_output_sign:nnn {#1} {#2} {#8}
1507     \__siunitx_number_output_integer:nnn {#3} {#4} {#7}
1508     \__siunitx_number_output_decimal:nn {#4} {#8}
1509     \__siunitx_number_output_uncertainty:nnn {#5} {#4} {#8}
1510     \__siunitx_number_output_exponent:nnnn {#6} {#7} { #3 . #4 } {#8}
1511     \__siunitx_number_output_end:
1512   }
\__siunitx_number_output_uncertainty_unaligned:n
\__siunitx_number_output_uncertainty_S:nnnw
\__siunitx_number_output_uncertainty_S_aux:nnn
\__siunitx_number_output_uncertainty_S:nnnw
\__siunitx_number_output_uncertainty_S:fnw
\__siunitx_number_output_uncertainty_S:nnw
\__siunitx_number_output_exponent:nnnn
\__siunitx_number_output_end:

```

Adding brackets for the combination of a separate uncertainty with an exponent may need brackets. This needs testing up-front, so has to come before the main formatting

routines.

```

1513 \cs_new:Npn \__siunitx_number_output_bracket:nn #1#2
1514 {
1515   \bool_lazy_all:nT
1516   {
1517     { \l__siunitx_number_uncert_separate_bool }
1518     { \l__siunitx_number_bracket_ambiguous_bool }
1519     { ! \tl_if_blank_p:n {#1} } }
1520   {
1521     \bool_lazy_or_p:nn
1522     { \l__siunitx_number_zero_exponent_bool }
1523     { ! \str_if_eq_p:nn {#2} { 0 } } }
1524   }
1525 }
1526 \__siunitx_number_output_bracket:w
1527 }
1528 \cs_new:Npn \__siunitx_number_output_bracket:w #1 \__siunitx_number_output_exponent:nnnn
1529 {
1530   \exp_not:V \l__siunitx_number_bracket_open_tl
1531   #1
1532   \exp_not:V \l__siunitx_number_bracket_close_tl
1533   \__siunitx_number_output_exponent:nnnn
1534 }

```

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

```

1535 \cs_new:Npn \__siunitx_number_output_comparator:nn #1#2
1536 {
1537   \tl_if_blank:nF {#1}
1538   { \exp_not:n { \mathord {#1} } } }
1539   \exp_not:n {#2}
1540 }

```

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. We also have the comparator passed here: if it is present, we need to deal with tighter spacing.

```

1541 \cs_new:Npn \__siunitx_number_output_sign:nnn #1#2#3
1542 {
1543   \tl_if_blank:nTF {#2}
1544   {
1545     \bool_if:NT \l__siunitx_number_implicit_plus_bool
1546     { \__siunitx_number_output_sign:nN {#1} + }
1547   }
1548   {
1549     \str_if_eq:nnTF {#2} { - }
1550     {
1551       \tl_if_empty:NF \l__siunitx_number_negative_color_tl
1552       { \__siunitx_number_output_sign_color:w }
1553       \bool_if:NTF \l__siunitx_number_bracket_negative_bool
1554       { \__siunitx_number_output_sign_brackets:w }
1555       { \__siunitx_number_output_sign:nN {#1} #2 }
1556     }
1557     { \__siunitx_number_output_sign:nN {#1} #2 }

```

```

1558     }
1559     \exp_not:n {#3}
1560   }
1561   \cs_new:Npn \__siunitx_number_output_sign:nN #1#2
1562   {
1563     \tl_if_blank:nTF {#1}
1564     { \__siunitx_number_output_sign:N #2 }
1565     { \exp_not:n { \mathord {#2} } }
1566   }
1567   \cs_new:Npn \__siunitx_number_output_sign:N #1
1568   {
1569     \bool_if:NTF \l__siunitx_number_tight_bool
1570     { \exp_not:n { \mathord {#1} } }
1571     { \exp_not:n {#1} }
1572   }
1573   \cs_new:Npn
1574     \__siunitx_number_output_sign_color:w #1 \__siunitx_number_output_end:
1575   {
1576     \exp_not:N \textcolor { \exp_not:V \l__siunitx_number_negative_color_tl }
1577     {
1578       #1
1579       \__siunitx_number_output_end:
1580     }
1581   }
1582   \cs_new:Npn
1583     \__siunitx_number_output_sign_brackets:w #1 \__siunitx_number_output_end:
1584   {
1585     \exp_not:V \l__siunitx_number_bracket_open_tl
1586     #1
1587     \exp_not:V \l__siunitx_number_bracket_close_tl
1588     \__siunitx_number_output_end:
1589   }

```

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.

```

1590   \cs_new:Npn \__siunitx_number_output_integer:nnn #1#2#3
1591   {
1592     \bool_lazy_all:nF
1593     {
1594       { \str_if_eq_p:nn {#1} { 1 } }
1595       { \tl_if_blank_p:n {#2} }
1596       { ! \str_if_eq_p:nn {#3} { 0 } }
1597       { ! \l__siunitx_number_unity_mantissa_bool }
1598     }
1599     { \__siunitx_number_output_digits:nn { integer } {#1} }
1600   }
1601   \cs_new:Npn \__siunitx_number_output_decimal:nn #1#2
1602   {
1603     \exp_not:n {#2}
1604     \tl_if_blank:nF {#1}
1605     {
1606       \str_if_eq:VnTF \l__siunitx_number_output_decimal_tl { , }
1607       { \exp_not:N \mathord {

```

```

1608         { \use:n }
1609         { \exp_not:V \l_siunitx_number_output_decimal_tl }
1610     }
1611     \exp_not:n {#2}
1612     \__siunitx_number_output_digits:nn { decimal } {#1}
1613 }
1614 \cs_generate_variant:Nn \__siunitx_number_output_decimal:nn { f }
1615 \cs_new:Npn \__siunitx_number_output_digits:nn #1#2
1616 {
1617     \bool_if:cTF { l__siunitx_number_group_ #1 _ bool }
1618     {
1619         \int_compare:nNnTF
1620         { \tl_count:n {#2} } < \l__siunitx_number_group_minimum_int
1621         { \exp_not:n {#2} }
1622         { \use:c { __siunitx_number_output_ #1 _aux:n } {#2} }
1623     }
1624     { \exp_not:n {#2} }
1625 }

```

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.

```

1626 \cs_new:Npn \__siunitx_number_output_integer_aux:n #1
1627 {
1628     \use:c
1629     {
1630         __siunitx_number_output_integer_aux_
1631         \int_eval:n { \int_mod:nn { \tl_count:n {#1} } { 3 } }
1632         :n
1633     } {#1}
1634 }
1635 \cs_new:cpn { __siunitx_number_output_integer_aux_0:n } #1
1636 { __siunitx_number_output_integer_first:nnNN #1 \q_nil }
1637 \cs_new:cpn { __siunitx_number_output_integer_aux_1:n } #1
1638 { __siunitx_number_output_integer_first:nnNN { } { } #1 \q_nil }
1639 \cs_new:cpn { __siunitx_number_output_integer_aux_2:n } #1
1640 { __siunitx_number_output_integer_first:nnNN { } #1 \q_nil }
1641 \cs_new:Npn \__siunitx_number_output_integer_first:nnNN #1#2#3#4
1642 {
1643     \exp_not:n {#1#2#3}
1644     \quark_if_nil:NF #4
1645     { \__siunitx_number_output_integer_loop:NNNN #4 }
1646 }
1647 \cs_new:Npn \__siunitx_number_output_integer_loop:NNNN #1#2#3#4
1648 {
1649     \str_if_eq:VnTF \l__siunitx_number_group_separator_tl { , }
1650     { \exp_not:N \mathord }
1651     { \use:n }
1652     { \exp_not:V \l__siunitx_number_group_separator_tl }
1653     \exp_not:n {#1#2#3}
1654     \quark_if_nil:NF #4
1655     { \__siunitx_number_output_integer_loop:NNNN #4 }
1656 }

```

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.

```

1657 \cs_new:Npn \__siunitx_number_output_decimal_aux:n #1
1658 {
1659   \__siunitx_number_output_decimal_loop:NNNN \c_empty_tl
1660   #1 \q_nil \q_nil \q_nil
1661 }
1662 \cs_new:Npn \__siunitx_number_output_decimal_loop:NNNN #1#2#3#4
1663 {
1664   \quark_if_nil:NF #2
1665   {
1666     \exp_not:V #1
1667     \exp_not:n {#2}
1668     \quark_if_nil:NTF #3
1669     { \use_none:n }
1670     {
1671       \exp_not:n {#3}
1672       \quark_if_nil:NTF #4
1673       { \use_none:nn }
1674       {
1675         \exp_not:n {#4}
1676         \__siunitx_number_output_decimal_loop:NNNN
1677         \l__siunitx_number_group_separator_tl
1678       }
1679     }
1680   }
1681 }

```

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.

```

1682 \cs_new:Npn \__siunitx_number_output_uncertainty:nnn #1#2#3
1683 {
1684   \tl_if_blank:NTF {#1}
1685   { \__siunitx_number_output_uncertainty_unaligned:n {#3} }
1686   {
1687     \use:c { \__siunitx_number_output_uncertainty_ \tl_head:n {#1} :nnnw }
1688     {#2} {#3} #1
1689   }
1690 }
1691 \cs_new:Npn \__siunitx_number_output_uncertainty_unaligned:n #1
1692 { \exp_not:n { #1 #1 #1 #1 } }
1693 \cs_new:Npn \__siunitx_number_output_uncertainty_S:nnnw #1#2#3#4
1694 {
1695   \bool_if:NTF \l__siunitx_number_uncert_separate_bool
1696   {
1697     \exp_not:n {#2}
1698     \__siunitx_number_output_sign:N \pm
1699     \exp_not:n {#2}
1700     \exp_args:Nf \__siunitx_number_output_uncertainty_S_aux:nnn
1701     { \int_eval:n { \tl_count:n {#4} - \tl_count:n {#1} } }

```

```

1702         {#4} {#2}
1703     }
1704     {
1705         \exp_not:V \l__siunitx_number_uncert_separator_tl
1706         \exp_not:V \l__siunitx_number_output_uncert_open_tl
1707         \exp_not:n {#4}
1708         \exp_not:V \l__siunitx_number_output_uncert_close_tl
1709         \__siunitx_number_output_uncertainty_unaligned:n {#2}
1710     }
1711 }
1712 \cs_new:Npn \__siunitx_number_output_uncertainty_S_aux:nnn #1#2#3
1713 {
1714     \int_compare:nNnTF {#1} > 0
1715     {
1716         \__siunitx_number_output_uncertainty_S_aux:fnnw
1717         { \int_eval:n { #1 - 1 } }
1718         {#3}
1719         { }
1720         #2 \q_nil
1721     }
1722     {
1723         0
1724         \__siunitx_number_output_decimal:fn
1725         {
1726             \prg_replicate:nn { \int_abs:n {#1} } { 0 }
1727             #2
1728         }
1729         {#3}
1730     }
1731 }
1732 \cs_new:Npn \__siunitx_number_output_uncertainty_S_aux:nnnw #1#2#3#4
1733 {
1734     \quark_if_nil:NF #4
1735     {
1736         \int_compare:nNnTF {#1} = 0
1737         { \__siunitx_number_output_uncertainty_S_aux:nnw {#3#4} {#2} }
1738         {
1739             \__siunitx_number_output_uncertainty_S_aux:fnnw
1740             { \int_eval:n { #1 - 1 } }
1741             {#2}
1742             {#3#4}
1743         }
1744     }
1745 }
1746 \cs_generate_variant:Nn \__siunitx_number_output_uncertainty_S_aux:nnnw { f }
1747 \cs_new:Npn \__siunitx_number_output_uncertainty_S_aux:nnw #1#2#3 \q_nil
1748 {
1749     \__siunitx_number_output_digits:nn { integer } {#1}
1750     \__siunitx_number_output_decimal:nn {#3} {#2}
1751 }

```

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.

```

1752 \cs_new:Npn \__siunitx_number_output_exponent:nnnn #1#2#3#4
1753 {
1754   \exp_not:n {#4}
1755   \bool_lazy_or:nnTF
1756     { \l__siunitx_number_zero_exponent_bool }
1757     { ! \str_if_eq_p:nn {#2} { 0 } }
1758   {
1759     \bool_lazy_and:nnTF
1760       { \str_if_eq_p:nn {#3} { 1. } }
1761       { ! \l__siunitx_number_unity_mantissa_bool }
1762     { \exp_not:n {#4} }
1763     {
1764       \bool_if:NTF \l__siunitx_number_tight_bool
1765         { \exp_not:N \mathord }
1766         { \use:n }
1767         { \exp_not:V \l__siunitx_number_exponent_product_tl }
1768       \exp_not:n {#4}
1769     }
1770     \exp_not:V \l__siunitx_number_exponent_base_tl
1771     ^
1772     {
1773       \tl_if_blank:nTF {#1}
1774       {
1775         \bool_if:NT \l__siunitx_number_implicit_plus_bool
1776         { \__siunitx_number_output_sign:N + }
1777       }
1778       { \__siunitx_number_output_sign:N #1 }
1779       \__siunitx_number_output_digits:nn { integer } {#2}
1780     }
1781   }
1782   { \exp_not:n {#4} }
1783 }

```

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

```

1784 \cs_new:Npn \__siunitx_number_output_end: { }

```

(End definition for \siunitx_number_output:N and others. These functions are documented on page 37.)

2.7 Miscellaneous tools

\l__siunitx_number_valid_tl The list of valid tokens.

```

1785 \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.

```

1786 \prg_new_protected_conditional:Npnn \siunitx_if_number:n #1
1787 { T , F , TF }

```

```

1788 {
1789   \group_begin:
1790     \bool_set_true:N \l__siunitx_number_validate_bool
1791     \bool_set_true:N \l_siunitx_number_parse_bool
1792     \siunitx_number_parse:nN {#1} \l__siunitx_number_parsed_tl
1793     \tl_if_empty:NTF \l__siunitx_number_parsed_tl
1794       {
1795         \group_end:
1796         \prg_return_false:
1797       }
1798       {
1799         \group_end:
1800         \prg_return_true:
1801       }
1802 }

```

(End definition for \siunitx_if_number:nTF. This function is documented on page 37.)

\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_token_auxi:NN 1803 \prg_new_conditional:Npnn \siunitx_if_number_token:N #1
  \_siunitx_number_if_token_auxii:NN 1804 { p , T , F , TF }
  \_siunitx_number_if_token_auxiii:NN 1805 {
1806   \__siunitx_number_token_auxi:NN #1
1807   \l_siunitx_number_input_decimal_tl
1808   \l_siunitx_number_input_uncert_close_tl
1809   \l_siunitx_number_input_comparator_tl
1810   \l_siunitx_number_input_digit_tl
1811   \l_siunitx_number_input_exponent_tl
1812   \l_siunitx_number_input_ignore_tl
1813   \l_siunitx_number_input_uncert_open_tl
1814   \l_siunitx_number_input_sign_tl
1815   \l_siunitx_number_input_uncert_sign_tl
1816   \q_recursion_tail
1817   \q_recursion_stop
1818 }
1819 \cs_new:Npn \__siunitx_number_token_auxi:NN #1#2
1820 {
1821   \quark_if_recursion_tail_stop_do:Nn #2 { \prg_return_false: }
1822   \__siunitx_number_token_auxii:NN #1 #2
1823   \__siunitx_number_token_auxi:NN #1
1824 }
1825 \cs_new:Npn \_siunitx_number_token_auxii:NN #1#2
1826 {
1827   \exp_after:wN \__siunitx_number_token_auxiii:NN \exp_after:wN #1
1828   #2 \q_recursion_tail \q_recursion_stop
1829 }
1830 \cs_new:Npn \_siunitx_number_token_auxiii:NN #1#2
1831 {
1832   \quark_if_recursion_tail_stop:N #2
1833   \str_if_eq:nnT {#1} {#2}
1834   {
1835     \use_i_delimit_by_q_recursion_stop:nw
1836     {

```

```

1837         \use_i_delimit_by_q_recursion_stop:nw
1838         { \prg_return_true: }
1839     }
1840 }
1841 \__siunitx_number_token_auxiii:NN #1
1842 }

```

(End definition for `\siunitx_if_number_token:NTF` and others. This function is documented on page 37.)

2.8 Messages

```

1843 \msg_new:nnnn { siunitx } { invalid-number }
1844 { Invalid-number~'#1'. }
1845 {
1846   The~input~'#1'~could~not~be~parsed~as~a~number~following~the~
1847   format~defined~in~module~documentation.
1848 }

```

2.9 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.

```

1849 \keys_set:nn { siunitx }
1850 {
1851   bracket-ambiguous-numbers = true ,
1852   bracket-negative-numbers  = false ,
1853   drop-exponent             = false ,
1854   drop-uncertainty          = false ,
1855   drop-zero-decimal         = false ,
1856   evaluate-expression        = false ,
1857   exponent-base              = 10 ,
1858   exponent-mode              = input ,
1859   exponent-product           = \times ,
1860   expression                 = #1 ,
1861   fixed-exponent             = 0 ,
1862   group-digits               = all ,
1863   group-minimum-digits       = 4 ,
1864   group-separator            = \, , % (
1865   input-close-uncertainty    = ) ,
1866   input-comparators          = { <=>\approx\ge\geq\gg\le\leq\ll\sim } ,
1867   input-decimal-markers      = { ., } ,
1868   input-digits               = 0123456789 ,
1869   input-exponent-markers     = dDeE ,
1870   input-ignore               = \, ,
1871   input-open-uncertainty     = ( , % )
1872   input-signs                 = +- \mp \pm ,
1873   input-uncertainty-signs    = \pm ,
1874   minimum-decimal-digits     = 0 ,
1875   minimum-integer-digits     = 0 ,
1876   negative-color              = , % (
1877   output-close-uncertainty    = ) ,
1878   output-decimal-marker      = . ,
1879   output-open-uncertainty     = ( , % )
1880   parse-numbers              = true ,

```



```

1881     print-implicit-plus      = false      ,
1882     print-unity-mantissa     = true       ,
1883     print-zero-exponent      = false      ,
1884     retain-explicit-plus     = false      ,
1885     retain-zero-uncertainty  = false      ,
1886     round-half               = up         ,
1887     round-minimum            = 0          ,
1888     round-mode                = none      ,
1889     round-pad                = true       ,
1890     round-precision          = 2         ,
1891     separate-uncertainty     = false      ,
1892     uncertainty-separator    =           ,
1893     tight-spacing            = false      ,
1894 }
1895 </package>

```

Part VI

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

<code>\siunitx_print:nn</code>
<code>\siunitx_print:(nV nx)</code>

`\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.

<code>\siunitx_print_match:n</code>
<code>\siunitx_print_math:n</code>
<code>\siunitx_print_text:n</code>

`\siunitx_print_match:n {<material>}`
`\siunitx_print_math:n {<material>}`
`\siunitx_print_text:n {<material>}`

Prints the *<material>* as described for `\siunitx_print:nn` but with a fixed text or math mode output. The printing does *not* set color (which is managed on a **unit/number** basis), but otherwise sets the font as described above. The **match** function uses either the prevailing math or text mode.

1.1 Key-value options

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

<code>color</code>

`color = <color>`

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

<u>mode</u>	<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> .
<u>number-color</u>	<code>number-color = <color></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).
<u>number-mode</u>	<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> .
<u>propagate-math-font</u>	<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> .
<u>reset-math-version</u>	<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> .
<u>reset-text-family</u>	<code>reset-text-family = true false</code> Switch to determine whether the active text family is reset to <code>\rmfamily</code> when printing in text mode. The standard setting is <code>true</code> .
<u>reset-text-series</u>	<code>reset-text-series = true false</code> Switch to determine whether the active text series is reset to <code>\mdseries</code> when printing in text mode. The standard setting is <code>true</code> .
<u>reset-text-shape</u>	<code>reset-text-shape = true false</code> Switch to determine whether the active text shape is reset to <code>\upshape</code> when printing in text mode. The standard setting is <code>true</code> .
<u>text-family-to-math</u>	<code>text-family-to-math = true false</code> 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 <code>false</code> .

<hr/> <hr/> text-weight-to-math	<p><code>text-weight-to-math = true false</code></p> <p>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 <code>false</code>.</p>
<hr/> unit-color	<p><code>unit-color = <color></code></p> <p>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).</p>
<hr/> unit-mode	<p><code>unit-mode = match math text</code></p> <p>Selects which mode (math or text) units 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_EX mode for printing. The standard setting is <code>math</code>.</p>
<hr/> weight-version-mapping	<p><code>weight-version-mapping / <weight> = <version></code></p> <p>Defines how <code>siunitx</code> maps from text font weight to math font version. The pre-defined weights are those used as-standard by <code>autoinst</code>:</p> <ul style="list-style-type: none"> • <code>ul</code> • <code>el</code> • <code>l</code> • <code>sl</code> • <code>m</code> • <code>sb</code> • <code>b</code> • <code>eb</code> • <code>ub</code>

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

2 siunitx-print 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_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 }
```

```
\tl_replace_all:Nvn Required variants.
```

```
5 \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
```

```
6 \box_new:N \l__siunitx_print_tmp_box
```

```
7 \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. As we are doing typesetting (if only in a box), we need to be right at the start of the document: this also avoids any issue with `fontspec`. With a sufficiently recent L^AT_EX 2_ε this is easy; for older kernels, we have to do things manually.

```
8 \IfFormatAtLeastTF { 2020-10-01 }
```

```
9 { \hook_gput_code:nnn { begindocument/end } { siunitx } }
```

```
10 { \tl_put_right:Nn \document }
```

```
11 {
```

```
12 \__siunitx_print_store_fam:n { rm }
```

```
13 \__siunitx_print_store_fam:n { sf }
```

```
14 \__siunitx_print_store_fam:n { tt }
```

```
15 }
```

```
16 \IfFormatAtLeastTF { 2020-10-01 }
```

```
17 { }
```

```
18 { \tl_put_right:Nn \document { \ignorespaces } }
```

```
19 \cs_new_protected:Npn \__siunitx_print_store_fam:n #1
```

```
20 {
```

```
21 \group_begin:
```

```
22 \hbox_set:Nn \l__siunitx_print_tmp_box
```

```
23 {
```

```
24 \ensuremath
```

```
25 {
```

```
26 \use:c { math #1 }
```

```
27 { \int_const:cn { c__siunitx_print_math #1 _int } { \fam } }
```

```
28 }
```

```
29 }
```

```
30 \group_end:
```

```
31 }
```

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

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
\l_siunitx_print_number_mode_tl
\l_siunitx_print_unit_color_tl
\l_siunitx_print_unit_mode_tl
\l_siunitx_print_math_font_bool
\l_siunitx_print_math_version_bool
\l_siunitx_print_math_family_bool
\l_siunitx_print_math_weight_bool
\l_siunitx_print_text_family_tl
\l_siunitx_print_text_series_tl
\l_siunitx_print_text_shape_tl

32 \tl_new:N \l__siunitx_print_number_mode_tl
33 \tl_new:N \l__siunitx_print_unit_mode_tl
34 \keys_define:nn { siunitx }
35 {
36   color .meta:n =
37     { number-color = #1 , unit-color = #1 } ,
38   mode .meta:n =
39     { number-mode = #1 , unit-mode = #1 } ,
40   number-color .tl_set:N =
41     \l__siunitx_print_number_color_tl ,
42   number-mode .choices:nn =
43     { match , math , text }
44     {
45       \tl_set_eq:NN
46         \l__siunitx_print_number_mode_tl \l_keys_choice_tl
47     } ,
48   propagate-math-font .bool_set:N =
49     \l_siunitx_print_math_font_bool ,
50   reset-math-version .bool_set:N =
51     \l_siunitx_print_math_version_bool ,
52   reset-text-family .bool_set:N =
53     \l_siunitx_print_text_family_bool ,
54   reset-text-series .bool_set:N =
55     \l_siunitx_print_text_series_bool ,
56   reset-text-shape .bool_set:N =
57     \l_siunitx_print_text_shape_bool ,
58   text-family-to-math .bool_set:N =
59     \l_siunitx_print_math_family_bool ,
60   text-weight-to-math .bool_set:N =
61     \l_siunitx_print_math_weight_bool ,
62   unit-color .tl_set:N =
63     \l__siunitx_print_unit_color_tl ,
64   unit-mode .choices:nn =
65     { match , math , text }
66     {
67       \tl_set_eq:NN
68         \l__siunitx_print_unit_mode_tl \l_keys_choice_tl
69     }
70 }

```

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

```

\l_siunitx_print_version_ul_tl
\l_siunitx_print_version_el_tl
\l_siunitx_print_version_l_tl
\l_siunitx_print_version_sl_tl
\l_siunitx_print_version_m_tl
\l_siunitx_print_version_sb_tl
\l_siunitx_print_version_b_tl
\l_siunitx_print_version_eb_tl
\l_siunitx_print_version_ub_tl

71 % One set of \enquoe{focussed} options.
72 \keys_define:nn { siunitx / weight-version-mapping }
73 {
74   ul . tl_set:N = \l_siunitx_print_version_ul_tl ,
75   el . tl_set:N = \l_siunitx_print_version_el_tl ,
76   l . tl_set:N = \l_siunitx_print_version_l_tl ,
77   sl . tl_set:N = \l_siunitx_print_version_sl_tl ,

```

```

78   m . tl_set:N = \l__siunitx_print_version_m_tl ,
79   sb . tl_set:N = \l__siunitx_print_version_sb_tl ,
80   b . tl_set:N = \l__siunitx_print_version_b_tl ,
81   eb . tl_set:N = \l__siunitx_print_version_eb_tl ,
82   ub . tl_set:N = \l__siunitx_print_version_ub_tl
83 }

```

(End definition for `\l__siunitx_print_version_ul_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.
`\siunitx_print:nx`

```

84 \cs_new_protected:Npn \siunitx_print:nn #1#2
85 {
86   \tl_if_empty:cTF { l__siunitx_print_ #1 _color_tl }
87   { \use:n }
88   { \exp_args:Nv \textcolor { l__siunitx_print_ #1 _color_tl } }
89   {
90     \use:c
91     {
92       siunitx_print_
93       \tl_use:c { l__siunitx_print_ #1 _mode_tl } :n
94     }
95     {#2}
96   }
97 }
98 \cs_generate_variant:Nn \siunitx_print:nn { nV , nx }

```

(End definition for `\siunitx_print:nn`. This function is documented on page 86.)

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

```

99 \cs_new_protected:Npn \siunitx_print_match:n #1
100 {
101   \mode_if_math:TF
102   { \siunitx_print_math:n {#1} }
103   { \siunitx_print_text:n {#1} }
104 }

```

(End definition for `\siunitx_print_match:n`. This function is documented on page 86.)

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

```

105 \cs_new_protected:Npn \__siunitx_print_replace_font:N #1
106 {
107   \tl_if_empty:NF \l_siunitx_unit_font_tl
108   {
109     \tl_replace_all:NVn #1
110     \l_siunitx_unit_font_tl
111     { \use:n }
112   }
113 }

```

(End definition for `__siunitx_print_replace_font:N`.)

`\c_siunitx_print_weight_uc_tl` Font widths where the *m* for weight is omitted.

```

114 \clist_map_inline:nn { uc , ec , c , sc , sx , x , ex , ux }
115 { \tl_const:cn { c__siunitx_print_weight_ #1 _tl } { m } }

```

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

```
\c__siunitx_print_weight_l_tl  Font widths with one letter.
\c__siunitx_print_weight_m_tl 116 \clist_map_inline:nn { l , m , b }
\c__siunitx_print_weight_b_tl 117 { \tl_const:cn { c__siunitx_print_weight_ #1 _tl } { #1 } }
```

(End definition for `\c__siunitx_print_weight_l_tl`, `\c__siunitx_print_weight_m_tl`, and `\c__siunitx_print_weight_b_tl`.)

`\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_extract_series:Nw
\__siunitx_print_convert_series:n
\__siunitx_print_convert_series:v 118 \cs_new_protected:Npn \siunitx_print_math:n #1
\__siunitx_print_math_version:nn 119 {
\__siunitx_print_math_version:Nn 120 \bool_if:NTF \l__siunitx_print_math_weight_bool
\__siunitx_print_math_auxi:n 121 {
\__siunitx_print_math_auxii:n 122 \tl_set:Nx \l__siunitx_print_tmp_tl
\__siunitx_print_math_auxiii:n 123 { \exp_after:wN \__siunitx_print_extract_series:Nw \f@series ? \q_stop }
\__siunitx_print_math_auxiv:n 124 \tl_if_empty:NTF \l__siunitx_print_tmp_tl
\__siunitx_print_math_auxv:n 125 { \__siunitx_print_math_auxi:n {#1} }
\__siunitx_print_math_aux:Nn 126 { \__siunitx_print_math_version:Nn \l__siunitx_print_tmp_tl {#1} }
\__siunitx_print_math_aux:cn 127 }
\__siunitx_print_math_sub:n 128 { \__siunitx_print_math_auxi:n {#1} }
\__siunitx_print_math_super:n 129 }
\__siunitx_print_math_script:n
\__siunitx_print_math_text:n
```

Look up the math version from the text series. The weight is omitted if it is `m` plus there are either one or two letters, so we have a little work to do. To keep things fast, we use a hash table based lookup rather than a sequence or property list.

```
130 \cs_new:Npn \__siunitx_print_extract_series:Nw #1#2 ? #3 \q_stop
131 {
132 \cs_if_exist:cTF { c__siunitx_print_weight_ #1#2 _tl }
133 { \__siunitx_print_convert_series:v { c__siunitx_print_weight_ #1#2 _tl } }
134 {
135 \cs_if_exist:cTF { c__siunitx_print_weight_ #1 _tl }
136 { \__siunitx_print_convert_series:v { c__siunitx_print_weight_ #1 _tl } }
137 { \__siunitx_print_convert_series:n {#1#2} }
138 }
139 }
140 \cs_new:Npn \__siunitx_print_convert_series:n #1
141 { \tl_use:c { l__siunitx_print_version_ #1 _tl } }
142 \cs_generate_variant:Nn \__siunitx_print_convert_series:n { v }
143 \cs_new_protected:Npn \__siunitx_print_math_auxi:n #1
144 {
145 \bool_if:NTF \l__siunitx_print_math_version_bool
146 { \__siunitx_print_math_version:nn { normal } {#1} }
147 { \__siunitx_print_math_auxii:n {#1} }
148 }
```

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.

```
149 \cs_new_protected:Npn \__siunitx_print_math_version:nn #1#2
150 {
```



```

151 \str_if_eq:VnTF \math@version { #1 }
152 { \__siunitx_print_math_auxii:n {#2} }
153 {
154   \mode_if_math:TF
155   { \text }
156   { \use:n }
157   {
158     \mathversion {#1}
159     \__siunitx_print_math_auxii:n {#2}
160   }
161 }
162 }
163 \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.

```

164 \cs_new_protected:Npn \__siunitx_print_math_auxii:n #1
165 { \ensuremath { \__siunitx_print_math_auxiii:n {#1} } }
166 \cs_new_protected:Npn \__siunitx_print_math_auxiii:n #1
167 {
168   \bool_if:NTF \l__siunitx_print_math_family_bool
169   {
170     \str_case_e:nnF { \f@family }
171     {
172       { \rmdefault } { \__siunitx_print_math_auxv:n }
173       { \sfdefault } { \__siunitx_print_math_aux:Nn \mathsf }
174       { \ttdefault } { \__siunitx_print_math_aux:Nn \mathtt }
175     }
176     { \__siunitx_print_math_auxiv:n }
177   }
178   { \__siunitx_print_math_auxiv:n }
179   {#1}
180 }

```

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.

```

181 \cs_new_protected:Npn \__siunitx_print_math_auxiv:n #1
182 {
183   \bool_if:NTF \l__siunitx_print_math_font_bool
184   {
185     \int_case:nnF \fam
186     {
187       \c__siunitx_print_mathsf_int { \__siunitx_print_math_aux:Nn \mathsf }
188       \c__siunitx_print_mathtt_int { \__siunitx_print_math_aux:Nn \mathtt }
189     }
190     { \use:n }
191   }
192   { \__siunitx_print_math_auxv:n }

```

```

193         {#1}
194     }
195 \cs_new_protected:Npn \__siunitx_print_math_auxv:n #1
196 {
197     \bool_lazy_or:nnTF
198     { \int_compare_p:nNn \fam = { -1 } }
199     { \int_compare_p:nNn \fam = \c__siunitx_print_mathrm_int }
200     { \use:n }
201     { \mathrm }
202     {#1}
203 }

```

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

```

204 \cs_new_protected:Npx \__siunitx_print_math_aux:Nn #1#2
205 {
206     \group_begin:
207     \tl_set:Nn \exp_not:N \l__siunitx_print_tmp_tl {#2}
208     \__siunitx_print_replace_font:N \exp_not:N \l__siunitx_print_tmp_tl
209     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
210     { \char_generate:nn { '\_ } { 8 } }
211     { \exp_not:N \__siunitx_print_math_sub:n }
212     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
213     { ^ }
214     { \exp_not:N \__siunitx_print_math_super:n }
215     #1 { \exp_not:N \tl_use:N \exp_not:N \l__siunitx_print_tmp_tl }
216     \group_end:
217 }
218 \cs_generate_variant:Nn \__siunitx_print_math_aux:Nn { c }
219 \cs_new_protected:Npx \__siunitx_print_math_sub:n #1
220 {
221     \char_generate:nn { '\_ } { 8 }
222     { \exp_not:N \__siunitx_print_math_script:n {#1} }
223 }
224 \cs_new_protected:Npn \__siunitx_print_math_super:n #1
225 { ^ { \__siunitx_print_math_script:n {#1} } }
226 \cs_new_protected:Npn \__siunitx_print_math_script:n #1
227 {
228     \group_begin:
229     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
230     \__siunitx_print_replace_font:N \l__siunitx_print_tmp_tl
231     \tl_use:N \l__siunitx_print_tmp_tl
232     \group_end:
233 }

```

For tex4ht, we need to have category code 12 ^ tokens in math mode. We handle that by intercepting at the first auxiliary that makes sense.

```

234 \AtBeginDocument
235 {
236     \@ifpackageloaded { tex4ht }
237     {
238         \cs_set_protected:Npn \__siunitx_print_math_auxii:n #1
239         {
240             \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
241             \exp_args:NNnx \tl_replace_all:Nnn \l__siunitx_print_tmp_tl

```

```

242         { ^ } { \token_to_str:N ^ }
243         \ensuremath { \exp_args:NV \__siunitx_print_math_auxiii:n \l__siunitx_print_tmp_t
244     }
245 }
246 { }
247 }

```

(End definition for `\siunitx_print_math:n` and others. This function is documented on page 86.)

```

\siunitx_print_text:n Typesetting in text mode is easy in font control terms but more tricky in the manipulation
  \__siunitx_print_text_replace:n of the input. The easy part comes first.
  \__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
248 \cs_new_protected:Npn \siunitx_print_text:n #1
249 {
250   \text
251   {
252     \bool_if:NT \l__siunitx_print_text_family_bool
253     {
254       \fontfamily { \familydefault }
255       \selectfont
256     }
257     \bool_if:NT \l__siunitx_print_text_series_bool
258     {
259       \fontseries { \seriesdefault }
260       \selectfont
261     }
262     \bool_if:NT \l__siunitx_print_text_shape_bool
263     {
264       \fontshape { \shapedefault }
265       \selectfont
266     }
267     \__siunitx_print_text_replace:n {#1}
268   }
269 }

```

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

```

270 \cs_new_protected:Npn \__siunitx_print_text_replace:n #1
271 {
272   \group_begin:
273   \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
274   \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
275   \tl_use:N \l__siunitx_print_tmp_tl
276   \group_end:
277 }
278 \cs_new_protected:Npx \__siunitx_print_text_replace:N #1
279 {
280   \__siunitx_print_replace_font:N #1
281   \exp_not:N \__siunitx_print_text_replace:NNn #1
282   \exp_not:N \mathord { }
283   \exp_not:N \pm
284   { \exp_not:N \textpm }
285   \exp_not:N \mp
286   { \exp_not:n { \ensuremath { \mp } } } }
287 -
288   { \exp_not:N \textminus }

```

```

289 \char_generate:nn { '\_ } { 8 }
290 { \exp_not:N \__siunitx_print_text_sub:n }
291 ^
292 { \exp_not:N \__siunitx_print_text_super:n }
293 \exp_not:N \q_recursion_tail
294 { ? }
295 \exp_not:N \q_recursion_stop
296 }
297 \cs_new_protected:Npn \__siunitx_print_text_replace:NnN #1#2#3
298 {
299 \quark_if_recursion_tail_stop:N #2
300 \tl_replace_all:Nnn #1 {#2} {#3}
301 \__siunitx_print_text_replace:NnN #1
302 }

```

When the bidi package is loaded, we need to make sure that `\text` is doing the correct thing.

```

303 \sys_if_engine_xetex:T
304 {
305 \AtBeginDocument
306 {
307 \ifpackageloaded { bidi }
308 {
309 \cs_set_protected:Npn \__siunitx_print_text_replace:n #1
310 {
311 \group_begin:
312 \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
313 \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
314 \LRE { \tl_use:N \l__siunitx_print_tmp_tl }
315 \group_end:
316 }
317 }
318 { }
319 }
320 }

```

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.

```

321 \cs_new_protected:Npn \__siunitx_print_text_sub:n #1
322 {
323 \__siunitx_print_text_scripts:NnN
324 \textsubscript {#1} \__siunitx_print_text_super:n
325 }
326 \cs_new_protected:Npn \__siunitx_print_text_super:n #1
327 {
328 \__siunitx_print_text_scripts:NnN
329 \textsuperscript {#1} \__siunitx_print_text_sub:n
330 }
331 \cs_new_protected:Npn \__siunitx_print_text_scripts:NnN #1#2#3
332 {
333 \cs_set_protected:Npn \__siunitx_print_text_scripts:
334 {
335 \if_meaning:w \l_peek_token #3
336 \exp_after:wN \__siunitx_print_text_scripts_two:NnNn
337 \else:

```

```

338         \exp_after:wN \__siunitx_print_text_scripts_one:Nn
339         \fi:
340         #1 {#2}
341     }
342     \peek_after:Nw \__siunitx_print_text_scripts:
343 }
344 \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.

```

345 \cs_new_protected:Npn \__siunitx_print_text_scripts_one:Nn #1#2
346 {
347     \group_begin:
348     \tl_set:Nn \l__siunitx_print_tmp_tl {#2}
349     \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
350     \exp_args:NNV \group_end:
351     #1 \l__siunitx_print_tmp_tl
352 }

```

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.

```

353 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:NnNn #1#2#3#4
354 {
355     \cs_if_eq:NNTF #1 \textsubscript
356     { \__siunitx_print_text_scripts_two:nn {#4} {#2} }
357     { \__siunitx_print_text_scripts_two:nn {#2} {#4} }
358 }
359 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:nn #1#2
360 {
361     \group_begin:
362     \exp_not:N \m@th
363     \exp_not:N \ensuremath
364     {
365         ~ { \exp_not:N \__siunitx_print_text_scripts_two:n {#1} }
366         \char_generate:nn { '\_ } { 8 }
367         { \exp_not:N \__siunitx_print_text_scripts_two:n {#2} }
368     }
369     \group_end:
370 }
371 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:n #1
372 {
373     \mbox
374     {
375         \fontsize \sf@size \z@ \selectfont
376         \__siunitx_print_text_scripts_one:Nn \use:n {#1}
377     }
378 }

```

(End definition for `\siunitx_print_text:n` and others. This function is documented on page 86.)

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.

```

379 \keys_set:nn { siunitx }
380 {
381     color                =      ,
382     mode                 = math ,
383     number-color         =      ,
384     number-mode          = math ,
385     propagate-math-font = false ,
386     reset-math-version   = true  ,
387     reset-text-shape     = true  ,
388     reset-text-series    = true  ,
389     reset-text-family    = true  ,
390     text-family-to-math  = false ,
391     text-weight-to-math  = false ,
392     unit-color           =      ,
393     unit-mode            = math
394 }

```

These are separate as they all fall inside the same key.

```

395 \keys_set:nn { siunitx / weight-version-mapping }
396 {
397     ul = light ,
398     el = light ,
399     l  = light ,
400     sl = light ,
401     m  = normal ,
402     sb = bold   ,
403     b  = bold   ,
404     eb = bold   ,
405     ub = bold
406 }
407 </package>

```

Part VII

siunitx-quantity — Quantities

<code>\siunitx_quantity:nn</code>	<code>\siunitx_quantity:nn {⟨number⟩} {⟨unit⟩}</code>
-----------------------------------	---

Parses the $\langle number \rangle$ and the $\langle unit \rangle$ as detailed for `\siunitx_number_parse:nn` and `\siunitx_unit_format:nN`, then prints the results using `\siunitx_print:nn`.

<code>\siunitx_quantity_print:nn</code>	<code>\siunitx_quantity_print:nn {⟨number⟩} {⟨unit⟩}</code>
<code>\siunitx_quantity_print:(nV VV xV)</code>	

A low-level function which prints the quantity directly: there is no processing applied to either the $\langle number \rangle$ or $\langle unit \rangle$. The two parts are printed using `\siunitx_print:nn` and appropriate spacing and break-prevention is applied.

<code>allow-quantity-breaks</code>	<code>allow-quantity-breaks = true false</code>
------------------------------------	---

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

<code>prefix-mode</code>	<code>prefix-mode = combine-exponent extract-exponent input</code>
--------------------------	--

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

<code>quantity-product</code>	<code>quantity-product = ⟨tokens⟩</code>
-------------------------------	--

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

<code>separate-uncertainty-units</code>	<code>separate-uncertainty-units = bracket repeat single</code>
---	---

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

1 siunitx-quantity 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_quantity⟩
```

1.1 Initial set-up

Scratch space.

```
\l__siunitx_quantity_tmp_fp
\l__siunitx_quantity_tmp_tl
3 \tl_new:N \l__siunitx_quantity_tmp_fp
4 \tl_new:N \l__siunitx_quantity_tmp_tl

(End definition for \l__siunitx_quantity_tmp_fp and \l__siunitx_quantity_tmp_tl.)
```

1.2 Main formatting routine

Purely internal for the present.

```
\l__siunitx_quantity_bracket_close_tl
\l__siunitx_quantity_bracket_open_tl
5 \tl_new:N \l__siunitx_quantity_bracket_close_tl
6 \tl_new:N \l__siunitx_quantity_bracket_open_tl
7 \tl_set:Nn \l__siunitx_quantity_bracket_open_tl { ( }
8 \tl_set:Nn \l__siunitx_quantity_bracket_close_tl { ) }

(End definition for \l__siunitx_quantity_bracket_close_tl and \l__siunitx_quantity_bracket_
open_tl.)

\l_siunitx_quantity_prefix_mode_tl
\l__siunitx_quantity_break_bool
\l__siunitx_quantity_product_tl
\l__siunitx_quantity_uncert_bracket_bool
\l__siunitx_quantity_uncert_repeat_bool
9 \tl_new:N \l_siunitx_quantity_prefix_mode_tl
10 \bool_new:N \l__siunitx_quantity_uncert_bracket_bool
11 \bool_new:N \l__siunitx_quantity_uncert_repeat_bool
12 \keys_define:nn { siunitx }
13 {
14   allow-quantity-breaks .bool_set:N =
15     \l__siunitx_quantity_break_bool ,
16   prefix-mode .choices:nn =
17     { combine-exponent , extract-exponent , input }
18     { \tl_set_eq:NN \l_siunitx_quantity_prefix_mode_tl \l_keys_choice_tl } ,
19   quantity-product .tl_set:N =
20     \l__siunitx_quantity_product_tl ,
21   separate-uncertainty-units .choice: ,
22   separate-uncertainty-units / bracket .code:n =
23     {
24       \bool_set_true:N \l__siunitx_quantity_uncert_bracket_bool
25       \bool_set_false:N \l__siunitx_quantity_uncert_repeat_bool
26     } ,
27   separate-uncertainty-units / repeat .code:n =
28     {
29       \bool_set_false:N \l__siunitx_quantity_uncert_bracket_bool
30       \bool_set_true:N \l__siunitx_quantity_uncert_repeat_bool
31     } ,
32   separate-uncertainty-units / single .code:n =
33     {
34       \bool_set_false:N \l__siunitx_quantity_uncert_bracket_bool
35       \bool_set_false:N \l__siunitx_quantity_uncert_repeat_bool
36     }
37 }
```

(End definition for \l_siunitx_quantity_prefix_mode_tl and others. This variable is documented on page ??.)


```

\l_siunitx_quantity_number_tl
\l__siunitx_quantity_unit_tl

```

```

38 \tl_new:N \l__siunitx_quantity_number_tl
39 \tl_new:N \l__siunitx_quantity_unit_tl

```

(End definition for `\l__siunitx_quantity_number_tl` and `\l__siunitx_quantity_unit_tl`.)

\siunitx_quantity:nn

```

\__siunitx_quantity_parsed:nn
_siunitx_quantity_parsed_combine-exponent:n
_siunitx_quantity_parsed_combine-exponent:n
\__siunitx_quantity_parsed_input:n
\__siunitx_quantity_parsed_aux:w
\__siunitx_quantity_parsed_aux:nnw
\__siunitx_quantity_parsed_aux:nnn

```

For quantities, there is bit to do to combine things. The first question is whether we are parsing at all: if not, things are quite short. Notice that within this group we turn off bracketing in the number formatter: we have to deal with quantity-based brackets instead.

```

40 \cs_new_protected:Npn \siunitx_quantity:nn #1#2
41 {
42   \group_begin:
43     \siunitx_unit_options_apply:n {#2}
44     \tl_if_blank:nTF {#1}
45     {
46       \siunitx_unit_format:nN {#2} \l__siunitx_quantity_unit_tl
47       \siunitx_print:nV { unit } \l__siunitx_quantity_unit_tl
48     }
49     {
50       \bool_if:NTF \l_siunitx_number_parse_bool
51       { \__siunitx_quantity_parsed:nn {#1} {#2} }
52       {
53         \tl_set:Nn \l__siunitx_quantity_number_tl { \ensuremath {#1} }
54         \siunitx_unit_format:nN {#2} \l__siunitx_quantity_unit_tl
55         \siunitx_quantity_print:VV
56           \l__siunitx_quantity_number_tl \l__siunitx_quantity_unit_tl
57       }
58     }
59   \group_end:
60 }

```

For parsed numbers, we have two major questions to think about: whether we are combining prefixes, and whether we have a multi-part numbers to handle. Number processing has to be delayed it needs to come after any extracted exponent is combined.

```

61 \cs_new_protected:Npn \__siunitx_quantity_parsed:nn #1#2
62 {
63   \bool_set_false:N \l_siunitx_number_bracket_ambiguous_bool
64   \siunitx_number_parse:nN {#1} \l__siunitx_quantity_number_tl
65   \use:c { __siunitx_quantity_parsed_ \l_siunitx_quantity_prefix_mode_tl :n } {#2}
66   \tl_set:Nx \l__siunitx_quantity_number_tl
67     { \siunitx_number_output:NN \l__siunitx_quantity_number_tl \q_nil }
68   \exp_after:wN \__siunitx_quantity_parsed_aux:w \l__siunitx_quantity_number_tl \q_stop
69 }
70 \cs_new_protected:cpn { __siunitx_quantity_parsed_combine-exponent:n } #1
71 {
72   \siunitx_number_process:NN \l__siunitx_quantity_number_tl \l__siunitx_quantity_number_tl
73   \exp_args:NV \__siunitx_quantity_extract_exp:nNN
74     \l__siunitx_quantity_number_tl \l__siunitx_quantity_tmp_fp \l__siunitx_quantity_number_
75   \siunitx_unit_format_combine-exponent:nnN {#1}
76     \l__siunitx_quantity_tmp_fp \l__siunitx_quantity_unit_tl
77 }
78 \cs_new_protected:cpn { __siunitx_quantity_parsed_extract-exponent:n } #1
79 {

```

```

80 \siunitx_unit_format_extract_prefixes:nNN {#1}
81 \l__siunitx_quantity_unit_tl \l__siunitx_quantity_tmp_fp
82 \tl_set:Nx \l__siunitx_quantity_number_tl
83 {
84   \siunitx_number_adjust_exponent:Nn
85   \l__siunitx_quantity_number_tl \l__siunitx_quantity_tmp_fp
86 }
87 \siunitx_number_process:NN \l__siunitx_quantity_number_tl \l__siunitx_quantity_number_tl
88 }
89 \cs_new_protected:Npn \__siunitx_quantity_parsed_input:n #1
90 {
91   \siunitx_number_process:NN \l__siunitx_quantity_number_tl \l__siunitx_quantity_number_tl
92   \siunitx_unit_format:nN {#1} \l__siunitx_quantity_unit_tl
93 }

```

To find out if we need to work harder, we first need to split the formatted number into the constituent parts. That is done using the table-like approach: that avoids needing to both check the settings and break down the input separately.

```

94 \cs_new_protected:Npn \__siunitx_quantity_parsed_aux:w
95   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
96   #8 \q_nil #9 \q_stop
97   { \__siunitx_quantity_parsed_aux:nnnw {#1} {#2#3#4#5} {#6#7#8} #9 \q_stop }
98 \cs_new_protected:Npn \__siunitx_quantity_parsed_aux:nnnw
99   #1#2#3 #4 \q_nil #5 \q_nil #6 \q_stop
100   { \__siunitx_quantity_parsed_aux:nnnn {#1} {#2} {#3#4} {#5#6} }
101 \cs_new_protected:Npn \__siunitx_quantity_parsed_aux:nnnn #1#2#3#4
102   {
103     \tl_if_blank:nTF {#3}
104     { \siunitx_quantity_print:nV {#1#2#4} \l__siunitx_quantity_unit_tl }
105     {
106       \bool_if:NTF \l__siunitx_quantity_uncert_bracket_bool
107       {
108         \siunitx_quantity_print:xV
109         {
110           \exp_not:n {#1}
111           \exp_not:V \l__siunitx_quantity_bracket_open_tl
112           \exp_not:n {#2#3}
113           \exp_not:V \l__siunitx_quantity_bracket_close_tl
114           \exp_not:n {#4}
115         }
116         \l__siunitx_quantity_unit_tl
117       }
118       {
119         \bool_if:NTF \l__siunitx_quantity_uncert_repeat_bool
120         {
121           \siunitx_quantity_print:nV {#1#2}
122           \l__siunitx_quantity_unit_tl
123           \siunitx_quantity_print:nV { { } #3 }
124           \l__siunitx_quantity_unit_tl
125           \siunitx_print:nn { number } { { } #4 }
126         }
127         { \siunitx_quantity_print:nV {#1#2#3#4} \l__siunitx_quantity_unit_tl }
128       }
129     }

```

```
130 }
```

(End definition for `\siunitx_quantity:nn` and others. This function is documented on page 99.)

```
\_siunitx_quantity_extract_exp:nnNN
\_siunitx_quantity_extract_exp:nnnnnnnnNN
```

To extract the exponent part for a combined prefix, we decompose the value and remove it.

```
131 \cs_new_protected:Npn \_siunitx_quantity_extract_exp:nnNN #1#2#3
132 { \_siunitx_quantity_extract_exp:nnnnnnnnNN #1 #2 #3 }
133 \cs_new_protected:Npn \_siunitx_quantity_extract_exp:nnnnnnnnNN #1#2#3#4#5#6#7#8#9
134 {
135   \fp_set:Nn #8 {#6#7}
136   \tl_set:Nx #9
137     { {#1} {#2} {#3} {#4} {#5} { } { 0 } }
138 }
```

(End definition for `_siunitx_quantity_extract_exp:nnNN` and `_siunitx_quantity_extract_exp:nnnnnnnnNN`.)

```
\siunitx_quantity_print:nn
\siunitx_quantity_print:nV
\siunitx_quantity_print:VV
\siunitx_quantity_print:xV
```

For printing a single part of a quantity. This is needed for compound quantities and so is public: that's also the reason for passing both argument explicitly.

```
139 \cs_new_protected:Npn \siunitx_quantity_print:nn #1#2
140 {
141   \siunitx_print:nn { number } {#1}
142   \tl_if_blank:nF {#2}
143   {
144     \tl_use:N \l__siunitx_quantity_product_tl
145     \bool_if:NF \l__siunitx_quantity_break_bool { \nobreak }
146     \siunitx_print:nn { unit } {#2}
147   }
148 }
149 \cs_generate_variant:Nn \siunitx_quantity_print:nn { nV , VV , xV }
```

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

1.3 Standard settings for module options

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

```
150 \keys_set:nn { siunitx }
151 {
152   allow-quantity-breaks      = false ,
153   prefix-mode                 = input ,
154   quantity-product           = \, ,
155   separate-uncertainty-units = bracket
156 }
```

1.4 Adjustments to units

```
\_siunitx_quantity_non_latin:n
\_siunitx_quantity_non_latin:nnnn
```

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

```
157 \bool_lazy_or:nnTF
158 { \sys_if_engine luatex_p: }
159 { \sys_if_engine xetex_p: }
160 {
161   \cs_new:Npn \_siunitx_quantity_non_latin:n #1
162     { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
```

```

163 }
164 {
165   \cs_new:Npn \__siunitx_quantity_non_latin:n #1
166   {
167     \exp_last_unbraced:Nf \__siunitx_quantity_non_latin:nnnn
168     { \char_to_utfviii_bytes:n {#1} }
169   }
170   \cs_new:Npn \__siunitx_quantity_non_latin:nnnn #1#2#3#4
171   {
172     \exp_after:wN \exp_after:wN \exp_after:wN
173     \exp_not:N \char_generate:nn {#1} { 13 }
174     \exp_after:wN \exp_after:wN \exp_after:wN
175     \exp_not:N \char_generate:nn {#2} { 13 }
176   }
177 }

```

(End definition for __siunitx_quantity_non_latin:n and __siunitx_quantity_non_latin:nnnn.)

\degree The `\degree` unit is re-declared here: this is needed for using it in quantities. This is done here as it avoids a dependency in `siunitx-unit` on options it does not contain.

```

178 \siunitx_declare_unit:Nxn \degree
179 { \__siunitx_quantity_non_latin:n { "00B0 } }
180 { quantity-product = { } }

```

(End definition for `\degree`. This function is documented on page [135](#).)

```

181 \endpackage

```

Part VIII

siunitx-symbol – Symbol-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`.)

A small number of commands are needed from the companion fonts when working with 8-bit engines. These are loaded by modern L^AT_EX 2_ε kernel, so for older ones, force loading them using `textcomp`.

```
5 \AtBeginDocument
6 {
7   \cs_if_free:cT { T@TS1 }
8   { \RequirePackage { textcomp } }
9 }
```

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

```
\_siunitx_symbol_non_latin:nnnn 10 \bool_lazy_or:nnTF
11 { \sys_if_engine luatex_p: }
12 { \sys_if_engine xetex_p: }
13 {
14   \cs_new:Npn \_siunitx_symbol_non_latin:n #1
15   { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
16 }
17 {
18   \cs_new:Npn \_siunitx_symbol_non_latin:n #1
19   {
20     \exp_last_unbraced:Nf \_siunitx_symbol_non_latin:nnnn
21     { \char_to_utfviii_bytes:n {#1} }
22   }
23   \cs_new:Npn \_siunitx_symbol_non_latin:nnnn #1#2#3#4
24   {
25     \exp_after:wN \exp_after:wN \exp_after:wN
26     \exp_not:N \char_generate:nn {#1} { 13 }
27     \exp_after:wN \exp_after:wN \exp_after:wN
28     \exp_not:N \char_generate:nn {#2} { 13 }
29   }
30 }
```

(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.

```

31 \prg_new_protected_conditional:Npnn \_siunitx_symbol_if_replace:Nn #1#2 { T , TF }
32 {
33   \group_begin:
34   \tl_set:Nx \l__siunitx_symbol_tmpa_tl { \_siunitx_symbol_non_latin:n {#2} }
35   \protected@edef \l__siunitx_symbol_tmpa_tl
36   { \exp_not:N \mathrm { \l__siunitx_symbol_tmpa_tl } }
37   \keys_set:nn { siunitx } { parse-units = false }
38   \siunitx_unit_format:nN {#1} \l__siunitx_symbol_tmpb_tl
39   \str_if_eq:VVTF \l__siunitx_symbol_tmpa_tl \l__siunitx_symbol_tmpb_tl
40   {
41     \group_end:
42     \prg_return_true:
43   }
44   {
45     \group_end:
46     \prg_return_false:
47   }
48 }

```

(End definition for `_siunitx_symbol_if_replace:NnT`.)

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”.

```

49 \AtBeginDocument
50 {
51   \_siunitx_symbol_if_replace:NnT \arcminute { "02B9 }
52   {
53     \siunitx_declare_unit:Nx \arcminute
54     { \exp_not:N \ensuremath { { } ' } } }
55   }
56   \_siunitx_symbol_if_replace:NnT \arcsecond { "02BA }
57   {
58     \siunitx_declare_unit:Nx \arcsecond
59     { \exp_not:N \ensuremath { { } '' } } }
60   }

```

For `\degree`, 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.

```

61 \_siunitx_symbol_if_replace:NnT \degree { "00B0 }
62 {
63   \@ifpackageloaded { fontspec }
64   { }
65   {
66     \siunitx_declare_unit:Nxn \degree
67     {
68       \siunitx_print_text:n
69       { \_siunitx_symbol_non_latin:n { "00B0 } }
70     }
71     { quantity-product = { } }
72   }
73 }

```

For `\degreeCelsius`, much the same to think about but the comparison must be done by hand.

```

74 \group_begin:
75 \tl_set:Nx \l__siunitx_symbol_tmpa_tl { \__siunitx_symbol_non_latin:n { "00B0 } C }
76 \protected@edef \l__siunitx_symbol_tmpa_tl
77 { \exp_not:N \mathrm { \l__siunitx_symbol_tmpa_tl } }
78 \keys_set:nn { siunitx } { parse-units = false }
79 \siunitx_unit_format:nN { \degreeCelsius } \l__siunitx_symbol_tmpb_tl
80 \str_if_eq:VTF \l__siunitx_symbol_tmpa_tl \l__siunitx_symbol_tmpb_tl
81 {
82   \group_end:
83   \ifpackageloaded { fontspec }
84   { }
85   {
86     \siunitx_declare_unit:Nx \degreeCelsius
87     {
88       \siunitx_print_text:n
89       { \__siunitx_symbol_non_latin:n { "00B0 } } C
90     }
91   }
92 }
93 { \group_end: }

```

For `\ohm`, there is a math mode symbol we can use, so there has to be a mode-dependent definition.

```

94 \__siunitx_symbol_if_replace:NnT \ohm { "03A9 }
95 {
96   \siunitx_declare_unit:Nx \ohm
97   {
98     \exp_not:N \ifmmode
99     \cs_if_exist:NTF \upOmega
100     { \exp_not:N \upOmega }
101     { \exp_not:N \Omega }
102   \exp_not:N \else
103     \siunitx_print_text:n
104     {
105       \bool_lazy_or:nnTF
106       { \sys_if_engine luatex_p: }
107       { \sys_if_engine xetex_p: }
108       { \__siunitx_symbol_non_latin:n { "03A9 } }
109       { \exp_not:N \textohm }
110     }
111   \exp_not:N \fi
112 }
113 }

```

Only a text mode command is available for `\micro` in the standard set up.

```

114 \__siunitx_symbol_if_replace:NnT \micro { "03BC }
115 {
116   \siunitx_declare_prefix:Nnx \micro { -6 }
117   {
118     \siunitx_print_text:n
119     {
120       \bool_lazy_or:nnTF

```

```

121         { \sys_if_engine luatex_p: }
122         { \sys_if_engine xetex_p: }
123         { \_siunitx_symbol_non_latin:n { "00B5 } }
124         { \exp_not:N \textmu }
125     }
126 }
127 }
128 }

```

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 use the TS1 version for text. Otherwise much the same as `\textmu` support. It's hard to check for the product symbol, so we just go with it an hope for the best!

```

129 \AtBeginDocument
130 {
131     \group_begin:
132     \tl_set:Nn \l__siunitx_symbol_tmpa_tl
133       { { } { } { 2 } { } { } { } { 1 } }
134     \tl_set:Nx \l__siunitx_symbol_tmpa_tl
135       { \siunitx_number_output:N \l__siunitx_symbol_tmpa_tl }
136     \tl_set:Nn \l__siunitx_symbol_tmpb_tl { 2 \times 10 ^ { 1 } }
137     \tl_if_eq:NNTF \l__siunitx_symbol_tmpa_tl \l__siunitx_symbol_tmpb_tl
138       {
139         \group_end:
140         \keys_set:nn { siunitx }
141           {
142             exponent-product =
143               \ifmmode \times \else \texttimes \fi ,
144             product-symbol =
145               \ifmmode \times \else \texttimes \fi
146           }
147       }
148     { \group_end: }
149 }

```

1.1 Bookmark definitions

Inside PDF strings we disable the text printing function. The definition of `\ohm` is also reset as otherwise engine-dependent strings are generated (XeTeX and LuaTeX give different outcomes using for example `\textohm`).

```

150 \AtBeginDocument
151 {
152     \@ifpackageloaded { hyperref }
153     {
154         \exp_args:Nx \pdfstringdefDisableCommands
155         {
156             \cs_set_eq:NN \siunitx_print_text:n \exp_not:N \use:n
157             \siunitx_declare_unit:Nn \exp_not:N \ohm
158             { \_siunitx_symbol_non_latin:n { "03A9 } }
159         }
160     }
161     { }
162 }

```


163 `</package>`

Part IX

siunitx-table – Formatting numbers in tables

1 Numbers in tables

`\siunitx_cell_begin:w`
`\siunitx_cell_end:`

`\siunitx_cell_begin:w` $\langle preamble \rangle$ `\ignorespaces`
 $\langle content \rangle$
`\siunitx_cell_end:`

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

1.1 Key–value options

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

`table-align-comparator`

`table-align-comparator = true|false`

`table-align-exponent`

`table-align-exponent = true|false`

`table-align-text-after`

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

`table-align-text-before`

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

`table-align-uncertainty`

`table-align-uncertainty = true|false`

`table-alignment`

`table-alignment = center|left|right`

`table-alignment-mode`

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

`table-auto-round`

`table-auto-round = true|false`

`table-column-width`

`table-column-width = $\langle width \rangle$`

`table-fixed-width`

`table-fixed-width = true|false`

\langle format \rangle

table-number-alignment = center|left|right

table-text-alignment = center|left|right

2 siunitx-table implementation

Start the DocStrip guards.

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

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

```
2  $\langle @@=siunitx\_table \rangle$ 
```

```
\l__siunitx_table_tmp_box
```

Scratch space.

```
\l__siunitx_table_tmp_dim
```

```
3 \box_new:N \l__siunitx_table_tmp_box
```

```
\l__siunitx_table_tmp_tl
```

```
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`.)

2.1 Interface functions

```
\l__siunitx_table_text_bool
```

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:
```

The start and end of the cell need to deal with the possibility of a cell containing only text.

```
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 110.)

2.2 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 `\text_lowercase: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.)

2.3 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:V VV
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
\__siunitx_table_split_loop:NNN track where we are up to based on the content of the token lists.
\__siunitx_table_split_group:NNNn 122 \cs_new_protected:Npn \__siunitx_table_split:nNNN #1#2#3#4
\__siunitx_table_split_token:NNNN 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:No #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`.)

2.4 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 Opt plus 0.5fill
\kern Opt
#
\hskip Opt 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`
`_siunitx_table_align_left:n`
`_siunitx_table_align_right:n`
`_siunitx_table_align_auxi:n`
`_siunitx_table_align_auxii: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.

```

183 \cs_new_protected:Npn \_siunitx\_table\_align\_center:n #1
184 { \_siunitx\_table\_align\_auxi:n {#1} { Opt-plus-0.5fill } }
185 \cs_new_protected:Npn \_siunitx\_table\_align\_left:n #1
186 { \_siunitx\_table\_align\_auxi:n {#1} { Opt } }
187 \cs_new_protected:Npn \_siunitx\_table\_align\_right:n #1
188 { \_siunitx\_table\_align\_auxi:n {#1} { Opt-plus-1fill } }
189 \cs_new_protected:Npn \_siunitx\_table\_align\_auxi:n #1#2
190 {
191   \bool_if:NTF \l\_siunitx\_table\_fixed\_width\_bool
192     { \hbox_to_wd:n \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{ 0pt~plus~-0.5fill }
210         \_siunitx_table_align_auxii:nn {#1} {#2}
211         \_siunitx_table_skip:n { 0pt~plus~-0.5fill }
212     }
213 }
214 { }
215 }

```

(End definition for _siunitx_table_align_center:n and others.)

2.5 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.)

2.6 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:.)

\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 } { retain-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`.)

2.7 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_output: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_nil #4 \q_stop
377 {
378   \hbox_set:Nn \l__siunitx_table_tmp_box
379     { \ensuremath { \__siunitx_table_cleanup_decimal:w #4 } }
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#3 } }
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.)

2.8 Printing numbers in cells: main functions

\l__siunitx_table_before_box For alignment of text outside of a number.

```

\l__siunitx_table_after_box
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.)

\l__siunitx_table_carry_dim 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.)

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

```

\l__siunitx_table_align_exponent_bool
\l__siunitx_table_align_after_bool
\l__siunitx_table_align_before_bool
\l__siunitx_table_align_uncertainty_bool
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:w
\_siunitx_table_print_marker_aux:w
\_siunitx_table_print_format: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_output: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_nil #4 \q_nil #4 \q_stop
474 {
475   \hbox_set:Nn \l__siunitx_table_integer_box
476     { \siunitx_print:nn { number } { #1#2#3 } }
477   \hbox_set:Nn \l__siunitx_table_decimal_box
478     {
479       \exp_args:Nnx \siunitx_print:nn { number }
480       { \__siunitx_table_print_marker_aux:w #4 }

```

```

481     }
482     \_siunitx_table_center_marker:
483     \box_use_drop:N \l__siunitx_table_integer_box
484     \box_use_drop:N \l__siunitx_table_decimal_box
485 }
486 \cs_new:Npn \_siunitx_table_print_marker_aux:w
487 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
488 {
489     \exp_not:n {#1#2#3#4#5}
490     \tl_if_blank:nT {#1#2#3#4#5} { { } }
491     \exp_not:n {#6#7}
492 }

```

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.

```

493 \cs_new_protected:Npn \_siunitx_table_print_format:nnn #1#2#3
494 {
495     \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_before_model_tl }
496     \hbox_set:Nn \l__siunitx_table_before_box {#1}
497     \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { Opt }
498     {
499         \box_clear:N \l__siunitx_table_before_box
500         #1
501     }
502     \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
503     { \box_wd:N \l__siunitx_table_tmp_box }
504     {
505         \_siunitx_table_fil:
506         \hbox_unpack:N \l__siunitx_table_before_box
507     }
508     \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
509     \group_begin:
510         \bool_if:NT \l__siunitx_table_auto_round_bool
511         {
512             \exp_args:Nx \keys_set:nn { siunitx }
513             {
514                 round-mode      = places ,
515                 round-pad      = true ,
516                 round-precision =
517                 \exp_after:wN \_siunitx_table_print_format:nnnnnn
518                 \l__siunitx_table_format_tl
519             }
520         }
521         \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
522         \exp_args:NNNV \group_end:
523         \tl_set:Nn \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
524         \tl_set:Nx \l__siunitx_table_tmp_tl
525         {
526             \siunitx_number_output:NN \l__siunitx_table_model_tl \q_nil
527             \exp_not:N \q_mark
528             \siunitx_number_output:NN \l__siunitx_table_tmp_tl \q_nil
529         }

```



```

530 \exp_after:wN \_siunitx_table_print_format_auxi:w
531 \l__siunitx_table_tmp_tl \q_stop
532 \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_after_model_tl }
533 \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
534 { \box_wd:N \l__siunitx_table_tmp_box + \l__siunitx_table_carry_dim }
535 {
536   \bool_if:NT \l__siunitx_table_align_after_bool
537   { \skip_horizontal:n { \l__siunitx_table_carry_dim } }
538   #3
539   \_siunitx_table_fil:
540 }
541 \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
542 {
543   \box_use_drop:N \l__siunitx_table_before_box
544   \box_use_drop:N \l__siunitx_table_integer_box
545   \box_use_drop:N \l__siunitx_table_decimal_box
546   \box_use_drop:N \l__siunitx_table_after_box
547 }
548 }
549 \cs_new:Npn \_siunitx_table_print_format:nnnnnn #1#2#3#4#5#6#7
550 { 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).

```

551 \cs_new_protected:Npn \_siunitx_table_print_format_auxi:w
552 #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
553 {
554   \_siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1}
555   \bool_if:NTF \l__siunitx_table_align_before_bool
556   {
557     \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
558     { \box_wd:N \l__siunitx_table_tmp_box }
559     {
560       \_siunitx_table_fil:
561       \tl_if_blank:nF {#3}
562       { \siunitx_print:nn { number } {#3} }
563     }
564   }
565   {
566     \_siunitx_table_print_format_box:Nn \l__siunitx_table_integer_box {#3}
567     \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
568     {
569       \box_wd:N \l__siunitx_table_before_box
570       + \box_wd:N \l__siunitx_table_tmp_box
571       - \box_wd:N \l__siunitx_table_integer_box
572     }
573     {
574       \_siunitx_table_fil:
575       \hbox_unpack:N \l__siunitx_table_before_box
576     }
577   }
578   \_siunitx_table_print_format_auxii:w #2 \q_mark #4 \q_stop
579 }

```

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`”.)

```

580 \cs_new_protected:Npn \__siunitx_table_print_format_auxii:w
581   #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
582   {
583     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
584     \bool_lazy_and:nnTF
585       { \l__siunitx_table_align_comparator_bool }
586       { \dim_compare_p:nNn { \box_wd:N \l__siunitx_table_integer_box } > { Opt } }
587       {
588         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
589         {
590           \box_wd:N \l__siunitx_table_integer_box
591         + \box_wd:N \l__siunitx_table_tmp_box
592         }
593         {
594           \hbox_unpack:N \l__siunitx_table_integer_box
595           \__siunitx_table_fil:
596           \siunitx_print:nn { number } {#4#5}
597         }
598       }
599     {
600       \bool_if:NTF \l__siunitx_table_align_before_bool
601       {
602         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
603         {
604           \box_wd:N \l__siunitx_table_integer_box
605         + \box_wd:N \l__siunitx_table_tmp_box
606         }
607         {
608           \__siunitx_table_fil:
609           \hbox_unpack:N \l__siunitx_table_integer_box
610           \siunitx_print:nn { number } {#4#5}
611         }
612       }
613     {
614       \dim_set:Nn \l__siunitx_table_tmp_dim
615       { \box_wd:N \l__siunitx_table_integer_box }
616       \hbox_set:Nn \l__siunitx_table_integer_box
617       {
618         \hbox_unpack:N \l__siunitx_table_integer_box
619         \siunitx_print:nn { number } {#4#5}
620       }
621       \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
622       {
623         \box_wd:N \l__siunitx_table_before_box
624       + \box_wd:N \l__siunitx_table_tmp_box
625       + \l__siunitx_table_tmp_dim
626       - \box_wd:N \l__siunitx_table_integer_box
627     }

```

```

628         {
629             \_siunitx_table_fil:
630             \hbox_unpack:N \l__siunitx_table_before_box
631         }
632     }
633 }
634 \_siunitx_table_print_format_auxiii:w #3 \q_mark #6 \q_stop
635 }

```

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.

```

636 \cs_new_protected:Npn \_siunitx_table_print_format_auxiii:w
637 #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
638 {
639     \_siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
640     \_siunitx_table_print_format_box:Nn \l__siunitx_table_decimal_box {#4#5}
641     \dim_set:Nn \l__siunitx_table_carry_dim
642     {
643         \box_wd:N \l__siunitx_table_tmp_box
644         - \box_wd:N \l__siunitx_table_decimal_box
645     }
646     \_siunitx_table_print_format_auxiv:w #3 \q_mark #6 \q_stop
647 }

```

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.

```

648 \cs_new_protected:Npn \_siunitx_table_print_format_auxiv:w
649 #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
650 {
651     \tl_if_blank:nTF {#1}
652     { \_siunitx_table_print_format_auxv:w }
653     { \_siunitx_table_print_format_auxvi:w }
654     #1#2 \q_mark #3#4 \q_stop
655 }
656 \cs_new_protected:Npn \_siunitx_table_print_format_auxv:w
657 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
658 #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
659 { \_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.

```

660 \cs_new_protected:Npn \_siunitx_table_print_format_auxvi:w
661 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
662 #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
663 {
664     \_siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2#3 }
665     \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
666     \_siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #5#6#7 }
667     \_siunitx_table_print_format_after:N \l__siunitx_table_align_uncertainty_bool
668     \_siunitx_table_print_format_auxvii:w #4 \q_mark #8 \q_stop
669 }

```

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.

```

670 \cs_new_protected:Npn \__siunitx_table_print_format_auxvii:w
671   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
672   {
673     \tl_if_blank:nF {#2}
674     {
675       \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2 }
676       \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
677       \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #3#4 }
678       \__siunitx_table_print_format_after:N \l__siunitx_table_align_exponent_bool
679     }
680   }

```

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

```

681 \cs_new_protected:Npn \__siunitx_table_print_format_box:Nn #1#2
682   {
683     \hbox_set:Nn #1
684     {
685       \tl_if_blank:nF {#2}
686       { \siunitx_print:nn { number } {#2} }
687     }
688   }

```

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

```

689 \cs_new_protected:Npn \__siunitx_table_print_format_after:N #1
690   {
691     \bool_if:NTF #1
692     {
693       \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
694       {
695         \box_wd:N \l__siunitx_table_decimal_box
696         + \l__siunitx_table_carry_dim
697         + \box_wd:N \l__siunitx_table_tmp_box
698       }
699       {
700         \hbox_unpack:N \l__siunitx_table_decimal_box
701         \__siunitx_table_fil:
702         \hbox_unpack:N \l__siunitx_table_tmp_box
703       }
704       \dim_set:Nn \l__siunitx_table_carry_dim
705       {
706         \l__siunitx_table_tmp_dim
707         - \box_wd:N \l__siunitx_table_tmp_box
708       }
709     }
710     {
711       \hbox_set:Nn \l__siunitx_table_decimal_box
712       {
713         \hbox_unpack:N \l__siunitx_table_decimal_box
714         \hbox_unpack:N \l__siunitx_table_tmp_box
715       }
716       \dim_add:Nn \l__siunitx_table_carry_dim

```

```

717     {
718         \l__siunitx_table_tmp_dim
719         - \box_wd:N \l__siunitx_table_tmp_box
720     }
721 }
722 }

```

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

```

723 \cs_new_protected:Npn \__siunitx_table_print_none:nnn #1#2#3
724 {
725     \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
726     {
727         #1
728         \siunitx_number_format:nN {#2} \l__siunitx_table_tmp_tl
729         \siunitx_print:nV { number } \l__siunitx_table_tmp_tl
730         #3
731     }
732 }

```

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

2.9 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.

```

733 \keys_set:nn { siunitx }
734 {
735     table-align-comparator = true ,
736     table-align-exponent   = true ,
737     table-align-text-after  = true ,
738     table-align-text-before = true ,
739     table-align-uncertainty = true ,
740     table-alignment         = center ,
741     table-auto-round        = false ,
742     table-column-width      = Opt ,
743     table-fixed-width       = false ,
744     table-format            = 2.2 ,
745     table-number-alignment  = center ,
746     table-text-alignment    = center ,

```

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

```

747     table-alignment-mode    = marker
748 }
749 \end{package}

```

Part X

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

<code>\siunitx_unit_format:nN</code>
<code>\siunitx_unit_format:xN</code>

`\siunitx_unit_format:nN` $\langle\{units\}\rangle$ $\langle tl\ var\rangle$

This function converts the input $\langle units\rangle$ into a processed $\langle tl\ var\rangle$ which can then be inserted in math mode to typeset the material. Where the $\langle units\rangle$ are given in symbolic form, described elsewhere, this formatting process takes place in two stages: the $\langle units\rangle$ are parsed into a structured form before the generation of the appropriate output form based on the active settings. When the $\langle units\rangle$ 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_extract_prefixes:nnN</code>	<code>\siunitx_unit_format_extract_prefixes: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_extract_prefixes:nnN { \kilo \metre \per \second }
\l_tmpa_tl \l_tmpa_fp
```

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

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

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

<code>\siunitx_unit_format_combine_exponent:nnN</code>	<code>\siunitx_unit_format_combine_exponent:nnN {<units>} <exponent> <tl var></code>
--	--

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

For example,

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

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

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

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

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

For example,

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

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

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

2 Defining symbolic units

<code>\siunitx_declare_prefix:Nnn</code>	<code>\siunitx_declare_prefix:Nnn <prefix> {<power>} {<symbol>}</code>
<code>\siunitx_declare_prefix:Nnx</code>	

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

<code>\siunitx_declare_prefix:Nn</code>	<code>\siunitx_declare_prefix:Nn <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>	<code>\siunitx_declare_unit:Nnn <unit> {<meaning>} {<options>}</code>
<code>\siunitx_declare_unit:Nnn</code>	
<code>\siunitx_declare_unit:Nxn</code>	

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

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

`\l_siunitx_unit_symbolic_seq`

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

`\l_siunitx_unit_seq`

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

3 Per-unit options

`\siunitx_unit_options_apply:n` `\siunitx_unit_options_apply:n` $\langle unit(s) \rangle$

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

4 Units in (PDF) strings

`\siunitx_unit_pdfstring_context:` `\group_begin:`
 `\siunitx_unit_pdfstring_context:`
 $\langle Expansion\ context \rangle$ $\langle units \rangle$
 `\group_end:`

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

5 Pre-defined symbolic unit components

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

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

<code>\kilogram</code>	The base units as defined in 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> .
--------------------	--

<code>\yocto</code>	Prefixes, all of which are integer powers of 10: the powers are stored internally by the module and can be used for conversion from prefixes to their numerical equivalent. These prefixes are documented in Section 3.1 of the SI Brochure.
<code>\zepto</code>	
<code>\atto</code>	
<code>\femto</code>	
<code>\pico</code>	
<code>\nano</code>	
<code>\micro</code>	
<code>\milli</code>	
<code>\centi</code>	
<code>\deci</code>	
<code>\deca</code>	
<code>\deka</code>	
<code>\hecto</code>	
<code>\kilo</code>	
<code>\mega</code>	
<code>\giga</code>	
<code>\tera</code>	
<code>\peta</code>	
<code>\exa</code>	
<code>\zetta</code>	
<code>\yotta</code>	

<hr/> <code>\becquerel</code> <code>\degreeCelsius</code> <code>\coulomb</code> <code>\farad</code> <code>\gray</code> <code>\hertz</code> <code>\henry</code> <code>\joule</code> <code>\katal</code> <code>\lumen</code> <code>\lux</code> <code>\newton</code> <code>\ohm</code> <code>\pascal</code> <code>\radian</code> <code>\siemens</code> <code>\sievert</code> <code>\steradian</code> <code>\tesla</code> <code>\volt</code> <code>\watt</code> <code>\weber</code> <hr/>	<p>The defined SI units with defined names and symbols, as given in Table 4 of the SI Brochure. Notice that the names of the units are lower case with the exception of <code>\degreeCelsius</code>, and that this unit name includes “degree”.</p>
<hr/> <code>\astronomicalunit</code> <code>\bel</code> <code>\dalton</code> <code>\day</code> <code>\decibel</code> <code>\electronvolt</code> <code>\hectare</code> <code>\hour</code> <code>\litre</code> <code>\liter</code> <code>\neper</code> <code>\minute</code> <code>\tonne</code> <hr/>	<p>Units accepted for use with the SI: here <code>\minute</code> is a unit of time not of plane angle. These units are taken from Table 8 of the SI Brochure.</p> <p>For the unit <code>\litre</code>, both <code>l</code> and <code>L</code> are listed as acceptable symbols: the latter is the standard setting of the module. The alternative spelling <code>\liter</code> is also given for this unit for US users (as with <code>\metre</code>, the official spelling is “re”).</p>
<hr/> <code>\arcminute</code> <code>\arcsecond</code> <code>\degree</code> <hr/>	<p>Units for plane angles accepted for use with the SI: to avoid a clash with units for time, here <code>\arcminute</code> and <code>\arcsecond</code> are used in place of <code>\minute</code> and <code>\second</code>. These units are taken from Table 8 of the SI Brochure.</p>
<hr/> <code>\percent</code> <hr/>	<p>The mathematical concept of percent, usable with the SI as detailed in Section 5.4.7 of the SI Brochure.</p>
<hr/> <code>\square</code> <code>\cubic</code> <hr/>	<p><code>\square</code> $\langle prefix \rangle \langle unit \rangle$ <code>\cubic</code> $\langle prefix \rangle \langle unit \rangle$</p> <p>Pre-defined unit powers which apply to the next $\langle prefix \rangle / \langle unit \rangle$ combination.</p>

<hr/>	<code>\squared</code>	<code><prefix> <unit> \squared</code>
<hr/>	<code>\cubed</code>	<code><prefix> <unit> \cubed</code>

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

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

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

<hr/>	<code>\cancel</code>	<code>\cancel <prefix> <unit> <power></code>
-------	----------------------	--

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

<hr/>	<code>\highlight</code>	<code>\highlight {<color>} <prefix> <unit> <power></code>
-------	-------------------------	---

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

<hr/>	<code>\of</code>	<code><prefix> <unit> <power> \of {<qualifier>}</code>
-------	------------------	--

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

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

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

5.1 Key-value options

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

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

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

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

qualifier-mode

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

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

qualifier-phrase

`qualifier-phrase = \langle phrase \rangle`

Defines the \langle phrase \rangle used when `qualifier-mode` is set to `phrase`.

sticky-per

`sticky-per = true|false`

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

6 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 $\rangle$ 
```

6.1 Initial set up

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

Variants not provided by `expl3`.

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

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

Scratch space.

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

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

```
\c_siunitx_unit_math_subscript_tl
```

Useful tokens with awkward category codes.

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

(End definition for `\c_siunitx_unit_math_subscript_tl`.)

```
\l_siunitx_unit_parsing_bool
```

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

```
9 \bool_new:N \l_siunitx_unit_parsing_bool
```

(End definition for `\l__siunitx_unit_parsing_bool`.)

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

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

(End definition for `\l__siunitx_unit_test_bool`.)

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

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

(End definition for `__siunitx_unit_if_symbolic:nTF`.)

6.2 Defining symbolic unit

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

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

```
21 \seq_new:N \l_siunitx_unit_symbolic_seq
```

(End definition for `\l_siunitx_unit_symbolic_seq`. This variable is documented on page 133.)

`\l_siunitx_unit_seq` A second list featuring only the units themselves.

```
22 \seq_new:N \l_siunitx_unit_seq
```

(End definition for `\l_siunitx_unit_seq`. This variable is documented on page 133.)

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

mechanism is set up in a two-part fashion allowing for flexibility at the slight cost of additional functions.

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

```

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

```

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

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

```

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

```

(End definition for `\siunitx_declare_power:NNn`. This function is documented on page [132](#).)

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

`\l_siunitx_unit_prefixes_forward_prop`
`\l_siunitx_unit_prefixes_reverse_prop`

```

49 \cs_new_protected:Npn \siunitx_declare_prefix:Nn #1#2
50   {
51     \__siunitx_unit_set_symbolic:Nnn #1
52     {#2}
53     { \__siunitx_unit_parse_prefix:Nn #1 {#2} }
54   }
55 \cs_new_protected:Npn \siunitx_declare_prefix:Nnn #1#2#3

```



```

56 {
57   \siunitx_declare_prefix:Nn #1 {#3}
58   \prop_put:Nnn \l__siunitx_unit_prefixes_forward_prop {#3} {#2}
59   \prop_put:Nnn \l__siunitx_unit_prefixes_reverse_prop {#2} {#3}
60 }
61 \cs_generate_variant:Nn \siunitx_declare_prefix:Nnn { Nnx }
62 \prop_new:N \l__siunitx_unit_prefixes_forward_prop
63 \prop_new:N \l__siunitx_unit_prefixes_reverse_prop

```

(End definition for `\siunitx_declare_prefix:Nn` and others. These functions are documented on page 132.)

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

```

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

```

(End definition for `\siunitx_declare_qualifier:Nn`. This function is documented on page 132.)

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

```

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

```

(End definition for `\siunitx_declare_unit:Nn` and `\siunitx_declare_unit:Nnn`. These functions are documented on page 132.)

6.3 Applying unit options

```

\l__siunitx_unit_options_bool
88 \bool_new:N \l__siunitx_unit_options_bool

```

(End definition for `\l__siunitx_unit_options_bool`.)

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

```

89 \cs_new_protected:Npn \siunitx_unit_options_apply:n #1
90 {
91   \bool_if:NF \l__siunitx_unit_options_bool
92   {
93     \tl_if_single_token:nT {#1}
94     {
95       \tl_if_exist:cT { l__siunitx_unit_options_ \token_to_str:N #1 _tl }
96       {
97         \keys_set:nv { siunitx }
98         { l__siunitx_unit_options_ \token_to_str:N #1 _tl }
99       }
100     }
101   }
102   \bool_set_true:N \l__siunitx_unit_options_bool
103 }

```

(End definition for `\siunitx_unit_options_apply:n`. This function is documented on page 133.)

6.4 Non-standard symbolic units

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

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

```

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

```

(End definition for `\per`. This function is documented on page 136.)

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

```

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

```

(End definition for `\cancel` and `\highlight`. These functions are documented on page 136.)

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

```

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

```

(End definition for `\of`. This function is documented on page 136.)

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

`\tothe`

```

116 \__siunitx_unit_set_symbolic:Npnn \raiseto #1
117   { \__siunitx_unit_literal_power:nn {#1} }
118   { \__siunitx_unit_parse_power:nnN { \raiseto {#1} } {#1} \c_true_bool }
119 \__siunitx_unit_set_symbolic:Npnn \tothe #1
120   { ~ {#1} }
121   { \__siunitx_unit_parse_power:nnN { \tothe {#1} } {#1} \c_false_bool }

```

(End definition for `\raiseto` and `\tothe`. These functions are documented on page 136.)

6.5 Main formatting routine

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

`\l_siunitx_unit_font_tl`
`\l__siunitx_unit_product_tl`
`\l__siunitx_unit_mass_kilogram_bool`

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

```

122 \keys_define:nn { siunitx }
123   {
124     extract-mass-in-kilograms .bool_set:N =
125       \l__siunitx_unit_mass_kilogram_bool ,
126     font-command .tl_set:N =
127       \l_siunitx_unit_font_tl ,
128     inter-unit-product .tl_set:N =
129       \l__siunitx_unit_product_tl
130   }

```

(End definition for `\l_siunitx_unit_font_tl`, `\l__siunitx_unit_product_tl`, and `\l__siunitx_unit_mass_kilogram_bool`. This variable is documented on page 132.)

`\l__siunitx_unit_formatted_tl`

A token list for the final formatted result: may or may not be generated by the parser, depending on the nature of the input.

```

131 \tl_new:N \l__siunitx_unit_formatted_tl

```

(End definition for `\l__siunitx_unit_formatted_tl`.)

`\siunitx_unit_format:nN`
`\siunitx_unit_format_extract_prefixes:nN`
`\siunitx_unit_format_combine_exponent:nN`
`\siunitx_unit_format_multiply:nN`
`\siunitx_unit_format_multiply_extract_prefixes:nN`
`\siunitx_unit_format_multiply_combine_exponent:nnN`
`__siunitx_unit_format:nN`
`__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.

```

132 \cs_new_protected:Npn \siunitx_unit_format:nN #1#2
133 {
134   \bool_set_false:N \l__siunitx_unit_prefix_exp_bool
135   \fp_zero:N \l__siunitx_unit_combine_exp_fp
136   \fp_set:Nn \l__siunitx_unit_multiple_fp { \c_one_fp }
137   \__siunitx_unit_format:nNN {#1} #2 \l__siunitx_unit_tmp_fp
138 }
139 \cs_new_protected:Npn \siunitx_unit_format_extract_prefixes:nNN #1#2#3
140 {
141   \bool_set_true:N \l__siunitx_unit_prefix_exp_bool
142   \fp_zero:N \l__siunitx_unit_combine_exp_fp
143   \fp_set:Nn \l__siunitx_unit_multiple_fp { \c_one_fp }
144   \__siunitx_unit_format:nNN {#1} #2 #3
145 }
146 \cs_new_protected:Npn \siunitx_unit_format_combine_exponent:nnN #1#2#3
147 {
148   \bool_set_false:N \l__siunitx_unit_prefix_exp_bool
149   \fp_set:Nn \l__siunitx_unit_combine_exp_fp {#2}
150   \fp_set:Nn \l__siunitx_unit_multiple_fp { \c_one_fp }
151   \__siunitx_unit_format:nNN {#1} #3 \l__siunitx_unit_tmp_fp
152 }
153 \cs_new_protected:Npn \siunitx_unit_format_multiply:nnN #1#2#3
154 {
155   \bool_set_false:N \l__siunitx_unit_prefix_exp_bool
156   \fp_zero:N \l__siunitx_unit_combine_exp_fp
157   \fp_set:Nn \l__siunitx_unit_multiple_fp {#2}
158   \__siunitx_unit_format:nNN {#1} #3 \l__siunitx_unit_tmp_fp
159 }
160 \cs_new_protected:Npn \siunitx_unit_format_multiply_extract_prefixes:nnNN
161   #1#2#3#4
162 {
163   \bool_set_true:N \l__siunitx_unit_prefix_exp_bool
164   \fp_zero:N \l__siunitx_unit_combine_exp_fp
165   \fp_set:Nn \l__siunitx_unit_multiple_fp {#2}
166   \__siunitx_unit_format:nNN {#1} #3 #4
167 }
168 \cs_new_protected:Npn \siunitx_unit_format_multiply_combine_exponent:nnnN
169   #1#2#3#4
170 {
171   \bool_set_false:N \l__siunitx_unit_prefix_exp_bool
172   \fp_set:Nn \l__siunitx_unit_combine_exp_fp {#3}
173   \fp_set:Nn \l__siunitx_unit_multiple_fp {#2}
174   \__siunitx_unit_format:nNN {#1} #4 \l__siunitx_unit_tmp_fp
175 }
176 \cs_new_protected:Npn \__siunitx_unit_format:nNN #1#2#3
177 {
178   \group_begin:
179   \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
180     { \cs_set_eq:Nc ##1 { __siunitx_unit_token_to_str:N ##1 :w } }
181   \tl_clear:N \l__siunitx_unit_formatted_tl

```

```

182 \fp_zero:N \l__siunitx_unit_prefix_fp
183 \bool_if:NTF \l__siunitx_unit_parse_bool
184 {
185   \__siunitx_unit_if_symbolic:nTF {#1}
186   {
187     \__siunitx_unit_parse:n {#1}
188     \prop_if_empty:NF \l__siunitx_unit_parsed_prop
189     { \__siunitx_unit_format_parsed: }
190   }
191   {
192     \bool_if:NTF \l__siunitx_unit_forbid_literal_bool
193     { \msg_error:nnn { siunitx } { unit / literal } {#1} }
194     { \__siunitx_unit_format_literal:n {#1} }
195   }
196 }
197 { \__siunitx_unit_format_literal:n {#1} }
198 \cs_set_protected:Npx \__siunitx_unit_format_aux:
199 {
200   \tl_set:Nn \exp_not:N #2
201   { \exp_not:V \l__siunitx_unit_formatted_tl }
202   \fp_set:Nn \exp_not:N #3
203   { \fp_use:N \l__siunitx_unit_prefix_fp }
204 }
205 \exp_after:wN \group_end:
206 \__siunitx_unit_format_aux:
207 }
208 \cs_new_protected:Npn \__siunitx_unit_format_aux: { }

```

(End definition for `\siunitx_unit_format:nN` and others. These functions are documented on page 130.)

6.6 Formatting literal units

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

`__siunitx_unit_literal_power:nn`

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

```

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

```

(End definition for `__siunitx_unit_literal_power:nn`.)

`__siunitx_unit_literal_special:nN`

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

```

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

```

(End definition for `__siunitx_unit_literal_special:nN`.)

`__siunitx_unit_format_literal:n`

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

```

\__siunitx_unit_format_literal_tilde:
\__siunitx_unit_format_literal_subscript:
\__siunitx_unit_format_literal_superscript:
\__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

```

and superscript tokens, appropriate definitions are provided at this stage. Those have to be expandable macros rather than implicit character tokens.

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

```

211 \group_begin:
212   \char_set_catcode_active:n { '\~ }
213   \cs_new_protected:Npx \__siunitx_unit_format_literal:n #1
214   {
215     \group_begin:
216     \exp_not:n { \bool_set_false:N \l__siunitx_unit_parsing_bool }
217     \tl_set:Nn \exp_not:N \l__siunitx_unit_tmp_tl {#1}
218     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
219     { \token_to_str:N ^ } { ^ }
220     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
221     { \token_to_str:N _ } { \c__siunitx_unit_math_subscript_tl }
222     \char_set_active_eq:NN ^
223     \exp_not:N \__siunitx_unit_format_literal_superscript:
224     \char_set_active_eq:NN _
225     \exp_not:N \__siunitx_unit_format_literal_subscript:
226     \char_set_active_eq:NN \exp_not:N ~
227     \exp_not:N \__siunitx_unit_format_literal_tilde:
228     \exp_not:n
229     {
230       \protected@edef \l__siunitx_unit_tmp_tl
231       { \l__siunitx_unit_tmp_tl }
232       \tl_clear:N \l__siunitx_unit_formatted_tl
233       \tl_if_empty:NF \l__siunitx_unit_tmp_tl
234       {
235         \exp_after:wN \__siunitx_unit_format_literal_auxi:w
236         \l__siunitx_unit_tmp_tl .
237         \q_recursion_tail . \q_recursion_stop
238       }
239       \exp_args:NNNV \group_end:
240       \tl_set:Nn \l__siunitx_unit_formatted_tl
241       \l__siunitx_unit_formatted_tl
242     }
243   }
244 \group_end:
245 \cs_new:Npx \__siunitx_unit_format_literal_subscript: { \c__siunitx_unit_math_subscript_tl }
246 \cs_new:Npn \__siunitx_unit_format_literal_superscript: { ^ }
247 \cs_new:Npn \__siunitx_unit_format_literal_tilde: { . }

```

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

```

248 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxi:w #1 .
249 {
250   \quark_if_recursion_tail_stop:n {#1}
251   \__siunitx_unit_format_literal_auxii:n {#1}
252   \tl_set_eq:NN \l__siunitx_unit_separator_tl \l__siunitx_unit_product_tl
253   \__siunitx_unit_format_literal_auxi:w
254 }
255 \cs_set_protected:Npn \__siunitx_unit_format_literal_auxii:n #1

```

```

256 {
257   \_siunitx_unit_format_literal_auxiii:w
258   #1 / \q_recursion_tail / \q_recursion_stop
259 }
260 \cs_new_protected:Npn \_siunitx_unit_format_literal_auxiii:w #1 /
261 {
262   \quark_if_recursion_tail_stop:n {#1}
263   \_siunitx_unit_format_literal_auxiv:w #1 ^ ^ \q_stop
264   \tl_set:Nn \l_siunitx_unit_separator_tl { / }
265   \_siunitx_unit_format_literal_auxiii:w
266 }

```

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.

```

267 \use:x
268 {
269   \cs_new_protected:Npn \exp_not:N \_siunitx_unit_format_literal_auxiv:w
270   ##1 ^ ##2 ^ ##3 \exp_not:N \q_stop
271   {
272     \exp_not:N \_siunitx_unit_format_literal_auxv:w
273     ##1
274     \c_siunitx_unit_math_subscript_tl
275     \c_siunitx_unit_math_subscript_tl
276     \exp_not:N \q_mark
277     ##2
278     \c_siunitx_unit_math_subscript_tl
279     \c_siunitx_unit_math_subscript_tl
280     \exp_not:N \q_stop
281   }
282   \cs_new_protected:Npn \exp_not:N \_siunitx_unit_format_literal_auxv:w
283   ##1 \c_siunitx_unit_math_subscript_tl
284   ##2 \c_siunitx_unit_math_subscript_tl ##3
285   \exp_not:N \q_mark
286   ##4 \c_siunitx_unit_math_subscript_tl
287   ##5 \c_siunitx_unit_math_subscript_tl ##6
288   \exp_not:N \q_stop
289   {
290     \tl_set:Nx \exp_not:N \l_siunitx_unit_formatted_tl
291     {
292       \exp_not:N \exp_not:N
293       \exp_not:N \l_siunitx_unit_formatted_tl
294       \exp_not:N \tl_if_empty:NF
295       \exp_not:N \l_siunitx_unit_formatted_tl
296       {
297         \exp_not:N \exp_not:N
298         \exp_not:N \l_siunitx_unit_separator_tl
299       }
300       \exp_not:N \tl_if_blank:nF {##1}
301       {
302         \exp_not:N \exp_not:N

```

```

303         \exp_not:N \l_siunitx_unit_font_tl
304         { \exp_not:N \exp_not:n {##1} }
305     }
306     \exp_not:N \tl_if_blank:nF {##4}
307     { ^ { \exp_not:N \exp_not:n {##4} } }
308     \exp_not:N \tl_if_blank:nF {##2##5}
309     {
310         \c__siunitx_unit_math_subscript_tl
311         {
312             \exp_not:N \exp_not:N
313             \exp_not:N \l_siunitx_unit_font_tl
314             { \exp_not:N \exp_not:n {##2##5} }
315         }
316     }
317 }
318 }
319 }
320 \tl_new:N \l__siunitx_unit_separator_tl

```

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

6.7 (PDF) String creation

`\siunitx_unit_pdfstring_context:` A simple function that sets up to make units equal to their text representation.

```

321 \cs_new_protected:Npn \siunitx_unit_pdfstring_context:
322 {
323     \bool_set_false:N \l__siunitx_unit_parsing_bool
324     \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
325     { \cs_set_eq:Nc ##1 { __siunitx_unit_ \token_to_str:N ##1 :w } }
326 }

```

(End definition for `\siunitx_unit_pdfstring_context:.` This function is documented on page 133.)

6.8 Parsing symbolic units

Parsing units takes place by storing information about each unit in a `prop`. As well as the unit itself, there are various other optional data points, for example a prefix or a power. Some of these can come before the unit, others only after. The parser therefore tracks the number of units read and uses the current position to allocate data to individual units.

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

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

- **qualifier-*n*** Any qualifier which applies to the current unit.
- **special-*n*** Any “special effect” to apply to the current unit.
- **command-*l*** The command corresponding to **unit-*n***: needed to track base units; used for **\gram** only.

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

```

327 \keys_define:nn { siunitx }
328   {
329     sticky-per .bool_set:N = \l__siunitx_unit_sticky_per_bool
330   }

```

(End definition for **\l_siunitx_unit_sticky_per_bool**.)

\l_siunitx_unit_parsed_prop 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.

```

331 \prop_new:N \l__siunitx_unit_parsed_prop
332 \bool_new:N \l__siunitx_unit_per_bool
333 \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.

```

334 \cs_new_protected:Npn \__siunitx_unit_parse:n #1
335   {
336     \prop_clear:N \l__siunitx_unit_parsed_prop
337     \bool_set_true:N \l__siunitx_unit_parsing_bool
338     \bool_set_false:N \l__siunitx_unit_per_bool
339     \bool_set_false:N \l__siunitx_unit_test_bool
340     \int_zero:N \l__siunitx_unit_position_int
341     \siunitx_unit_options_apply:n {#1}
342     #1
343     \int_step_inline:nn \l__siunitx_unit_position_int
344       { \__siunitx_unit_parse_finalise:n {##1} }
345     \__siunitx_unit_parse_finalise:
346   }

```

(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.

```

347 \cs_new_protected:Npn \_siunitx_unit_parse_add:nnnn #1#2#3#4
348   {
349     \tl_set:Nx \l__siunitx_unit_tmp_tl { #1 - #2 }
350     \prop_if_in:NVTF \l__siunitx_unit_parsed_prop

```

```

351 \l__siunitx_unit_tmp_tl
352 {
353   \msg_error:nnxx { siunitx } { unit / duplicate-part }
354   { \exp_not:n {#1} } { \token_to_str:N #3 }
355 }
356 {
357   \prop_put:NVn \l__siunitx_unit_parsed_prop
358   \l__siunitx_unit_tmp_tl {#4}
359 }
360 }

```

(End definition for __siunitx_unit_parse_add:nnnn.)

__siunitx_unit_parse_prefix:Nn
 __siunitx_unit_parse_power:nnN
 __siunitx_unit_parse_qualifier:nn
 __siunitx_unit_parse_special:n

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

```

361 \cs_new_protected:Npn \__siunitx_unit_parse_prefix:Nn #1#2
362 {
363   \int_set:Nn \l__siunitx_unit_tmp_int { \l__siunitx_unit_position_int + 1 }
364   \__siunitx_unit_parse_add:nnnn { prefix }
365   { \int_use:N \l__siunitx_unit_tmp_int } {#1} {#2}
366 }
367 \cs_new_protected:Npn \__siunitx_unit_parse_power:nnN #1#2#3
368 {
369   \tl_set:Nx \l__siunitx_unit_tmp_tl
370   { unit- \int_use:N \l__siunitx_unit_position_int }
371   \bool_lazy_or:nnTF
372   {#3}
373   {
374     \prop_if_in_p:NV
375     \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
376   }
377   {
378     \__siunitx_unit_parse_add:nnnn { power }
379     {
380       \int_eval:n
381       { \l__siunitx_unit_position_int \bool_if:NT #3 { + 1 } }
382     }
383     {#1} {#2}
384   }
385   {
386     \msg_error:nnxx { siunitx }
387     { unit / part-before-unit } { power } { \token_to_str:N #1 }
388   }
389 }
390 \cs_new_protected:Npn \__siunitx_unit_parse_qualifier:nn #1#2
391 {
392   \tl_set:Nx \l__siunitx_unit_tmp_tl
393   { unit- \int_use:N \l__siunitx_unit_position_int }
394   \prop_if_in:NVTF \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
395   {
396     \__siunitx_unit_parse_add:nnnn { qualifier }

```

```

397         { \int_use:N \l__siunitx_unit_position_int } {#1} {#2}
398     }
399     {
400         \msg_error:nnnn { siunitx }
401         { unit / part-before-unit } { qualifier } { \token_to_str:N #1 }
402     }
403 }

```

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

```

404 \cs_new_protected:Npn \__siunitx_unit_parse_special:n #1
405 {
406     \__siunitx_unit_parse_add:nnnn { special }
407     { \int_eval:n { \l__siunitx_unit_position_int + 1 } } }
408     {#1} {#1}
409 }

```

(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.

```

410 \cs_new_protected:Npn \__siunitx_unit_parse_unit:Nn #1#2
411 {
412     \int_incr:N \l__siunitx_unit_position_int
413     \tl_if_eq:nnT {#1} { \gram }
414     {
415         \__siunitx_unit_parse_add:nnnn { command }
416         { \int_use:N \l__siunitx_unit_position_int }
417         {#1} {#1}
418     }
419     \__siunitx_unit_parse_add:nnnn { unit }
420     { \int_use:N \l__siunitx_unit_position_int }
421     {#1} {#2}
422     \bool_if:NT \l__siunitx_unit_per_bool
423     {
424         \__siunitx_unit_parse_add:nnnn { per }
425         { \int_use:N \l__siunitx_unit_position_int }
426         { \per } { true }
427     }
428 }

```

(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.

```

429 \cs_new_protected:Npn \__siunitx_unit_parse_per:
430 {
431     \bool_if:NTF \l__siunitx_unit_sticky_per_bool
432     {

```

```

433     \bool_set_true:N \l__siunitx_unit_per_bool
434     \cs_set_protected:Npn \per
435       { \msg_error:nn { siunitx } { unit / duplicate-sticky-per } }
436   }
437   {
438     \__siunitx_unit_parse_add:nnnn
439     { per } { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
440     { \per } { true }
441   }
442 }

```

(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.

```

443 \cs_new_protected:Npn \__siunitx_unit_parse_finalise:n #1
444 {
445   \tl_set:Nx \l__siunitx_unit_tmp_tl { per- #1 }
446   \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
447   {
448     \prop_remove:NV \l__siunitx_unit_parsed_prop
449     \l__siunitx_unit_tmp_tl
450     \tl_set:Nx \l__siunitx_unit_tmp_tl { power- #1 }
451     \prop_get:NVNTF
452       \l__siunitx_unit_parsed_prop
453       \l__siunitx_unit_tmp_tl
454       \l__siunitx_unit_part_tl
455     {
456       \tl_set:Nx \l__siunitx_unit_part_tl
457         { \fp_eval:n { \l__siunitx_unit_part_tl * -1 } }
458       \fp_compare:nNnTF \l__siunitx_unit_part_tl = 1
459       {
460         \prop_remove:NV \l__siunitx_unit_parsed_prop
461         \l__siunitx_unit_tmp_tl
462       }
463       {
464         \prop_put:NVV \l__siunitx_unit_parsed_prop
465         \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
466       }
467     }
468     {
469       \prop_put:NVN \l__siunitx_unit_parsed_prop
470       \l__siunitx_unit_tmp_tl { -1 }
471     }
472   }
473 }

```

(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.

```

474 \cs_new_protected:Npn \__siunitx_unit_parse_finalise:

```

```

475 {
476   \clist_map_inline:nn { per , power , prefix }
477   {
478     \tl_set:Nx \l__siunitx_unit_tmp_tl
479     { ##1 - \int_eval:n { \l__siunitx_unit_position_int + 1 } }
480     \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
481     { \msg_error:nnn { siunitx } { unit / dangling-part } { ##1 } }
482   }
483 }

```

(End definition for `__siunitx_unit_parse_finalise:.`)

6.9 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_parse_bool
\l__siunitx_unit_per_symbol_tl
\l__siunitx_unit_qualifier_mode_tl
\l__siunitx_unit_qualifier_phrase_tl
484 \keys_define:nn { siunitx }
485 {
486   bracket-unit-denominator .bool_set:N =
487     \l__siunitx_unit_denominator_bracket_bool ,
488   forbid-literal-units .bool_set:N =
489     \l__siunitx_unit_forbid_literal_bool ,
490   fraction-command .tl_set:N =
491     \l__siunitx_unit_fraction_function_tl ,
492   parse-units .bool_set:N =
493     \l__siunitx_unit_parse_bool ,
494   per-mode .choice: ,
495   per-mode / fraction .code:n =
496   {
497     \bool_set_false:N \l__siunitx_unit_autofrac_bool
498     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
499     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
500     \bool_set_true:N \l__siunitx_unit_two_part_bool
501   } ,
502   per-mode / power .code:n =
503   {
504     \bool_set_false:N \l__siunitx_unit_autofrac_bool
505     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
506     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
507     \bool_set_false:N \l__siunitx_unit_two_part_bool
508   } ,
509   per-mode / power-positive-first .code:n =
510   {
511     \bool_set_false:N \l__siunitx_unit_autofrac_bool
512     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
513     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
514     \bool_set_true:N \l__siunitx_unit_two_part_bool
515   } ,
516   per-mode / repeated-symbol .code:n =
517   {
518     \bool_set_false:N \l__siunitx_unit_autofrac_bool
519     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
520     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
521     \bool_set_false:N \l__siunitx_unit_two_part_bool
522   } ,

```

```

523   per-mode / symbol .code:n =
524   {
525     \bool_set_false:N \l__siunitx_unit_autofrac_bool
526     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
527     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
528     \bool_set_true:N \l__siunitx_unit_two_part_bool
529   } ,
530   per-mode / symbol-or-fraction .code:n =
531   {
532     \bool_set_true:N \l__siunitx_unit_autofrac_bool
533     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
534     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
535     \bool_set_true:N \l__siunitx_unit_two_part_bool
536   } ,
537   per-symbol .tl_set:N =
538     \l__siunitx_unit_per_symbol_tl ,
539   qualifier-mode .choices:nn =
540     { bracket , combine , phrase , subscript }
541     { \tl_set_eq:NN \l__siunitx_unit_qualifier_mode_tl \l_keys_choice_tl } ,
542   qualifier-phrase .tl_set:N =
543     \l__siunitx_unit_qualifier_phrase_tl
544 }

```

(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.

```
545 \bool_new:N \l__siunitx_unit_bracket_bool
```

(End definition for \l__siunitx_unit_bracket_bool.)

\l__siunitx_unit_bracket_open_tl Abstracted out but currently purely internal.

```

546 \tl_new:N \l__siunitx_unit_bracket_open_tl
547 \tl_new:N \l__siunitx_unit_bracket_close_tl
548 \tl_set:Nn \l__siunitx_unit_bracket_open_tl { ( }
549 \tl_set:Nn \l__siunitx_unit_bracket_close_tl { ) }

```

(End definition for \l__siunitx_unit_bracket_open_tl and \l__siunitx_unit_bracket_close_tl.)

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

```
550 \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.

```

551 \bool_new:N \l__siunitx_unit_autofrac_bool
552 \bool_new:N \l__siunitx_unit_per_symbol_bool
553 \bool_new:N \l__siunitx_unit_powers_positive_bool
554 \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).

```
555 \bool_new:N \l__siunitx_unit_numerator_bool
```

(End definition for \l__siunitx_unit_numerator_bool.)

\l__siunitx_unit_qualifier_mode_tl For storing the text of options which are best handled by picking function names.

556 \tl_new:N \l__siunitx_unit_qualifier_mode_tl

(End definition for \l__siunitx_unit_qualifier_mode_tl.)

\l__siunitx_unit_combine_exp_fp For combining an exponent with the first unit.

557 \fp_new:N \l__siunitx_unit_combine_exp_fp

(End definition for \l__siunitx_unit_combine_exp_fp.)

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

558 \bool_new:N \l__siunitx_unit_prefix_exp_bool

(End definition for \l__siunitx_unit_prefix_exp_bool.)

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

559 \fp_new:N \l__siunitx_unit_prefix_fp

(End definition for \l__siunitx_unit_prefix_fp.)

\l__siunitx_unit_multiple_fp For multiplying units.

560 \fp_new:N \l__siunitx_unit_multiple_fp

(End definition for \l__siunitx_unit_multiple_fp.)

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

561 \tl_new:N \l__siunitx_unit_current_tl

562 \tl_new:N \l__siunitx_unit_part_tl

(End definition for \l__siunitx_unit_current_tl and \l__siunitx_unit_part_tl.)

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

563 \tl_new:N \l__siunitx_unit_denominator_tl

(End definition for \l__siunitx_unit_denominator_tl.)

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

564 \int_new:N \l__siunitx_unit_total_int

(End definition for \l__siunitx_unit_total_int.)

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

```

565 \cs_new_protected:Npn \_siunitx_unit_format_parsed:
566 {
567   \int_set_eq:NN \l__siunitx_unit_total_int \l__siunitx_unit_position_int
568   \tl_clear:N \l__siunitx_unit_denominator_tl
569   \tl_clear:N \l__siunitx_unit_formatted_tl
570   \fp_zero:N \l__siunitx_unit_prefix_fp
571   \int_zero:N \l__siunitx_unit_position_int
572   \fp_compare:nNnF \l__siunitx_unit_combine_exp_fp = \c_zero_fp
573     { \_siunitx_unit_format_combine_exp: }
574   \fp_compare:nNnF \l__siunitx_unit_multiple_fp = \c_one_fp
575     { \_siunitx_unit_format_multiply: }
576   \bool_lazy_and:nnT
577     { \l__siunitx_unit_prefix_exp_bool }
578     { \l__siunitx_unit_mass_kilogram_bool }
579     { \_siunitx_unit_format_mass_to_kilogram: }
580   \int_do_while:nNnn
581     \l__siunitx_unit_position_int < \l__siunitx_unit_total_int
582     {
583       \bool_set_false:N \l__siunitx_unit_bracket_bool
584       \tl_clear:N \l__siunitx_unit_current_tl
585       \bool_set_false:N \l__siunitx_unit_font_bool
586       \bool_set_true:N \l__siunitx_unit_numerator_bool
587       \int_incr:N \l__siunitx_unit_position_int
588       \clist_map_inline:nn { prefix , unit , qualifier , power , special }
589         { \_siunitx_unit_format_parsed_aux:n {##1} }
590       \_siunitx_unit_format_output:
591     }
592   \_siunitx_unit_format_finalise:
593 }
594 \cs_new_protected:Npn \_siunitx_unit_format_parsed_aux:n #1
595 {
596   \tl_set:Nx \l__siunitx_unit_tmp_tl
597     { #1 - \int_use:N \l__siunitx_unit_position_int }
598   \prop_get:NVNT \l__siunitx_unit_parsed_prop
599     \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
600   { \use:c { \_siunitx_unit_format_ #1 : } }
601 }

```

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

`_siunitx_unit_format_combine_exp:` To combine an exponent into the first prefix, we first adjust for any power, then deal with any existing prefix, before looking up the final result.

```

602 \cs_new_protected:Npn \_siunitx_unit_format_combine_exp:
603 {
604   \prop_get:NnNF \l__siunitx_unit_parsed_prop { power-1 } \l__siunitx_unit_tmp_tl
605     { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
606   \fp_set:Nn \l__siunitx_unit_tmp_fp
607     { \l__siunitx_unit_combine_exp_fp / \l__siunitx_unit_tmp_tl }
608   \prop_get:NnNTF \l__siunitx_unit_parsed_prop { prefix-1 } \l__siunitx_unit_tmp_tl
609     {
610       \prop_get:NVNF \l__siunitx_unit_prefixes_forward_prop
611         \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl

```



```

612     {
613         \prop_get:NnN \l__siunitx_unit_parsed_prop { prefix-1 } \l__siunitx_unit_tmp_tl
614         \msg_error:nnx { siunitx } { unit / non-numeric-exponent }
615         { \l__siunitx_unit_tmp_tl }
616         \tl_set:Nn \l__siunitx_unit_tmp_tl { 0 }
617     }
618 }
619 { \tl_set:Nn \l__siunitx_unit_tmp_tl { 0 } }
620 \tl_set:Nx \l__siunitx_unit_tmp_tl
621 { \fp_eval:n { \l__siunitx_unit_tmp_fp + \l__siunitx_unit_tmp_tl } }
622 \fp_compare:nNnTF \l__siunitx_unit_tmp_tl = \c_zero_fp
623 { \prop_remove:Nn \l__siunitx_unit_parsed_prop { prefix-1 } }
624 {
625     \prop_get:NVNTF \l__siunitx_unit_prefixes_reverse_prop
626     \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
627     { \prop_put:NnV \l__siunitx_unit_parsed_prop { prefix-1 } \l__siunitx_unit_tmp_tl }
628     {
629         \msg_error:nnx { siunitx } { unit / non-convertible-exponent }
630         { \l__siunitx_unit_tmp_tl }
631     }
632 }
633 }

```

(End definition for __siunitx_unit_format_combine_exp:.)

__siunitx_unit_format_multiply: A simple mapping.

```

634 \cs_new_protected:Npn \__siunitx_unit_format_multiply:
635 {
636     \int_step_inline:nn { \prop_count:N \l__siunitx_unit_parsed_prop }
637     {
638         \prop_get:NnNF \l__siunitx_unit_parsed_prop { power- ##1 } \l__siunitx_unit_tmp_tl
639         { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
640         \fp_set:Nn \l__siunitx_unit_tmp_fp
641         { \l__siunitx_unit_tmp_tl * \l__siunitx_unit_multiple_fp }
642         \fp_compare:nNnTF \l__siunitx_unit_tmp_fp = \c_one_fp
643         { \prop_remove:N \l__siunitx_unit_parsed_prop { power- ##1 } }
644         {
645             \prop_put:Nnx \l__siunitx_unit_parsed_prop { power- ##1 }
646             { \fp_use:N \l__siunitx_unit_tmp_fp }
647         }
648     }
649 }

```

(End definition for __siunitx_unit_format_multiply:.)

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

```

650 \cs_new_protected:Npn \__siunitx_unit_format_mass_to_kilogram:
651 {
652     \int_step_inline:nn \l__siunitx_unit_total_int
653     {
654         \prop_if_in:NnT \l__siunitx_unit_parsed_prop { command- ##1 }

```

```

655     {
656       \prop_if_in:NnF \l__siunitx_unit_parsed_prop { prefix- ##1 }
657       {
658         \group_begin:
659         \bool_set_false:N \l__siunitx_unit_parsing_bool
660         \tl_set:Nx \l__siunitx_unit_tmp_tl { \kilo }
661         \exp_args:NNNV \group_end:
662         \tl_set:Nn \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
663         \prop_put:NnV \l__siunitx_unit_parsed_prop { prefix- ##1 }
664         \l__siunitx_unit_tmp_tl
665         \prop_get:NnNF \l__siunitx_unit_parsed_prop { power- ##1 }
666         \l__siunitx_unit_tmp_tl
667         { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
668         \fp_set:Nn \l__siunitx_unit_prefix_fp
669         { \l__siunitx_unit_prefix_fp - 3 * \l__siunitx_unit_tmp_tl }
670       }
671     }
672   }
673 }

```

(End definition for `_siunitx_unit_format_mass_to_kilogram:`.)

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

```

674 \cs_new:Npn \_siunitx_unit_format_bracket:N #1
675 {
676   \bool_if:NTF \l__siunitx_unit_bracket_bool
677   {
678     \exp_not:V \l__siunitx_unit_bracket_open_tl
679     \exp_not:V #1
680     \exp_not:V \l__siunitx_unit_bracket_close_tl
681   }
682   { \exp_not:V #1 }
683 }

```

(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.

```

684 \cs_new_protected:Npn \_siunitx_unit_format_power:
685 {
686   \_siunitx_unit_format_font:
687   \exp_after:wN \_siunitx_unit_format_power_aux:wTF
688   \l__siunitx_unit_part_tl - \q_stop
689   { \_siunitx_unit_format_power_negative: }
690   { \_siunitx_unit_format_power_positive: }
691 }
692 \cs_new:Npn \_siunitx_unit_format_power_aux:wTF #1 - #2 \q_stop
693 { \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 $\neq 1$).

```

694 \cs_new_protected:Npn \__siunitx_unit_format_power_positive:
695 { \__siunitx_unit_format_power_superscript: }

```

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

```

696 \cs_new_protected:Npn \__siunitx_unit_format_power_negative:
697 {
698   \bool_set_false:N \l__siunitx_unit_numerator_bool
699   \bool_if:NTF \l__siunitx_unit_powers_positive_bool
700   {
701     \tl_set:Nx \l__siunitx_unit_part_tl
702     {
703       \exp_after:wN \__siunitx_unit_format_power_negative_aux:w
704       \l__siunitx_unit_part_tl \q_stop
705     }
706     \str_if_eq:VnF \l__siunitx_unit_part_tl { 1 }
707     { \__siunitx_unit_format_power_superscript: }
708   }
709   { \__siunitx_unit_format_power_superscript: }
710 }
711 \cs_new:Npn \__siunitx_unit_format_power_negative_aux:w - #1 \q_stop
712 { \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.

```

713 \cs_new_protected:Npn \__siunitx_unit_format_power_superscript:
714 {
715   \exp_after:wN \__siunitx_unit_format_power_superscript:w
716   \l__siunitx_unit_part_tl . . \q_stop
717 }
718 \cs_new_protected:Npn \__siunitx_unit_format_power_superscript:w #1 . #2 . #3 \q_stop
719 {
720   \tl_if_blank:nTF {#2}
721   {
722     \tl_set:Nx \l__siunitx_unit_current_tl
723     {
724       \__siunitx_unit_format_bracket:N \l__siunitx_unit_current_tl
725       ^ { \exp_not:n {#1} }
726     }
727   }
728   {
729     \tl_set:Nx \l__siunitx_unit_tmp_tl
730     {
731       { }
732       \tl_if_head_eq_charcode:nNTF {#1} -
733       { { - } { \exp_not:o { \use_none:n #1 } } }
734       { { } { \exp_not:n {#1} } }
735       {#2}
736       { }
737       { }
738       { 0 }

```

```

739     }
740     \tl_set:Nx \l__siunitx_unit_current_tl
741     {
742         \__siunitx_unit_format_bracket:N \l__siunitx_unit_current_tl
743         ~ { \siunitx_number_output:N \l__siunitx_unit_tmp_tl }
744     }
745 }
746 \bool_set_false:N \l__siunitx_unit_bracket_bool
747 }

```

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

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

```

748 \cs_new_protected:Npn \__siunitx_unit_format_prefix:
749 {
750     \bool_if:NTF \l__siunitx_unit_prefix_exp_bool
751     { \__siunitx_unit_format_prefix_exp: }
752     { \__siunitx_unit_format_prefix_symbol: }
753 }
754 \cs_new_protected:Npn \__siunitx_unit_format_prefix_exp:
755 {
756     \prop_get:NVNTF \l__siunitx_unit_prefixes_forward_prop
757     \l__siunitx_unit_part_tl \l__siunitx_unit_part_tl
758     {
759         \bool_if:NT \l__siunitx_unit_mass_kilogram_bool
760         {
761             \tl_set:Nx \l__siunitx_unit_tmp_tl
762             { command- \int_use:N \l__siunitx_unit_position_int }
763             \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
764             { \__siunitx_unit_format_prefix_gram: }
765         }
766         \tl_set:Nx \l__siunitx_unit_tmp_tl
767         { power- \int_use:N \l__siunitx_unit_position_int }
768         \prop_get:NVNF \l__siunitx_unit_parsed_prop
769         \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
770         { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
771         \fp_add:Nn \l__siunitx_unit_prefix_fp
772         { \l__siunitx_unit_tmp_tl * \l__siunitx_unit_part_tl }
773     }
774     { \__siunitx_unit_format_prefix_symbol: }
775 }

```

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

```

776 \cs_new_protected:Npn \__siunitx_unit_format_prefix_gram:
777 {
778     \tl_set:Nx \l__siunitx_unit_part_tl
779     { \int_eval:n { \l__siunitx_unit_part_tl - 3 } }
780     \group_begin:
781     \bool_set_false:N \l__siunitx_unit_parsing_bool
782     \tl_set:Nx \l__siunitx_unit_current_tl { \kilo }
783     \exp_args:NNNV \group_end:

```

```

784 \tl_set:Nn \l__siunitx_unit_current_tl \l__siunitx_unit_current_tl
785 }
786 \cs_new_protected:Npn \__siunitx_unit_format_prefix_symbol:
787 { \tl_set_eq:NN \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl }

```

(End definition for __siunitx_unit_format_prefix: and others.)

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

```

\__siunitx_unit_format_qualifier_bracket:
\__siunitx_unit_format_qualifier_combine:
\__siunitx_unit_format_qualifier_phrase:
\__siunitx_unit_format_qualifier_subscript:
788 \cs_new_protected:Npn \__siunitx_unit_format_qualifier:
789 {
790   \use:c
791   {
792     __siunitx_unit_format_qualifier_
793     \l__siunitx_unit_qualifier_mode_tl :
794   }
795   \tl_put_right:NV \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
796 }
797 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_bracket:
798 {
799   \__siunitx_unit_format_font:
800   \tl_set:Nx \l__siunitx_unit_part_tl
801   {
802     \exp_not:V \l__siunitx_unit_bracket_open_tl
803     \exp_not:V \l__siunitx_unit_font_tl
804     { \exp_not:V \l__siunitx_unit_part_tl }
805     \exp_not:V \l__siunitx_unit_bracket_close_tl
806   }
807 }
808 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_combine: { }
809 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_phrase:
810 {
811   \__siunitx_unit_format_font:
812   \tl_set:Nx \l__siunitx_unit_part_tl
813   {
814     \exp_not:V \l__siunitx_unit_qualifier_phrase_tl
815     \exp_not:V \l__siunitx_unit_font_tl
816     { \exp_not:V \l__siunitx_unit_part_tl }
817   }
818 }
819 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_subscript:
820 {
821   \__siunitx_unit_format_font:
822   \tl_set:Nx \l__siunitx_unit_part_tl
823   {
824     \c__siunitx_unit_math_subscript_tl
825     {
826       \exp_not:V \l__siunitx_unit_font_tl
827       { \exp_not:V \l__siunitx_unit_part_tl }
828     }
829   }
830 }

```

(End definition for __siunitx_unit_format_qualifier: and others.)

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

```

831 \cs_new_protected:Npn \_siunitx_unit_format_special:
832 {
833   \tl_set:Nx \l__siunitx_unit_current_tl
834   {
835     \exp_not:V \l__siunitx_unit_part_tl
836     {
837       \bool_if:NTF \l__siunitx_unit_font_bool
838       { \use:n }
839       { \exp_not:V \l_siunitx_unit_font_tl }
840       { \exp_not:V \l__siunitx_unit_current_tl }
841     }
842   }
843   \bool_set_true:N \l__siunitx_unit_font_bool
844 }

```

(End definition for `_siunitx_unit_format_special:.`)

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

```

845 \cs_new_protected:Npn \_siunitx_unit_format_unit:
846 {
847   \tl_put_right:NV
848   \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
849 }

```

(End definition for `_siunitx_unit_format_unit:.`)

`_siunitx_unit_format_output:` The first step here is to make a choice based on whether the current part should be stored as part of the numerator or denominator of a fraction. In all cases, if the switch `\l__siunitx_unit_numerator_bool` is true then life is simple: add the current part to the numerator with a standard separator

```

\_siunitx_unit_format_output_aux:nn
\_siunitx_unit_format_output_aux:nV
\_siunitx_unit_format_output_aux:nv
850 \cs_new_protected:Npn \_siunitx_unit_format_output:
851 {
852   \_siunitx_unit_format_font:
853   \bool_set_false:N \l__siunitx_unit_bracket_bool
854   \use:c
855   {
856     \_siunitx_unit_format_output_
857     \bool_if:NTF \l__siunitx_unit_numerator_bool
858     { aux: }
859     { denominator: }
860   }
861 }
862 \cs_new_protected:Npn \_siunitx_unit_format_output_aux:
863 {
864   \_siunitx_unit_format_output_aux:nV { formatted }
865   \l__siunitx_unit_product_tl
866 }

```

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.

```

867 \cs_new_protected:Npn \__siunitx_unit_format_output_denominator:
868 {
869   \bool_if:NTF \l__siunitx_unit_two_part_bool
870   {
871     \bool_lazy_and:nnT
872     { \l__siunitx_unit_denominator_bracket_bool }
873     { ! \tl_if_empty_p:N \l__siunitx_unit_denominator_tl }
874     { \bool_set_true:N \l__siunitx_unit_bracket_bool }
875     \__siunitx_unit_format_output_aux:nV { denominator }
876     \l__siunitx_unit_product_tl
877   }
878   {
879     \bool_lazy_and:nnT
880     { \l__siunitx_unit_per_symbol_bool }
881     { \tl_if_empty_p:N \l__siunitx_unit_formatted_tl }
882     { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
883     \__siunitx_unit_format_output_aux:nv { formatted }
884     {
885       \l__siunitx_unit_
886       \bool_if:NTF \l__siunitx_unit_per_symbol_bool
887       { per_symbol }
888       { product }
889       _tl
890     }
891   }
892 }
893 \cs_new_protected:Npn \__siunitx_unit_format_output_aux:nn #1#2
894 {
895   \tl_set:cx { \l__siunitx_unit_ #1 _tl }
896   {
897     \exp_not:v { \l__siunitx_unit_ #1 _tl }
898     \tl_if_empty:cF { \l__siunitx_unit_ #1 _tl }
899     { \exp_not:n {#2} }
900     \exp_not:V \l__siunitx_unit_current_tl
901   }
902 }
903 \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.

```

904 \cs_new_protected:Npn \__siunitx_unit_format_font:
905 {
906   \bool_if:NF \l__siunitx_unit_font_bool
907   {
908     \tl_set:Nx \l__siunitx_unit_current_tl
909     {

```

```

910         \exp_not:V \l_siunitx_unit_font_tl
911         { \exp_not:V \l__siunitx_unit_current_tl }
912     }
913     \bool_set_true:N \l__siunitx_unit_font_bool
914 }
915 }

```

(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:
916 \cs_new_protected:Npn \__siunitx_unit_format_finalise:
917 {
918     \tl_if_empty:NF \l__siunitx_unit_denominator_tl
919     {
920         \bool_if:NTF \l__siunitx_unit_powers_positive_bool
921         { \__siunitx_unit_format_finalise_fractional: }
922         { \__siunitx_unit_format_finalise_power: }
923     }
924 }

```

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.

```

925 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fractional:
926 {
927     \tl_if_empty:NT \l__siunitx_unit_formatted_tl
928     { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
929     \bool_if:NTF \l__siunitx_unit_autofrac_bool
930     { \__siunitx_unit_format_finalise_autofrac: }
931     {
932         \bool_if:NTF \l__siunitx_unit_per_symbol_bool
933         { \__siunitx_unit_format_finalise_symbol: }
934         { \__siunitx_unit_format_finalise_fraction: }
935     }
936 }

```

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.

```

937 \cs_new_protected:Npn \__siunitx_unit_format_finalise_autofrac:
938 {
939     \group_begin:
940     \__siunitx_unit_format_finalise_fraction:
941     \exp_args:NNNV \group_end:
942     \tl_set:Nn \l__siunitx_unit_tmp_tl \l__siunitx_unit_formatted_tl
943     \__siunitx_unit_format_finalise_symbol:
944     \tl_set:Nx \l__siunitx_unit_formatted_tl
945     {
946         \mathchoice
947         { \exp_not:V \l__siunitx_unit_tmp_tl }

```



```

948         { \exp_not:V \l__siunitx_unit_formatted_tl }
949         { \exp_not:V \l__siunitx_unit_formatted_tl }
950         { \exp_not:V \l__siunitx_unit_formatted_tl }
951     }
952 }

```

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

```

953 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fraction:
954 {
955     \tl_set:Nx \l__siunitx_unit_formatted_tl
956     {
957         \exp_not:V \l__siunitx_unit_fraction_function_tl
958         { \exp_not:V \l__siunitx_unit_formatted_tl }
959         { \exp_not:V \l__siunitx_unit_denominator_tl }
960     }
961 }
962 \cs_new_protected:Npn \__siunitx_unit_format_finalise_symbol:
963 {
964     \tl_set:Nx \l__siunitx_unit_formatted_tl
965     {
966         \exp_not:V \l__siunitx_unit_formatted_tl
967         \exp_not:V \l__siunitx_unit_per_symbol_tl
968         \__siunitx_unit_format_bracket:N \l__siunitx_unit_denominator_tl
969     }
970 }

```

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.

```

971 \cs_new_protected:Npn \__siunitx_unit_format_finalise_power:
972 {
973     \tl_if_empty:NTF \l__siunitx_unit_formatted_tl
974     {
975         \tl_set_eq:NN
976         \l__siunitx_unit_formatted_tl
977         \l__siunitx_unit_denominator_tl
978     }
979     {
980         \tl_set:Nx \l__siunitx_unit_formatted_tl
981         {
982             \exp_not:V \l__siunitx_unit_formatted_tl
983             \exp_not:V \l__siunitx_unit_product_tl
984             \exp_not:V \l__siunitx_unit_denominator_tl
985         }
986     }
987 }

```

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

6.10 Non-Latin character support

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

```

988 \bool_lazy_or:nnTF
989 { \sys_if_engine luatex_p: }
990 { \sys_if_engine xetex_p: }

```

```

991 {
992   \cs_new:Npn \_siunitx_unit_non_latin:n #1
993     { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
994 }
995 {
996   \cs_new:Npn \_siunitx_unit_non_latin:n #1
997     {
998       \exp_last_unbraced:Nf \_siunitx_unit_non_latin:nnnn
999       { \char_to_utfviii_bytes:n {#1} }
1000    }
1001   \cs_new:Npn \_siunitx_unit_non_latin:nnnn #1#2#3#4
1002     {
1003       \exp_after:wN \exp_after:wN \exp_after:wN
1004       \exp_not:N \char_generate:nn {#1} { 13 }
1005       \exp_after:wN \exp_after:wN \exp_after:wN
1006       \exp_not:N \char_generate:nn {#2} { 13 }
1007     }
1008 }

```

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

6.11 Pre-defined unit components

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

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

```

\metre \meter
\meter 1009 \siunitx_declare_unit:Nn \kilogram { \kilo \gram }
\mole 1010 \siunitx_declare_unit:Nn \metre { m }
\kelvin 1011 \siunitx_declare_unit:Nn \meter { \metre }
\candela 1012 \siunitx_declare_unit:Nn \mole { mol }
\second 1013 \siunitx_declare_unit:Nn \second { s }
\ampere 1014 \siunitx_declare_unit:Nn \ampere { A }
1015 \siunitx_declare_unit:Nn \kelvin { K }
1016 \siunitx_declare_unit:Nn \candela { cd }

```

(End definition for `\kilogram` and others. These functions are documented on page 134.)

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

```

1017 \siunitx_declare_unit:Nn \gram { g }

```

(End definition for `\gram`. This function is documented on page 134.)

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

```

\zepto 1018 \siunitx_declare_prefix:Nnn \yocto { -24 } { y }
\atto 1019 \siunitx_declare_prefix:Nnn \zepto { -21 } { z }
\femto 1020 \siunitx_declare_prefix:Nnn \atto { -18 } { a }
\pico 1021 \siunitx_declare_prefix:Nnn \femto { -15 } { f }
\nano 1022 \siunitx_declare_prefix:Nnn \pico { -12 } { p }
\micro 1023 \siunitx_declare_prefix:Nnn \nano { -9 } { n }
\milli 1024 \siunitx_declare_prefix:Nnn \micro { -6 } { \_siunitx_unit_non_latin:n { "03BC } }
\centi 1025 \siunitx_declare_prefix:Nnn \milli { -3 } { m }
\deci 1026 \siunitx_declare_prefix:Nnn \centi { -2 } { c }
1027 \siunitx_declare_prefix:Nnn \deci { -1 } { d }

```

(End definition for \yocto and others. These functions are documented on page 134.)

```

\deca Now the large ones.
\deka 1028 \siunitx_declare_prefix:Nnn \deka { 1 } { da }
\hecto 1029 \siunitx_declare_prefix:Nnn \deka { 1 } { da }
\kilo 1030 \siunitx_declare_prefix:Nnn \hecto { 2 } { h }
\mega 1031 \siunitx_declare_prefix:Nnn \kilo { 3 } { k }
\giga 1032 \siunitx_declare_prefix:Nnn \mega { 6 } { M }
\tera 1033 \siunitx_declare_prefix:Nnn \giga { 9 } { G }
\peta 1034 \siunitx_declare_prefix:Nnn \tera { 12 } { T }
\exa 1035 \siunitx_declare_prefix:Nnn \peta { 15 } { P }
\zetta 1036 \siunitx_declare_prefix:Nnn \exa { 18 } { E }
\yotta 1037 \siunitx_declare_prefix:Nnn \zetta { 21 } { Z }
1038 \siunitx_declare_prefix:Nnn \yotta { 24 } { Y }

```

(End definition for \deca and others. These functions are documented on page 134.)

```

\becquerel Named derived units: first half of alphabet.
\degreeCelsius 1039 \siunitx_declare_unit:Nn \becquerel { Bq }
\coulomb 1040 \siunitx_declare_unit:Nx \degreeCelsius { \_siunitx_unit_non_latin:n { "00B0 } C }
\farad 1041 \siunitx_declare_unit:Nn \coulomb { C }
\gray 1042 \siunitx_declare_unit:Nn \farad { F }
\hertz 1043 \siunitx_declare_unit:Nn \gray { Gy }
\henry 1044 \siunitx_declare_unit:Nn \hertz { Hz }
\joule 1045 \siunitx_declare_unit:Nn \henry { H }
\katal 1046 \siunitx_declare_unit:Nn \joule { J }
\katal 1047 \siunitx_declare_unit:Nn \katal { kat }
\lumen 1048 \siunitx_declare_unit:Nn \lumen { lm }
\lux 1049 \siunitx_declare_unit:Nn \lux { lx }

```

(End definition for \becquerel and others. These functions are documented on page 135.)

```

\newton Named derived units: second half of alphabet.
\ohm 1050 \siunitx_declare_unit:Nn \newton { N }
\pascal 1051 \siunitx_declare_unit:Nx \ohm { \_siunitx_unit_non_latin:n { "03A9 } }
\radian 1052 \siunitx_declare_unit:Nn \pascal { Pa }
\siemens 1053 \siunitx_declare_unit:Nn \radian { rad }
\sievert 1054 \siunitx_declare_unit:Nn \siemens { S }
\steradian 1055 \siunitx_declare_unit:Nn \sievert { Sv }
\tesla 1056 \siunitx_declare_unit:Nn \steradian { sr }
\volt 1057 \siunitx_declare_unit:Nn \tesla { T }
\watt 1058 \siunitx_declare_unit:Nn \volt { V }
\weber 1059 \siunitx_declare_unit:Nn \watt { W }
1060 \siunitx_declare_unit:Nn \weber { Wb }

```

(End definition for \newton and others. These functions are documented on page 135.)

```

\astronomicalunit Non-SI, but accepted for general use. Once again there are two spellings, here for litre
\bel and with different output in this case.
\dalton 1061 \siunitx_declare_unit:Nn \astronomicalunit { au }
\day 1062 \siunitx_declare_unit:Nn \bel { B }
\decibel 1063 \siunitx_declare_unit:Nn \decibel { \deci \bel }
\electronvolt 1064 \siunitx_declare_unit:Nn \dalton { Da }
\hectare 1065 \siunitx_declare_unit:Nn \day { d }
\hour
\litre
\liter
\minute
\neper
\tonne

```

```

1066 \siunitx_declare_unit:Nn \electronvolt { eV }
1067 \siunitx_declare_unit:Nn \hectare { ha }
1068 \siunitx_declare_unit:Nn \hour { h }
1069 \siunitx_declare_unit:Nn \litre { L }
1070 \siunitx_declare_unit:Nn \liter { \litre }
1071 \siunitx_declare_unit:Nn \minute { min }
1072 \siunitx_declare_unit:Nn \neper { Np }
1073 \siunitx_declare_unit:Nn \tonne { t }

```

(End definition for \astronomicalunit and others. These functions are documented on page 135.)

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

```

1074 \siunitx_declare_unit:Nx \arcminute { \_siunitx_unit_non_latin:n { "02B9 } }
1075 \siunitx_declare_unit:Nx \arcsecond { \_siunitx_unit_non_latin:n { "02BA } }
1076 \siunitx_declare_unit:Nx \degree { \_siunitx_unit_non_latin:n { "00B0 } }

```

(End definition for \arcminute, \arcsecond, and \degree. These functions are documented on page 135.)

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

```

1077 \siunitx_declare_unit:Nx \percent { \cs_to_str:N \% }

```

(End definition for \percent. This function is documented on page 135.)

\square Basic powers.

```

1078 \siunitx_declare_power:NNn \square \squared { 2 }
1079 \siunitx_declare_power:NNn \cubic \cubed { 3 }

```

(End definition for \square and others. These functions are documented on page 135.)

6.12 Messages

```

1080 \msg_new:nnnn { siunitx } { unit / dangling-part }
1081 { Found~#1~part~with~no~unit. }
1082 {
1083   Each~#1~part~must~be~associated~with~a~unit:~a~#1~part~was~found~
1084   but~no~following~unit~was~given.
1085 }
1086 \msg_new:nnnn { siunitx } { unit / duplicate-part }
1087 { Duplicate~#1~part:~#2. }
1088 {
1089   Each~unit~may~have~only~one~#1:\\
1090   the~additional~#1~part~'~#2'~will~be~ignored.
1091 }
1092 \msg_new:nnnn { siunitx } { unit / duplicate-sticky-per }
1093 { Duplicate~\token_to_str:N \per. }
1094 {
1095   When~the~'sticky-per'~option~is~active,~only~one~
1096   \token_to_str:N \per \ may~appear~in~a~unit.
1097 }
1098 \msg_new:nnnn { siunitx } { unit / literal }
1099 { Literal~units~disabled. }
1100 {
1101   You~gave~the~literal~input~'~#1'~
1102   but~literal~unit~output~is~disabled.

```

```

1103 }
1104 \msg_new:nnnn { siunitx } { unit / non-convertible-exponent }
1105 { Exponent~'#1'~cannot~be~converted~into~a~symbolic~prefix. }
1106 {
1107   The~exponent~'#1'~does~not~match~with~any~of~the~symbolic~prefixes~
1108   set~up.
1109 }
1110 \msg_new:nnnn { siunitx } { unit / non-numeric-exponent }
1111 { Prefix~'#1'~does~not~have~a~numerical~value. }
1112 {
1113   The~prefix~'#1'~needs~to~be~combined~with~a~number,~but~it~has~no
1114   numerical~value.
1115 }
1116 \msg_new:nnnn { siunitx } { unit / part-before-unit }
1117 { Found~#1~part~before~first~unit::~#2. }
1118 {
1119   The~#1~part~'#2'~must~follow~after~a~unit::~
1120   it~cannot~appear~before~any~units~and~will~therefore~be~ignored.
1121 }

```

6.13 Standard settings for module options

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

```

1122 \keys_set:nn { siunitx }
1123 {
1124   font-command           = \mathrm    ,
1125   bracket-unit-denominator = true      ,
1126   forbid-literal-units    = false     ,
1127   fraction-command        = \frac     ,
1128   inter-unit-product      = \,        ,
1129   extract-mass-in-kilograms = true     ,
1130   parse-units             = true      ,
1131   per-mode                = power     ,
1132   per-symbol              = /         ,
1133   qualifier-mode          = subscript ,
1134   qualifier-phrase        =           ,
1135   sticky-per              = false     ,
1136 }
1137 </package>

```

References

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

Part XI

siunitx-abbreviations – Abbreviations

`\A` Abbreviations for currents.
`\pA`
`\nA`
`\uA`
`\mA`
`\kA`

`\fg` Abbreviations for masses.
`\pg`
`\ng`
`\ug`
`\mg`
`\g`
`\kg`

`\K` Abbreviations for temperature.

`\m` Abbreviations for lengths.
`\pm`
`\nm`
`\um`
`\mm`
`\cm`
`\dm`
`\km`

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

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

<code>\mol</code>	Abbreviations for moles.
<code>\fmol</code>	
<code>\pmol</code>	
<code>\nmol</code>	
<code>\umol</code>	
<code>\mmol</code>	
<code>\kmol</code>	

<code>\V</code>	Abbreviations for potentials.
<code>\pV</code>	
<code>\nV</code>	
<code>\uV</code>	
<code>\mV</code>	
<code>\kV</code>	

<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>	

<code>\W</code>	Abbreviations for powers.
<code>\uW</code>	
<code>\mW</code>	
<code>\kW</code>	
<code>\MW</code>	
<code>\GW</code>	

<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>	

<code>\N</code>	Abbreviations for forces.
<code>\mN</code>	
<code>\kN</code>	
<code>\MN</code>	

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

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

(End definition for \A and others. These functions are documented on page 170.)

`\Hz` Then frequencies.

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

(End definition for \Hz and others. These functions are documented on page 170.)

`\mol` Amounts of substance (moles).
`\fmol` 14 `\siunitx_declare_unit:Nn \mol { \mole }`
`\pmol` 15 `\siunitx_declare_unit:Nn \fmol { \femto \mole }`
`\nmol` 16 `\siunitx_declare_unit:Nn \pmol { \pico \mole }`
`\umol` 17 `\siunitx_declare_unit:Nn \nmol { \nano \mole }`
`\mmol` 18 `\siunitx_declare_unit:Nn \umol { \micro \mole }`
`\kmmol` 19 `\siunitx_declare_unit:Nn \mmol { \milli \mole }`
20 `\siunitx_declare_unit:Nn \kmmol { \kilo \mole }`

(End definition for `\mol` and others. These functions are documented on page 171.)

`\V` Potentials.
`\pV` 21 `\siunitx_declare_unit:Nn \V { \volt }`
`\nV` 22 `\siunitx_declare_unit:Nn \pV { \pico \volt }`
`\uV` 23 `\siunitx_declare_unit:Nn \nV { \nano \volt }`
`\mV` 24 `\siunitx_declare_unit:Nn \uV { \micro \volt }`
`\kV` 25 `\siunitx_declare_unit:Nn \mV { \milli \volt }`
26 `\siunitx_declare_unit:Nn \kV { \kilo \volt }`

(End definition for `\V` and others. These functions are documented on page 171.)

`\hl` Volumes.
`\l` 27 `\siunitx_declare_unit:Nn \hl { \hecto \litre }`
`\ml` 28 `\siunitx_declare_unit:Nn \l { \litre }`
`\ul` 29 `\siunitx_declare_unit:Nn \ml { \milli \litre }`
`\hL` 30 `\siunitx_declare_unit:Nn \ul { \micro \litre }`
`\L` 31 `\siunitx_declare_unit:Nn \hL { \hecto \liter }`
`\mL` 32 `\siunitx_declare_unit:Nn \L { \liter }`
`\uL` 33 `\siunitx_declare_unit:Nn \mL { \milli \liter }`
34 `\siunitx_declare_unit:Nn \uL { \micro \liter }`

(End definition for `\hl` and others. These functions are documented on page 171.)

`\fg` Masses.
`\pg` 35 `\siunitx_declare_unit:Nn \fg { \femto \gram }`
`\ng` 36 `\siunitx_declare_unit:Nn \pg { \pico \gram }`
`\ug` 37 `\siunitx_declare_unit:Nn \ng { \nano \gram }`
`\mg` 38 `\siunitx_declare_unit:Nn \ug { \micro \gram }`
`\g` 39 `\siunitx_declare_unit:Nn \mg { \milli \gram }`
`\kg` 40 `\siunitx_declare_unit:Nn \g { \gram }`
41 `\siunitx_declare_unit:Nn \kg { \kilo \gram }`

(End definition for `\fg` and others. These functions are documented on page 170.)

`\W` Energies and powers
`\uW` 42 `\siunitx_declare_unit:Nn \W { \watt }`
`\mW` 43 `\siunitx_declare_unit:Nn \uW { \micro \watt }`
`\kW` 44 `\siunitx_declare_unit:Nn \mW { \milli \watt }`
`\MW` 45 `\siunitx_declare_unit:Nn \kW { \kilo \watt }`
`\GW` 46 `\siunitx_declare_unit:Nn \MW { \mega \watt }`
`\kJ` 47 `\siunitx_declare_unit:Nn \GW { \giga \watt }`
`\J` 48 `\siunitx_declare_unit:Nn \J { \joule }`
`\mJ` 49 `\siunitx_declare_unit:Nn \uJ { \micro \joule }`
`\uJ` 50 `\siunitx_declare_unit:Nn \mJ { \milli \joule }`
51 `\siunitx_declare_unit:Nn \kJ { \kilo \joule }`
`\eV`
`\meV`
`\keV`
`\MeV`
`\GeV`
`\TeV`
`\kWh`

```

52 \siunitx_declare_unit:Nn \eV { \electronvolt }
53 \siunitx_declare_unit:Nn \meV { \milli \electronvolt }
54 \siunitx_declare_unit:Nn \keV { \kilo \electronvolt }
55 \siunitx_declare_unit:Nn \MeV { \mega \electronvolt }
56 \siunitx_declare_unit:Nn \GeV { \giga \electronvolt }
57 \siunitx_declare_unit:Nn \TeV { \tera \electronvolt }
58 \siunitx_declare_unit:Nnn \kWh { \kilo \watt \hour }
59 { inter-unit-product = }

```

(End definition for \W and others. These functions are documented on page 171.)

\m Lengths.

```

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

```

(End definition for \m and others. These functions are documented on page 170.)

\K Temperatures.

```

68 \siunitx_declare_unit:Nn \K { \kelvin }

```

(End definition for \K. This function is documented on page 170.)

\dB

```

69 \siunitx_declare_unit:Nn \dB { \deci \bel }

```

(End definition for \dB. This function is documented on page 172.)

\F Capacitance.

```

\fF 70 \siunitx_declare_unit:Nn \F { \farad }
\pF 71 \siunitx_declare_unit:Nn \fF { \femto \farad }
72 \siunitx_declare_unit:Nn \pF { \pico \farad }

```

(End definition for \F, \fF, and \pF. These functions are documented on page 172.)

\N Forces.

```

\mN 73 \siunitx_declare_unit:Nn \N { \newton }
\kN 74 \siunitx_declare_unit:Nn \mN { \milli \newton }
\MN 75 \siunitx_declare_unit:Nn \kN { \kilo \newton }
76 \siunitx_declare_unit:Nn \MN { \mega \newton }

```

(End definition for \N and others. These functions are documented on page 171.)

\Pa Pressures.

```

\kPa 77 \siunitx_declare_unit:Nn \Pa { \pascal }
\MPa 78 \siunitx_declare_unit:Nn \kPa { \kilo \pascal }
\GPa 79 \siunitx_declare_unit:Nn \MPa { \mega \pascal }
80 \siunitx_declare_unit:Nn \GPa { \giga \pascal }

```

(End definition for \Pa and others. These functions are documented on page 172.)

\mohm Resistances.

\kohm 81 `\siunitx_declare_unit:Nn \mohm { \milli \ohm }`

\Mohm 82 `\siunitx_declare_unit:Nn \kohm { \kilo \ohm }`

83 `\siunitx_declare_unit:Nn \Mohm { \mega \ohm }`

(End definition for \mohm, \kohm, and \Mohm. These functions are documented on page 172.)

\s Finally, times.

\as 84 `\siunitx_declare_unit:Nn \s { \second }`

\fs 85 `\siunitx_declare_unit:Nn \as { \atto \second }`

\ps 86 `\siunitx_declare_unit:Nn \fs { \femto \second }`

\ns 87 `\siunitx_declare_unit:Nn \ps { \pico \second }`

\us 88 `\siunitx_declare_unit:Nn \ns { \nano \second }`

\ms 89 `\siunitx_declare_unit:Nn \us { \micro \second }`

90 `\siunitx_declare_unit:Nn \ms { \milli \second }`

(End definition for \s and others. These functions are documented on page 170.)

91 `\</package>`

Part XII

siunitx-binary – Binary units

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

<code>\kibi</code>	Prefixes, all of which are integer powers of 2: the powers are <i>not</i> stored or available for conversion.
<code>\mebi</code>	
<code>\gibi</code>	
<code>\tebi</code>	
<code>\pebi</code>	
<code>\exbi</code>	
<code>\zebi</code>	
<code>\yobi</code>	

<code>\bit</code>	Units for bits and bytes.
<code>\byte</code>	

1 siunitx-binary implementation

Start the DocStrip guards.

```
1 <*package>
```

```
\kibi All very simple.
\mebi 2 \siunitx_declare_prefix:Nn \kibi { Ki }
\gibi 3 \siunitx_declare_prefix:Nn \mebi { Mi }
\tebi 4 \siunitx_declare_prefix:Nn \gibi { Gi }
\pebi 5 \siunitx_declare_prefix:Nn \tebi { Ti }
\exbi 6 \siunitx_declare_prefix:Nn \pebi { Pi }
\zebi 7 \siunitx_declare_prefix:Nn \exbi { Ei }
\yobi 8 \siunitx_declare_prefix:Nn \zebi { Zi }
      9 \siunitx_declare_prefix:Nn \yobi { Yi }
```

(End definition for `\kibi` and others. These functions are documented on page 176.)

```
\bit
\byte 10 \siunitx_declare_unit:Nn \bit { bit }
      11 \siunitx_declare_unit:Nn \byte { B }
```

(End definition for `\bit` and `\byte`. These functions are documented on page 176.)

```
12 </package>
```

Part XIII

siunitx-command – Units as document command

1 Creating units as document commands

<code>\siunitx_command_create:</code>	<code>\siunitx_command_create:</code>
---------------------------------------	---------------------------------------

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

1.1 Key-value options

The options defined by this submodule are available within the l3keys `siunitx` tree. These options are all preamble-only.

<code>free-standing-units</code>	<code>free-standing-units = true false</code>
----------------------------------	---

<code>overwrite-commands</code>	<code>overwrite-commands = true false</code>
---------------------------------	--

<code>space-before-unit</code>	<code>space-before-unit = true false</code>
--------------------------------	---

<code>unit-optional-argument</code>	<code>unit-optional-argument = true false</code>
-------------------------------------	--

<code>use-xspace</code>	<code>use-xspace = true false</code>
-------------------------	--------------------------------------

2 siunitx-command 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_command>`

`\l__siunitx_command_tmp_tl`

3 `\tl_new:N \l__siunitx_command_tmp_tl`

(End definition for `\l__siunitx_command_tmp_tl`.)

2.1 Options

```

\l_siunitx_command_create_bool
\l_siunitx_command_overwrite_bool
\l_siunitx_command_prespace_bool
\l_siunitx_command_optarg_bool
\l_siunitx_command_xspace_bool

4 \keys_define:nn { siunitx }
5   {
6     free-standing-units .bool_set:N =
7       \l_siunitx_command_create_bool ,
8     overwrite-commands .bool_set:N =
9       \l_siunitx_command_overwrite_bool ,
10    space-before-unit .bool_set:N =
11      \l_siunitx_command_prespace_bool ,
12    unit-optional-argument .bool_set:N =
13      \l_siunitx_command_optarg_bool ,
14    use-xspace .bool_set:N =
15      \l_siunitx_command_xspace_bool
16  }

```

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

These preamble-only options are all disabled at the start of the document.

```

17 \AtBeginDocument
18   {
19     \clist_map_inline:nn
20       {
21         free-standing-units ,
22         overwrite-commands ,
23         space-before-unit ,
24         unit-optional-argument ,
25         use-xspace
26       }
27     {
28       \keys_define:nn { siunitx }
29         {
30           #1 .code:n =
31             { \msg_warning:nnn { siunitx } { option-preamble-only } {#1} }
32         }
33     }
34   }
35 \msg_new:nnn { siunitx } { option-preamble-only }
36   { Option~'#1'~only-available-in-the-preamble. }

```

2.2 Creation of unit document commands

`\siunitx_command_create:` Creating document commands is all done by a single function which is set up using expansion: that way the tests are only run once. Other than that, this is all just a question of picking up all the various routes. Where the `soulpos` package is loaded *after* `siunitx`, the commands `\hl` and `\ul` will be created only after the hook is used. The `soul` package creates those using `\newcommand`, so we have to avoid an issue.

```

37 \cs_new_protected:Npn \siunitx_command_create:
38   {
39     \bool_if:NT \l_siunitx_command_create_bool
40     {
41       \__siunitx_command_create:
42       \@ifpackageloaded { soulpos }

```

```

43     {
44       \@ifpackageloaded { soul }
45       { }
46       {
47         \cs_undefine:N \hl
48         \cs_undefine:N \ul
49       }
50     }
51     { }
52   }

```

At the beginning of table cells and inside x-type expansion, all symbolic units need to have *some* definition.

```

53   \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
54   {
55     \cs_if_free:NT ##1
56     { \cs_set_protected:Npn ##1 { \ERROR } }
57   }
58 }
59 \AtBeginDocument { \siunitx_command_create: }
60 \cs_new_protected:Npn \__siunitx_command_create:
61 {
62   \bool_if:NT \l__siunitx_command_xspace_bool
63   { \RequirePackage { xspace } }
64   \bool_if:NT \l__siunitx_command_overwrite_bool
65   {
66     \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
67     { \cs_undefine:N ##1 }
68   }
69   \cs_set_protected:Npx \__siunitx_command_create:N ##1
70   {
71     \ProvideDocumentCommand ##1 { \bool_if:NT \l__siunitx_command_optarg_bool { o } }
72     {
73       \mode_leave_vertical:
74       \group_begin:
75       \bool_if:NTF \l__siunitx_command_optarg_bool
76       { \exp_not:N \IfNoValueTF {####1} }
77       { \use_i:nn }
78       {
79         \siunitx_unit_options_apply:n {##1}
80         \bool_if:NT \l__siunitx_command_prespace_bool { \exp_not:N \ }
81         \siunitx_unit_format:nN {##1}
82         \exp_not:N \l__siunitx_command_tmp_tl
83         \siunitx_print:nV { unit }
84         \exp_not:N \l__siunitx_command_tmp_tl
85       }
86       { \siunitx_quantity:nn {####1} {##1} }
87       \group_end:
88       \bool_if:NT \l__siunitx_command_xspace_bool { \exp_not:N \xspace }
89     }
90   }
91   \seq_map_function:NN \l_siunitx_unit_seq \__siunitx_command_create:N
92 }
93 \cs_new_protected:Npn \__siunitx_command_create:N #1 { }

```

(End definition for `\siunitx_command_create:`, `_siunitx_command_create:`, and `_siunitx_command_create:N`. This function is documented on page [177](#).)

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.

```
94 \keys_set:nn { siunitx }
95   {
96     free-standing-units    = false ,
97     overwrite-commands    = false ,
98     space-before-unit     = false ,
99     unit-optional-argument = false ,
100    use-xspace              = false
101  }
102 \end{package}
```


Part XIV

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>
   Start the DocStrip guards.
2 <*package>
3 <*options>
   Some messages.
4 \msg_new:nnn { siunitx } { option-deprecated }
5 {
6   Option~"#1"~has~been~deprecated~in~this~release.\\ \\
7   Use~"#2"~as~a~replacement.
8 }
9 \msg_new:nnn { siunitx } { option-removed }
10 { 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.

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

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

```
\_siunitx_option_removed:n
\_siunitx_option_removed:V
```

Abstract out a simple wrapper.

```
22 \cs_new_protected:Npn \_siunitx_option_removed:n #1
23 {
24   \msg_warning:nnx { siunitx } { option-removed }
25   {#1}
26 }
27 \cs_generate_variant:Nn \_siunitx_option_removed:n { V }
```

(End definition for `_siunitx_option_removed:n`.)

1.1 Load-time option

```
28 \clist_map_inline:nn { abbreviations , binary-units , version-1-compatibility }
29 {
30   \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
31 }
```

1.2 Angle options

All straight-forward emulation.

```
32 \keys_define:nn { siunitx }
33 {
34   add-arc-degree-zero .code:n =
35   {
36     \__siunitx_option_deprecated:nnV
37     { add-arc-degree-zero }
38     { fill-arc-degrees }
39     \l_keys_value_tl
40   } ,
41   add-arc-degree-zero .default:n = true ,
42   add-arc-minute-zero .code:n =
43   {
44     \__siunitx_option_deprecated:nnV
45     { add-arc-minute-zero }
46     { fill-arc-minutes }
47     \l_keys_value_tl
48   } ,
49   add-arc-minute-zero .default:n = true ,
50   add-arc-second-zero .code:n =
51   {
52     \__siunitx_option_deprecated:nnV
53     { add-arc-second-zero }
54     { fill-arc-seconds }
55     \l_keys_value_tl
56   } ,
57   add-arc-second-zero .default:n = true
58 }
```

1.3 Combination functions options

```
59 \keys_define:nn { siunitx }
60 {
61   list-units / brackets .code:n =
62   {
63     \__siunitx_option_deprecated:nn
64     { list-units~~brackets }
65     { list-units~~bracket }
66   } ,
67   range-units / brackets .code:n =
68   {
69     \__siunitx_option_deprecated:nn
70     { range-units~~brackets }
71     { range-units~~bracket }
72   } ,
73   product-units / brackets .code:n =
```

```

74     {
75         \_siunitx_option_deprecated:nn
76         { product-units~==brackets }
77         { product-units~==bracket }
78     }
79 }

```

1.4 Command options

```

80 \keys_define:nn { siunitx }
81 {
82     overwrite-functions .code:n =
83     {
84         \_siunitx_option_deprecated:nnV
85         { overwrite-functions }
86         { overwrite-commands }
87         \l_keys_value_tl
88     } ,
89     overwrite-functions .default:n = true
90 }

```

1.5 Print options

```

91 \clist_map_inline:nn
92 {
93     detect-all      ,
94     detect-display-math ,
95     detect-family    ,
96     detect-inline-family ,
97     detect-inline-weight ,
98     detect-mode      ,
99     detect-none      ,
100    detect-shape      ,
101    detect-weight
102 }
103 {
104     \keys_define:nn { siunitx } { #1 .code:n = \_siunitx_option_removed:n {#1} }
105 }

```

The old font insertion options.

```

106 \clist_map_inline:nn
107 {
108     math-rm      ,
109     math-sf      ,
110     math-tt      ,
111     number-math-rm ,
112     number-math-sf ,
113     number-math-tt ,
114     number-text-rm ,
115     number-text-sf ,
116     number-text-tt ,
117     text-rm      ,
118     text-sf      ,
119     text-tt      ,
120     unit-math-rm ,
121     unit-math-sf ,

```

```

122     unit-math-tt    ,
123     unit-text-rm    ,
124     unit-text-sf     ,
125     unit-text-tt
126 }
127 {
128     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
129 }

```

1.6 Symbol options

```

130 \clist_map_inline:nn
131 {
132     math-angstrom    ,
133     math-arcminute   ,
134     math-arcsecond   ,
135     math-celsius     ,
136     math-degree      ,
137     math-micro       ,
138     math-ohm         ,
139     text-angstrom    ,
140     text-arcminute   ,
141     text-arcsecond   ,
142     text-celsius     ,
143     text-degree      ,
144     text-micro       ,
145     text-ohm         ,
146 }
147 {
148     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
149 }

```

1.7 Number options

A small number of removed options.

```

150 \clist_map_inline:nn
151 {
152     input-protect-tokens ,
153     input-quotient       ,
154     quotient-mode
155 }
156 {
157     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
158 }

```

Options for number processing: largely removals.

```

159 \keys_define:nn { siunitx }
160 {
161     add-decimal-zero .choice: ,
162     add-decimal-zero / false .code:n =
163     {
164         \__siunitx_option_deprecated:nn
165         { add-decimal-zero }
166         { minimum-decimal-digits~=-0 }
167     } ,
168     add-decimal-zero / true .code:n =

```

```

169     {
170         \_siunitx_option_deprecated:nn
171         { add-decimal-zero }
172         { minimum-decimal-digits~==1 }
173     } ,
174     add-decimal-zero .default:n = true ,
175     add-integer-zero .code:n =
176     { \_siunitx_option_removed:V \l_keys_key_tl } ,
177     close-bracket .code:n =
178     { \_siunitx_option_removed:V \l_keys_key_tl } ,
179     bracket-numbers .choice: ,
180     bracket-numbers / false .code:n =
181     {
182         \_siunitx_option_deprecated:nn
183         { bracket-numbers }
184         { bracket-ambiguous-numbers~==false }
185     } ,
186     bracket-numbers / true .code:n =
187     {
188         \_siunitx_option_deprecated:nn
189         { bracket-numbers }
190         { bracket-ambiguous-numbers~==true }
191     } ,
192     bracket-numbers .default:n = true ,
193     explicit-sign .code:n =
194     {
195         \str_if_eq:nnTF {#1} { + }
196         {
197             \_siunitx_option_deprecated:nn
198             { explicit-sign }
199             { print-implicit-plus~==true }
200         }
201         { \_siunitx_option_removed:V \l_keys_key_tl }
202     } ,
203     omit-uncertainty .code:n =
204     {
205         \_siunitx_option_deprecated:nnV
206         { omit-uncertainty }
207         { drop-uncertainty }
208         \l_keys_value_tl
209     } ,
210     omit-uncertainty .default:n = true ,
211     open-bracket .code:n =
212     { \_siunitx_option_removed:V \l_keys_key_tl } ,
213     retain-unity-mantissa .code:n =
214     {
215         \_siunitx_option_deprecated:nnV
216         { retain-unity-mantissa }
217         { print-unity-mantissa }
218         \l_keys_value_tl
219     } ,
220     retain-unity-mantissa .default:n = true ,
221     retain-zero-exponent .code:n =
222     {

```

```

223     \_siunitx_option_deprecated:nnV
224     { retain-zero-exponent }
225     { print-zero-exponent }
226     \l_keys_value_tl
227   } ,
228   retain-zero-exponent .default:n = true ,
229   scientific-notation .choice: ,
230   scientific-notation / engineering .code:n =
231   {
232     \_siunitx_option_deprecated:nn
233     { scientific-notation~~engineering }
234     { exponent-mode~~engineering }
235   } ,
236   scientific-notation / fixed .code:n =
237   {
238     \_siunitx_option_deprecated:nn
239     { scientific-notation~~fixed }
240     { exponent-mode~~fixed }
241   } ,
242   scientific-notation / false .code:n =
243   {
244     \_siunitx_option_deprecated:nn
245     { scientific-notation~~false }
246     { exponent-mode~~input }
247   } ,
248   scientific-notation / true .code:n =
249   {
250     \_siunitx_option_deprecated:nn
251     { scientific-notation~~true }
252     { exponent-mode~~scientific }
253   } ,
254   scientific-notation .default:n = true ,
255   zero-decimal-to-integer .code:n =
256   {
257     \_siunitx_option_deprecated:nnV
258     { zero-decimal-to-integer }
259     { drop-zero-decimal }
260     \l_keys_value_tl
261   } ,
262   zero-decimal-to-integer .default:n = true
263 }

```

1.7.1 Table options

All straight-forward emulation.

```

264 \keys_define:nn { siunitx }
265 {
266   table-align-text-post .code:n =
267   {
268     \_siunitx_option_deprecated:nnV
269     { table-align-text-post }
270     { table-align-text-after }
271     \l_keys_value_tl
272   } ,

```

```

273 table-align-text-post .default:n = true ,
274 table-align-text-pre .code:n =
275 {
276   \__siunitx_option_deprecated:nnV
277   { table-align-text-pre }
278   { table-align-text-before }
279   \l_keys_value_tl
280 } ,
281 table-align-text-pre .default:n = true ,
282 table-number-alignment / center-decimal-marker .code:n =
283 {
284   \msg_info:nnnn { siunitx } { option-deprecated }
285   { table-number-alignment~==center-decimal-marker }
286   { table-alignment-mode~==marker }
287   \keys_set:nn
288   { siunitx }
289   { table-alignment-mode = marker }
290 } ,
291 table-omit-exponent .code:n =
292 {
293   \__siunitx_option_deprecated:nnV
294   { table-omit-exponent }
295   { drop-exponent }
296   \l_keys_value_tl
297 } ,
298 table-omit-exponent .default:n = true ,
299 table-parse-only .code:n =
300 {
301   \msg_info:nnnn { siunitx } { option-deprecated }
302   { table-parse-only }
303   { table-alignment-mode~==none }
304   \str_if_eq:VnTF \l_keys_value_tl { false }
305   {
306     \keys_set:nn
307     { siunitx }
308     { table-alignment-mode = marker }
309   }
310   {
311     \keys_set:nn
312     { siunitx }
313     { table-alignment-mode = none }
314   }
315 } ,
316 table-space-text-post .code:n =
317 {
318   \msg_info:nnnn { siunitx } { option-deprecated }
319   { table-space-text-post }
320   { table-format }
321   \tl_set:Nn \l__siunitx_table_after_model_tl {#1}
322 } ,
323 table-space-text-pre .code:n =
324 {
325   \msg_info:nnnn { siunitx } { option-deprecated }
326   { table-space-text-post }

```

```

327         { table-format }
328         \tl_set:Nn \l__siunitx_table_before_model_tl {#1}
329     }
330 }

\__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

331 \cs_new_protected:Npn \__siunitx_option_table_format:n #1
332 {
333     \msg_info:nnnn { siunitx } { option-deprecated }
334     { table- #1 }
335     { table-format }
336     \tl_set:Nx \l__siunitx_table_format_tl
337     {
338         \cs:w __siunitx_option_table_ #1 :nnnnnnnn
339         \exp_after:wN \exp_after:wN \exp_after:wN \cs_end:
340         \exp_after:wN \l__siunitx_table_format_tl
341         \exp_after:wN { \l_keys_value_tl }
342     }
343     \exp_after:wN \__siunitx_table_generate_model:nnnnnnn
344     \l__siunitx_table_format_tl
345 }
346 \cs_new:Npn \__siunitx_option_table_comparator:nnnnnnnn #1#2#3#4#5#6#7#8
347 { \exp_not:n { {#8} {#2} {#3} {#4} {#5} {#6} {#7} } }
348 \cs_new:cpn { __siunitx_option_table_figures-decimal:nnnnnnnn }
349 #1#2#3#4#5#6#7#8
350 { \exp_not:n { {#1} {#2} {#3} {#8} {#5} {#6} {#7} } }
351 \cs_new:cpn { __siunitx_option_table_figures-exponent:nnnnnnnn }
352 #1#2#3#4#5#6#7#8
353 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#8} } }
354 \cs_new:cpn { __siunitx_option_table_figures-integer:nnnnnnnn }
355 #1#2#3#4#5#6#7#8
356 { \exp_not:n { {#1} {#2} {#8} {#4} {#5} {#6} {#7} } }
357 \cs_new:cpn { __siunitx_option_table_figures-uncertainty:nnnnnnnn }
358 #1#2#3#4#5#6#7#8
359 { \exp_not:n { {#1} {#2} {#3} {#4} { { S } {#8} } {#6} {#7} } }
360 \cs_new:cpn { __siunitx_option_table_sign-exponent:nnnnnnnn }
361 #1#2#3#4#5#6#7#8
362 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#8} {#7} } }
363 \cs_new:cpn { __siunitx_option_table_sign-mantissa:nnnnnnnn }
364 #1#2#3#4#5#6#7#8
365 { \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.

```

366 \keys_define:nn { siunitx }
367 {
368     table-comparator .code:n =
369     { \__siunitx_option_table_format:n { comparator } } ,
370     table-figures-decimal .code:n =
371     { \__siunitx_option_table_format:n { figures-decimal } } ,
372     table-figures-exponent .code:n =
373     { \__siunitx_option_table_format:n { figures-exponent } } ,
374     table-figures-integer .code:n =
375     { \__siunitx_option_table_format:n { figures-integer } } ,

```



```

376 table-figures-uncertainty .code:n =
377   { \_siunitx_option_table_format:n { figures-uncertainty } } ,
378 table-sign-exponent .code:n =
379   { \_siunitx_option_table_format:n { sign-exponent } } ,
380 table-sign-mantissa .code:n =
381   { \_siunitx_option_table_format:n { sign-mantissa } }
382 }

```

1.8 Unit options

```

383 \keys_define:nn { siunitx }
384 {
385   fraction-function .code:n =
386   {
387     \_siunitx_option_deprecated:nnV
388     { fraction-function }
389     { fraction-command }
390     \l_keys_value_tl
391   } ,
392   literal-superscript-as-power .code:n =
393   { \_siunitx_option_removed:V \l_keys_key_tl } ,
394   per-mode / reciprocal .code:n =
395   {
396     \_siunitx_option_deprecated:nn
397     { per-mode~~reciprocal }
398     { per-mode~~power }
399   } ,
400   per-mode / reciprocal-positive-first .code:n =
401   {
402     \_siunitx_option_deprecated:nn
403     { per-mode~~reciprocal-positive-first }
404     { per-mode~~power-positive-first }
405   } ,
406   power-font .code:n =
407   { \_siunitx_option_removed:V \l_keys_key_tl } ,
408   qualifier-mode / brackets .code:n =
409   {
410     \_siunitx_option_deprecated:nn
411     { qualifier-mode~~brackets }
412     { qualifier-mode~~bracket }
413   } ,
414   qualifier-mode / space .code:n =
415   {
416     \msg_info:nnnn { siunitx } { option-deprecated }
417     { qualifier-mode~~space }
418     { qualifier-mode~~phrase"~plus~"qualifier-phrase=\ }
419     \keys_set:nn
420     { siunitx }
421     { qualifier-mode = phrase, qualifier-phrase = \ }
422   } ,
423   qualifier-mode / text .code:n =
424   {
425     \_siunitx_option_deprecated:nn
426     { qualifier-mode~~text }

```

```

427         { qualifier-mode~~combine }
428     }
429 }

```

1.9 Quantity units

```

430 \keys_define:nn { siunitx }
431 {
432   allow-number-unit-breaks .code:n =
433   {
434     \__siunitx_option_deprecated:nnV
435     { allow-number-unit-breaks }
436     { allow-quantity-breaks }
437     \l_keys_value_tl
438   } ,
439   allow-number-unit-breaks .default:n = true ,
440   exponent-to-prefix .choice: ,
441   exponent-to-prefix / false .code:n =
442   {
443     \__siunitx_option_deprecated:nn
444     { exponent-to-prefix~~false }
445     { prefix-mode~~input }
446   } ,
447   exponent-to-prefix / true .code:n =
448   {
449     \__siunitx_option_deprecated:nn
450     { exponent-to-prefix~~true }
451     { prefix-mode~~combine-exponent }
452   } ,
453   exponent-to-prefix .default:n = true ,
454   multi-part-units .choice: ,
455   multi-part-units / brackets . code:n =
456   {
457     \__siunitx_option_deprecated:nn
458     { multi-part-units~~brackets }
459     { separate-uncertainty-units~~bracket }
460   } ,
461   multi-part-units / repeat . code:n =
462   {
463     \__siunitx_option_deprecated:nn
464     { multi-part-units~~repeat }
465     { separate-uncertainty-units~~repeat }
466   } ,
467   multi-part-units / single . code:n =
468   {
469     \__siunitx_option_deprecated:nn
470     { multi-part-units~~single }
471     { separate-uncertainty-units~~single }
472   } ,
473   number-unit-product .code:n =
474   {
475     \__siunitx_option_deprecated:nnV
476     { number-unit-product }
477     { quantity-product }
478     \l_keys_value_tl

```

```

479     } ,
480     prefixes-as-symbols .choice: ,
481     prefixes-as-symbols / false . code:n =
482     {
483         \_siunitx_option_deprecated:nn
484         { prefixes-as-symbols~~false }
485         { prefix-mode~~extract-exponent }
486     } ,
487     prefixes-as-symbols / true . code:n =
488     {
489         \_siunitx_option_deprecated:nn
490         { prefixes-as-symbols~~true }
491         { prefix-mode~~input }
492     } ,
493     prefixes-as-symbols .default:n = true
494 }
495 </options>

```

1.10 Preamble commands

```

496 <*interfaces>

```

`\DeclareBinaryPrefix` We simply drop #3.

```

497 \NewDocumentCommand \DeclareBinaryPrefix { +m m m }
498 {
499     \siunitx_declare_prefix:Nn #1 {#2}
500 }

```

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

`\DeclareSIPrePower` `\DeclareSIPostPower` Simply use a throw-away command for the part we do not need: this can be followed by some clean-up.

```

501 \NewDocumentCommand \DeclareSIPrePower { +m m }
502 {
503     \siunitx_declare_power:NNn #1 \_siunitx_tmp:w {#2}
504     \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
505 }
506 \NewDocumentCommand \DeclareSIPostPower { +m m }
507 {
508     \siunitx_declare_power:NNn \_siunitx_tmp:w #1 {#2}
509     \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
510 }

```

(End definition for \DeclareSIPrePower and \DeclareSIPostPower. These functions are documented on page ??.)

1.11 Document commands

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

```

511 \NewDocumentCommand \si { O { } m }
512 {
513     \mode_leave_vertical:
514     \group_begin:
515         \keys_set:nn { siunitx } {#1}
516         \siunitx_unit_format:nN {#2} \l_siunitx_tmp_tl

```

```

517     \siunitx_print:nV { unit } \l__siunitx_tmp_tl
518   \group_end:
519 }

```

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

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

```

520 \NewDocumentCommand \SI { O { } m o m }
521 {
522   \mode_leave_vertical:
523   \group_begin:
524     \keys_set:nn { siunitx } {#1}
525     \IfNoValueF {#3}
526     {
527       \siunitx_unit_format:nN {#3} \l__siunitx_tmp_tl
528       \siunitx_print:nV { unit } \l__siunitx_tmp_tl
529       \nobreak
530     }
531     \siunitx_quantity:nn {#2} {#4}
532   \group_end:
533 }

```

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

\SIlist Straight copies.

```

\SIrange 534 \NewDocumentCommand \SIlist
535 { O { } > { \SplitList { ; } } m > { \TrimSpaces } m }
536 {
537   \mode_leave_vertical:
538   \group_begin:
539     \siunitx_unit_options_apply:n {#3}
540     \keys_set:nn { siunitx } {#1}
541     \siunitx_quantity_list:nn {#2} {#3}
542   \group_end:
543 }
544 \NewDocumentCommand \SIrange { O { } m m > { \TrimSpaces } m }
545 {
546   \mode_leave_vertical:
547   \group_begin:
548     \siunitx_unit_options_apply:n {#4}
549     \keys_set:nn { siunitx } {#1}
550     \siunitx_quantity_range:nnn {#2} {#3} {#4}
551   \group_end:
552 }

```

(End definition for \SIlist and \SIrange. These functions are documented on page ??.)

```

553 \end{interfaces}

```

```

554 \end{package}

```

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

$\%$ 1077
 \backslash , 9, 15, 21, 27, 99, 130, 154, 1128, 1864, 1870
 $\backslash-$ 101
 $\backslash\backslash$ 6, 86, 1089
 $\backslash_$ 8, 210, 221, 289, 366
 $\backslash\sim$ 212
 $\backslash\sqcup$ 80, 114, 418, 421, 1096

A

$\backslash A$ 2, 170
allow-quantity-breaks 99
 \backslash ampere 2, 3, 4, 5, 6, 7, 134, 1009
 \backslash ang 6, 102, 279, 304
angle-mode 11
 \backslash approx 1866
arc-separator 11
arc-separator-over-decimal 11
 \backslash arcminute ... 51, 53, 135, 169, 337, 1074
 \backslash arcsecond ... 56, 58, 135, 170, 342, 1074
 \backslash as 84, 170
 \backslash astronomicalunit 135, 1061
 \backslash AtBeginDocument 5, 17, 39, 49,
59, 64, 129, 150, 200, 234, 246, 305, 308
 \backslash atto 85, 134, 1018

B

\backslash becquerel 135, 1039
 \backslash begin 193
 \backslash bel 69, 135, 1061
 \backslash bfseries 86
 \backslash bit 10, 176
 \backslash boldmath 87
bool commands:
 \backslash bool_if:NTF ... 10, 16, 17, 18, 32,
34, 39, 41, 50, 55, 62, 64, 71, 75, 80,
88, 91, 94, 106, 110, 119, 120, 121,
138, 145, 145, 158, 162, 168, 177,
183, 183, 191, 192, 233, 233, 235,
244, 246, 252, 257, 262, 356, 380,
381, 407, 415, 422, 431, 437, 441,
454, 476, 477, 510, 521, 529, 536,
555, 600, 676, 691, 699, 750, 759,
837, 857, 869, 886, 906, 920, 929,
932, 932, 942, 1219, 1383, 1545,
1553, 1569, 1617, 1695, 1764, 1775
 \backslash bool_lazy_all:nTF 1515, 1592

\backslash bool_lazy_all_p:n 1030, 1048
 \backslash bool_lazy_and:nnTF 109,
137, 217, 247, 262, 431, 567, 576,
584, 623, 647, 871, 879, 1312, 1759
 \backslash bool_lazy_or:nnTF 10,
105, 120, 157, 197, 237, 317, 371,
443, 506, 988, 1027, 1045, 1343, 1755
 \backslash bool_lazy_or_p:nn 1521
 \backslash bool_new:N 3, 6, 9, 10, 10, 11,
17, 18, 19, 20, 21, 22, 37, 38, 76, 88,
332, 407, 545, 550, 551, 552, 553,
554, 555, 558, 610, 1423, 1482, 1483
 \backslash bool_set_false:N
..... 9, 12, 16, 21, 22, 25,
28, 29, 34, 35, 38, 39, 49, 56, 57, 63,
63, 67, 69, 73, 74, 79, 80, 81, 134,
148, 155, 171, 199, 216, 236, 240,
323, 338, 339, 418, 497, 498, 504,
505, 506, 507, 511, 512, 513, 518,
521, 525, 583, 585, 599, 659, 698,
746, 781, 853, 1444, 1448, 1453, 1454
 \backslash bool_set_true:N ... 11, 14, 17, 24,
29, 30, 33, 34, 51, 55, 61, 62, 68, 75,
102, 141, 163, 189, 198, 214, 225,
234, 288, 337, 416, 433, 499, 500,
514, 519, 520, 526, 527, 528, 532,
533, 534, 535, 586, 597, 843, 874,
913, 1438, 1439, 1443, 1449, 1790, 1791
 \backslash c_false_bool 47, 121, 387, 449, 453,
458, 460, 467, 491, 497, 511, 515, 538
 \backslash c_true_bool 44, 118,
372, 387, 447, 460, 464, 491, 497, 511
box commands:
 \backslash box_clear:N 440, 499
 \backslash box_new:N
..... 3, 6, 164, 165, 229, 230, 416, 417
 \backslash box_use:N 239
 \backslash box_use_drop:N
..... 234, 236, 365, 366, 410, 411,
463, 470, 483, 484, 543, 544, 545, 546
 \backslash box_wd:N 218, 219, 238, 239,
242, 250, 253, 342, 381, 385, 399,
438, 445, 446, 449, 457, 497, 503,
534, 558, 569, 570, 571, 586, 590,
591, 604, 605, 615, 623, 624, 626,
643, 644, 665, 676, 695, 697, 707, 719
bracket-ambiguous-numbers 38
bracket-negative-numbers 38

bracket-unit-denominator	136	486, 549, 619, 673, 674, 679, 686,	
\byte	10, 176	688, 692, 694, 711, 714, 726, 737,	
C			
\cancel	107, 130, 136, 142	743, 749, 755, 767, 774, 792, 814,	
\candela	134, 1009	820, 821, 828, 848, 853, 867, 877,	
\cdot	8, 33	890, 898, 911, 924, 938, 950, 962,	
\centi	65, 134, 1018	964, 970, 979, 992, 995, 996, 1001,	
char commands:		1009, 1020, 1041, 1059, 1089, 1116,	
\char_generate:nn	8, 15, 26, 28, 162, 173,	1123, 1130, 1132, 1135, 1146, 1156,	
	175, 210, 221, 289, 366, 993, 1004, 1006	1163, 1170, 1171, 1172, 1179, 1181,	
\char_set_active_eq:NN	222, 224, 226, 348, 389	1187, 1192, 1206, 1214, 1223, 1238,	
\char_set_catcode_active:N	101	1246, 1264, 1280, 1281, 1299, 1308,	
\char_set_catcode_active:n	212	1310, 1332, 1341, 1358, 1367, 1389,	
\char_set_mathcode:nn	349, 390	1399, 1401, 1406, 1411, 1413, 1484,	
\char_to_utfviii_bytes:n	21, 168, 999	1486, 1488, 1490, 1492, 1497, 1502,	
\char_value_catcode:n	15, 162, 993	1513, 1528, 1535, 1541, 1561, 1567,	
clist commands:		1573, 1582, 1590, 1601, 1615, 1626,	
\clist_map_break:	113	1635, 1637, 1639, 1641, 1647, 1657,	
\clist_map_function:nN	36, 41	1662, 1682, 1691, 1693, 1712, 1732,	
\clist_map_inline:nn	19, 28, 91, 101,	1747, 1752, 1784, 1819, 1825, 1830	
	106, 114, 116, 117, 130, 150, 476, 588	\cs_new:Npx	245, 375, 384
\cm	60, 170	\cs_new_eq:NN	204, 231, 712
color	86	\cs_new_protected:Npn	7, 11,
\complexnum	183		14, 14, 16, 19, 22, 23, 24, 25, 27, 30,
\complexqty	183		36, 37, 38, 39, 40, 40, 44, 49, 49, 53,
compound-exponents	20		55, 60, 61, 62, 64, 70, 70, 73, 77, 78,
compound-final-separator	20		84, 84, 85, 88, 89, 89, 91, 93, 94, 95,
compound-pair-separator	20		98, 99, 101, 103, 104, 105, 108, 115,
compound-separator	20		118, 120, 122, 127, 131, 131, 132,
compound-separator-mode	20		133, 133, 133, 134, 137, 139, 139,
compound-units	20		141, 143, 144, 146, 149, 150, 151,
\coulomb	135, 1039		153, 156, 160, 160, 160, 164, 165,
\cr	28, 112		166, 166, 166, 168, 171, 172, 173,
cs commands:			176, 181, 183, 185, 186, 187, 187,
\cs:w	111, 338, 662, 956		189, 194, 194, 195, 208, 208, 209,
\cs_end:	111, 339, 665, 959		212, 216, 221, 223, 223, 224, 226,
\cs_generate_variant:Nn			226, 228, 230, 235, 243, 244, 248,
	2, 3, 3, 4, 5, 5, 6, 21, 27,		248, 248, 251, 259, 260, 260, 262,
	61, 72, 87, 98, 114, 142, 149, 163,		265, 269, 270, 272, 276, 282, 282,
	170, 172, 193, 207, 218, 228, 281,		285, 293, 294, 297, 299, 307, 315,
	352, 360, 434, 458, 736, 791, 819,		317, 318, 321, 321, 323, 325, 326,
	903, 969, 1131, 1134, 1145, 1614, 1746		329, 331, 331, 334, 335, 337, 337,
\cs_if_eq:NNTF	355		344, 345, 347, 347, 350, 350, 352,
\cs_if_exist:NTF			352, 353, 354, 360, 361, 367, 368,
	99, 132, 135, 225, 252, 323		368, 371, 375, 376, 376, 383, 390,
\cs_if_free:NTF	7, 55		390, 394, 398, 403, 404, 410, 414,
\cs_new:Npn	14, 18, 23, 67, 130,		415, 425, 429, 429, 432, 432, 435,
	140, 161, 162, 165, 170, 209, 210,		439, 441, 443, 445, 451, 456, 462,
	229, 232, 240, 246, 247, 321, 328,		469, 472, 472, 474, 479, 481, 486,
	330, 345, 346, 348, 351, 353, 354,		493, 500, 546, 551, 551, 557, 562,
	357, 360, 363, 363, 368, 370, 373,		565, 580, 594, 602, 614, 633, 634,
			636, 645, 648, 650, 656, 658, 660,
			670, 681, 684, 689, 694, 696, 713,
			718, 723, 748, 754, 776, 786, 788,
			797, 808, 809, 819, 831, 845, 850,

862, 867, 893, 904, 906, 916, 925, 930, 937, 940, 952, 953, 962, 971, 1381	
\cs_new_protected:Npx	102, 171, 179, 195, 200, 204, 213, 219, 278, 359
\cs_set:Npn	30, 34
\cs_set_eq:NN	156, 180, 311, 316, 325
\cs_set_protected:Npn	28, 56, 68, 85, 207, 231, 238, 250, 255, 277, 309, 333, 434
\cs_set_protected:Npx	69, 198
\cs_to_str:N	275, 317, 1077
\cs_undefine:N	47, 48, 67, 227, 254
\cubed	136, 1078
\cubic	135, 1078
D	
\dalton	135, 1061
\day	135, 1061
\dB	69, 172
\deca	134, 1028
\deci	66, 69, 134, 1018, 1063
\decibel	135, 1061
\DeclareBinaryPrefix	497
\DeclareCurrentRelease	5, 10
\DeclareExpandableDocumentCommand	274
\DeclareRelease	4, 6, 7, 8, 9
\DeclareSIPostPower	501
\DeclareSIPower	64
\DeclareSIPrefix	64
\DeclareSIPrePower	501
\DeclareSIQualifier	64
\DeclareSIUnit	64
\DeclareTranslation	40, 41, 42, 43
\def	323
\degree 61, 66, 104, 135, 168, 178, 333, 1074	
\degreeCelsius	79, 86, 135, 1039
\deka	134, 1028
dim commands:	
\dim_add:Nn	716
\dim_compare:nNnTF	218, 237, 438, 444, 497
\dim_compare_p:nNn	586
\dim_new:N	4, 4, 418
\dim_set:Nn 253, 614, 641, 665, 676, 704	
\dm	60, 170
\document	8
drop-exponent	38
drop-uncertainty	38
drop-zero-decimal	38
E	
\electronvolt	52, 53, 54, 55, 56, 57, 135, 1061
\else	102, 143, 145
else commands:	
\else:	337
\end	54, 75, 190
\endinput	23
\enquoe	71
\ensuremath	24, 24, 37, 53, 54, 59, 86, 130, 158, 165, 243, 247, 286, 340, 363, 379, 383
\ERROR	56
\eV	42, 171
evaluate-expression	38
\exa	134, 1028
\exbi	2, 176
exp commands:	
\exp_after:wN	25, 27, 68, 80, 109, 123, 124, 138, 140, 144, 147, 148, 167, 168, 170, 172, 174, 205, 233, 235, 245, 295, 336, 338, 339, 340, 341, 343, 358, 371, 372, 372, 468, 483, 517, 530, 559, 617, 639, 664, 670, 687, 703, 715, 909, 935, 945, 958, 1003, 1005, 1201, 1326, 1386, 1495, 1827
\exp_args:Nc	123, 274
\exp_args:Ne	42, 57, 1268, 1409
\exp_args:Nf	386, 675, 690, 1004, 1015, 1261, 1700
\exp_args:NNc	234
\exp_args:Nnno	1351
\exp_args:NNNV	25, 130, 239, 291, 522, 661, 783, 941
\exp_args:NNnx	241
\exp_args:NNV	16, 350
\exp_args:Nnx	479
\exp_args:NV	73, 108, 191, 243, 272, 1159, 1404
\exp_args:Nv	88
\exp_args:Nx	154, 278, 512
\exp_args:Nxx	96
\exp_last_unbraced:Nf	20, 167, 998, 1097, 1360
\exp_not:N	26, 28, 36, 54, 59, 76, 77, 80, 82, 84, 88, 98, 100, 101, 102, 105, 109, 111, 124, 156, 157, 173, 173, 175, 175, 181, 183, 197, 198, 200, 202, 202, 204, 207, 208, 209, 211, 212, 214, 215, 217, 218, 220, 222, 223, 225, 226, 227, 269, 270, 272, 276, 278, 280, 281, 281, 282, 282, 283, 284, 285, 285, 288, 288, 288, 289, 290, 290, 292, 292, 293, 293, 294, 295, 295, 297, 298, 300, 302, 303, 304, 306, 307, 308, 312, 313, 314, 362,

363, 365, 367, 377, 380, 387, 388, 527, 1004, 1006, 1576, 1607, 1650, 1765	fill-arc-minutes 11
\exp_not:n 108, 110, 111, 112, 113, 114, 161, 163, 201, 215, 216, 221, 228, 242, 267, 269, 270, 274, 279, 280, 286, 287, 288, 289, 292, 296, 297, 302, 303, 303, 304, 307, 312, 313, 314, 326, 327, 333, 340, 341, 346, 347, 350, 353, 354, 356, 356, 357, 359, 362, 365, 365, 366, 381, 386, 396, 398, 400, 436, 442, 443, 450, 451, 452, 475, 489, 491, 491, 536, 627, 630, 651, 678, 679, 680, 681, 682, 684, 687, 696, 704, 712, 722, 725, 733, 734, 746, 777, 782, 802, 803, 804, 805, 814, 815, 816, 820, 826, 827, 830, 835, 839, 840, 850, 857, 861, 862, 863, 871, 881, 885, 886, 887, 897, 899, 900, 902, 910, 911, 913, 916, 919, 922, 939, 947, 948, 949, 950, 951, 957, 958, 959, 963, 966, 967, 982, 983, 984, 1229, 1232, 1236, 1270, 1278, 1280, 1316, 1321, 1322, 1323, 1330, 1339, 1346, 1350, 1353, 1355, 1391, 1393, 1396, 1397, 1408, 1416, 1417, 1530, 1532, 1538, 1539, 1559, 1565, 1570, 1571, 1576, 1585, 1587, 1603, 1609, 1611, 1621, 1624, 1643, 1652, 1653, 1666, 1667, 1671, 1675, 1692, 1697, 1699, 1705, 1706, 1707, 1708, 1754, 1762, 1767, 1768, 1770, 1782	fill-arc-seconds 11 fixed-exponent 39 \fmol 14, 171 \fmtversion 44 font-command 137 \fontfamily 86, 254 \fontseries 86, 259 \fontshape 86, 264 \fontsize 375 forbid-literal-units 137 fp commands: \fp_add:Nn 771 \fp_compare:nNnTF 458, 572, 574, 622, 642 \fp_eval:n 42, 58, 67, 68, 70, 122, 457, 621, 1409 \fp_new:N 4, 4, 557, 559, 560 \fp_set:Nn 135, 136, 143, 149, 150, 157, 165, 172, 173, 202, 229, 606, 640, 668 \fp_use:N 203, 646 \fp_zero:N 135, 142, 156, 164, 182, 570 \c_one_fp 136, 143, 150, 574, 642 \l_tmpa_fp 131 \c_zero_fp 572, 622 \frac 130, 1127 fraction-command 137 free-standing-units 177 \fs 84, 170
	G
\expandafter 236 \ExplFileDate 1, 14 exponent-base 38 exponent-mode 38 exponent-product 38 expression 38 extract-mass-in-kilograms 137	\g 35, 170 \ge 98, 1866 \geq 1866 \GetTranslation 46, 47, 48 \GeV 42, 171 \gg 97, 1866 \GHz 8, 170 \gibi 2, 176 \giga 12, 47, 56, 80, 134, 1028 \GPa 77, 172 \gram 35, 36, 37, 38, 39, 40, 41, 134, 149, 413, 1009, 1017 \gray 135, 1039 group commands: \group_begin: 13, 16, 21, 33, 42, 74, 74, 86, 87, 96, 100, 105, 113, 117, 122, 131, 133, 133, 142, 151, 161, 169, 178, 178, 186, 194, 203, 206, 211, 215, 228, 232, 272, 287, 311, 347, 361, 378, 385, 427, 434, 474, 481, 509, 514, 523, 538, 547, 658, 780, 939, 1789
F	
\F 70, 172 \fam 27, 89, 185, 198, 199 \familydefault 86, 254 \farad 70, 71, 72, 135, 1039 \femto 15, 35, 71, 86, 134, 1018 \fF 70, 172 \fg 35, 170, 323 \fi 111, 143, 145 fi commands: \fi: 27, 33, 339 file commands: \file_if_exist:nTF 37 fill-arc-degrees 11	

<code>\l_keys_key_tl</code>	176, 178, 201, 212, 393, 407	<code>\mathord</code>	282, 340, 401, 1538, 1565, 1570, 1607, 1650, 1765
<code>\keys_set:nn</code>	2, 14, 19, 37, 44, 53, 78, 88, 94, 97, 98, 106, 114, 124, 135, 140, 143, 144, 150, 153, 158, 162, 171, 179, 187, 196, 204, 211, 266, 272, 287, 289, 306, 311, 379, 392, 395, 419, 447, 488, 496, 512, 515, 524, 540, 549, 733, 1122, 1849	<code>\mathrm</code>	36, 77, 86, 130, 201, 1124
<code>\l_keys_value_tl</code>	39, 47, 55, 87, 208, 218, 226, 260, 271, 279, 296, 304, 341, 390, 437, 478	<code>\mathsf</code>	93, 173, 187
<code>\kg</code>	35, 170	<code>\mathtt</code>	93, 174, 188
<code>\kHz</code>	8, 170	<code>\mathversion</code>	86, 87, 88, 158
<code>\kibi</code>	2, 176	<code>\mbox</code>	130, 373
<code>\kilo</code>	7, 10, 20, 26, 41, 45, 51, 54, 58, 67, 75, 78, 82, 134, 148, 660, 782, 1009, 1028	<code>\mdseries</code>	87
<code>\kilogram</code>	134, 134, 134, 1009	<code>\mebi</code>	2, 176
<code>\kJ</code>	42, 171	<code>\mega</code>	11, 46, 55, 76, 79, 83, 134, 1028
<code>\km</code>	60, 170	<code>\MessageBreak</code>	20
<code>\kmol</code>	14, 171	<code>\meter</code>	134, 166, 1009
<code>\kN</code>	73, 171	<code>\metre</code>	60, 61, 62, 63, 64, 65, 66, 67, 134, 135, 148, 166, 1009
<code>\kohm</code>	81, 172	<code>\MeV</code>	42, 171
<code>\kPa</code>	77, 172	<code>\mE</code>	42, 171
<code>\kV</code>	21, 171	<code>\mg</code>	35, 170
<code>\kW</code>	42, 171	<code>\MHz</code>	8, 170
<code>\kWh</code>	42, 172	<code>\mHz</code>	8, 170
L			
<code>\L</code>	27, 171	<code>\micro</code>	5, 18, 24, 30, 34, 38, 43, 49, 63, 89, 114, 116, 134, 1018
<code>\l</code>	27, 171	<code>\milli</code>	6, 9, 19, 25, 29, 33, 39, 44, 50, 53, 64, 74, 81, 90, 134, 1018
<code>\le</code>	96, 1866	minimum-decimal-digits	39
<code>\leq</code>	1866	minimum-integer-digits	39
list-exponents	20	<code>\minute</code>	135, 135, 1061
list-final-separator	20	<code>\mJ</code>	42, 171
list-pair-separator	20	<code>\mL</code>	27, 171
list-separator	20	<code>\ml</code>	27, 171
list-units	20	<code>\mm</code>	60, 170
<code>\liter</code>	31, 32, 33, 34, 135, 1061	<code>\mmol</code>	14, 171
<code>\litre</code>	27, 28, 29, 30, 135, 1061	<code>\MN</code>	73, 171
<code>\l</code>	95, 1866	<code>\mN</code>	73, 171
<code>\LRE</code>	314	mode	87
<code>\lumen</code>	135, 1039	mode commands:	
<code>\lux</code>	135, 1039	<code>\mode_if_math:TF</code>	101, 154, 197
M		<code>\mode_leave_vertical:</code>	73, 95, 104, 112, 121, 132, 141, 150, 160, 168, 177, 185, 193, 202, 513, 522, 537, 546
<code>\m</code>	60, 170	<code>\Mohm</code>	81, 172
<code>\mA</code>	2, 170	<code>\mohm</code>	81, 172
math commands:		<code>\mol</code>	14, 171
<code>\c_math_toggle_token</code> ...	345, 354, 357, 362, 386, 396, 400, 405, 414, 415	<code>\mole</code> ..	14, 15, 16, 17, 18, 19, 20, 134, 1009
<code>\mathchoice</code>	130, 164, 946	<code>\mp</code>	93, 285, 286, 1872
<code>\mathit</code>	130	<code>\MPa</code>	77, 172
		<code>\ms</code>	84, 170
		msg commands:	
		<code>\msg_error:nn</code>	435
		<code>\msg_error:nnn</code>	33, 160, 193, 481, 614, 629
		<code>\msg_error:nnnn</code>	353, 386, 400
		<code>\msg_info:nnnn</code>	13, 18, 284, 301, 318, 325, 333, 416
		<code>\msg_new:nnn</code>	4, 9, 35, 84, 244

\msg_new:nnnn	27, 1080, 1086, 1092, 1098, 1104, 1110, 1116, 1843
\msg_warning:nn	89
\msg_warning:nnn	24, 31, 228, 255
\mV	21, 171
\MW	42, 171
\mW	42, 171
N	
\N	73, 171
\nA	2, 170
\nano	4, 17, 23, 37, 62, 88, 134, 1018
negative-color	39
\neper	135, 1061
\newcolumntype	230, 257
\NewDocumentCommand	64, 68, 72, 76, 93, 102, 110, 119, 129, 139, 147, 157, 166, 175, 183, 191, 200, 210, 497, 501, 506, 511, 520, 534, 544
\newton	73, 74, 75, 76, 135, 1050
\ng	35, 170
\nm	60, 170
\nmol	14, 171
\nobreak	145, 185, 267, 529
\ns	84, 170
\num	102, 129, 280, 304
number-angle-product	12
number-color	87
number-mode	87
\numlist	129, 282, 305
\numproduct	129, 296
\numrange	175, 298, 306
\nV	21, 171
O	
\of	113, 136
\ohm	81, 82, 83, 94, 96, 135, 157, 1050
\Omega	101
output-close-uncertainty	39
output-decimal-marker	39
output-open-uncertainty	40
\over	138
overwrite-commands	177
P	
\Pa	77, 172
\pA	2, 170
\PackageError	17
parse-numbers	40
parse-units	137
\pascal	77, 78, 79, 80, 135, 1050
\pdfstringdefDisableCommands	154, 312, 320
\pebi	2, 176
peek commands:	
\peek_after:Nw	342
\peek_catcode_ignore_spaces:NTF	40, 133, 328
\per	104, 136, 137, 138, 142, 149, 149, 151, 152, 154, 426, 434, 440, 1093, 1096
per-mode	137
per-symbol	137
\percent	135, 1077
\peta	134, 1028
\pF	70, 172
\pg	35, 170
\pico	3, 16, 22, 36, 61, 72, 87, 134, 1018
\pm	60, 94, 139, 170, 283, 1698, 1872, 1873
\pmol	14, 171
\power	136, 136
prefix-mode	99
prg commands:	
\prg_new_conditional:Npnn	1197, 1803
\prg_new_protected_conditional:Npnn	11, 31, 1786
\prg_replicate:nn	190, 304, 305, 317, 322, 628, 651, 783, 808, 928, 1220, 1316, 1394, 1726
\prg_return_false:	19, 46, 1204, 1212, 1796, 1821
\prg_return_true:	18, 42, 1209, 1800, 1838
print-implicit-plus	40
print-unity-mantissa	40
print-zero-exponent	40
\ProcessKeysOptions	58
product-exponents	20
product-mode	20
product-phrase	20
product-symbol	20
prop commands:	
\prop_clear:N	336
\prop_count:N	636
\prop_get:NnN	613
\prop_get:NnNTF	451, 598, 604, 608, 610, 625, 638, 665, 756, 768
\prop_if_empty:NTF	188
\prop_if_in:NnTF	350, 394, 446, 480, 654, 656, 763
\prop_if_in_p:Nn	374
\prop_new:N	62, 63, 331
\prop_put:Nnn	58, 59, 357, 464, 469, 627, 645, 663
\prop_remove:N	643
\prop_remove:Nn	448, 460, 623
propagate-math-font	87
\providecommand	4, 5, 44
\ProvideDocumentCommand	71

<code>\ProvidesExplPackage</code>	25		R	
<code>\ps</code>	84 , 170		<code>\radian</code>	135 , 1050
<code>\pV</code>	21 , 171		<code>\raiseto</code>	116 , 136
Q			<code>range-exponents</code>	20
<code>\qty</code>	82 , 192 , 278, 304		<code>range-phrase</code>	21
<code>\qtylist</code>	129 , 289, 305		<code>range-units</code>	21
<code>\qtyproduct</code>	129 , 297		<code>\relax</code>	55, 76
<code>\qtyrange</code>	129 , 300, 306		<code>\renewcommand</code>	234
<code>qualifier-mode</code>	138		<code>\RequirePackage</code>	3, 4, 8, 12, 39, 57, 63, 63, 222
<code>qualifier-phrase</code>	138		<code>reset-math-version</code>	87
<code>quantity-product</code>	99		<code>reset-text-family</code>	87
quark commands:			<code>reset-text-series</code>	87
<code>\q_mark</code> ...	276, 285, 363, 364, 371, 372, 527, 552, 578, 581, 634, 637, 646, 649, 654, 657, 659, 661, 668, 671		<code>reset-text-shape</code>	87
<code>\q_nil</code>	46, 67, 95, 96, 99, 123, 124, 128, 129, 134, 195, 233, 263, 288, 371, 376, 467, 473, 487, 526, 528, 552, 581, 637, 649, 657, 658, 661, 662, 671, 1636, 1638, 1640, 1660, 1720, 1747		<code>retain-explicit-plus</code>	40
<code>\quark_if_nil:N</code>	1644, 1654, 1664, 1668, 1672, 1734		<code>retain-zero-uncertainty</code>	40
<code>\quark_if_nil:n</code>	296		<code>\rmdefault</code>	172
<code>\quark_if_recursion_tail_stop:N</code>	87, 146, 299, 339, 548, 1832		<code>\rmfamily</code>	87
<code>\quark_if_recursion_tail_stop:n</code>	250, 262, 386		<code>round-half</code>	40
<code>\quark_if_recursion_tail_stop_-do:Nn</code>	231, 331, 378, 400, 485, 502, 972, 981, 997, 1011, 1022, 1043, 1061, 1091, 1165, 1194, 1208, 1240, 1248, 1821		<code>round-minimum</code>	40
<code>\q_recursion_stop</code>	83, 127, 225, 237, 241, 258, 295, 326, 348, 374, 381, 382, 486, 503, 552, 967, 975, 986, 990, 1000, 1025, 1071, 1117, 1124, 1161, 1190, 1202, 1231, 1817, 1828		<code>round-mode</code>	40
<code>\q_recursion_tail</code>	82, 127, 225, 237, 241, 258, 293, 326, 374, 381, 967, 975, 986, 990, 1000, 1025, 1071, 1117, 1124, 1161, 1190, 1202, 1231, 1816, 1828		<code>round-pad</code>	40
<code>\q_stop</code>	68, 93, 96, 97, 99, 123, 125, 129, 130, 131, 134, 178, 180, 195, 246, 248, 263, 263, 270, 280, 281, 288, 289, 302, 307, 329, 330, 359, 363, 365, 371, 372, 373, 373, 376, 469, 473, 531, 552, 560, 562, 578, 581, 634, 637, 646, 649, 654, 658, 659, 662, 668, 671, 688, 692, 702, 704, 711, 714, 716, 718, 719, 729, 737, 741, 747, 749, 765, 767, 861, 874, 877, 885, 1412, 1413		<code>round-precision</code>	40
			S	
			<code>\s</code>	84 , 170
			<code>\scriptspace</code>	192, 215, 240
			<code>\second</code>	84, 85, 86, 87, 88, 89, 90, 134 , 135 , 1009
			<code>\selectfont</code>	86, 255, 260, 265, 375
			<code>separate-uncertainty</code>	40
			<code>separate-uncertainty-units</code>	99
			seq commands:	
			<code>\seq_clear:N</code>	93
			<code>\seq_const_from_clist:Nn</code>	302
			<code>\seq_count:N</code>	279
			<code>\seq_item:Nn</code>	290, 297, 302
			<code>\seq_map_function:NN</code>	91
			<code>\seq_map_indexed_function:NN</code> ..	312
			<code>\seq_map_inline:Nn</code>	53, 66, 179, 314, 324
			<code>\seq_new:N</code>	5, 21, 22
			<code>\seq_put_right:Nn</code>	29, 75, 116, 136, 154, 158
			<code>\seq_remove_all:Nn</code>	504, 509
			<code>\seq_use:Nnnn</code>	27
			<code>\seriesdefault</code>	86, 259
			<code>\sfdefault</code>	173
			<code>\shapedefault</code>	86, 264
			<code>\SI</code>	520
			<code>\si</code>	511
			<code>\siemens</code>	135 , 1050
			<code>\sievert</code>	135 , 1050
			<code>\SIlist</code>	534
			<code>\sim</code>	1866

<code>\SIrange</code>	534	<code>\siunitx_if_number_token:NTF</code>	...	
<code>\sisetup</code>	210	37 , 37 , 149 , 1803	
siunitx commands:					
<code>\siunitx_angle:n</code>	11 , 11 , 39 , 215	<code>\siunitx_if_number_token_p:N</code>	.	1803
<code>\siunitx_angle:nnn</code>	11 , 11 , 39 , 218 , 219		<code>\l_siunitx_list_separator_final_-</code>	<code>tl</code> 19 , 287 , 294 , 362 , 396
<code>\l_siunitx_bracket_ambiguous_-</code>	<code>bool</code> 38	<code>\l_siunitx_list_separator_pair_-</code>	<code>tl</code> 19 , 285 , 292 , 362 , 398
<code>\siunitx_cell_begin:w</code>	7 , 110 , 205 , 267		<code>\l_siunitx_list_separator_tl</code> 19 , 286 , 293 , 362 , 400
<code>\siunitx_cell_end:</code>		<code>\siunitx_number_adjust_exponent:Nn</code>	37 , 84 , 212 , 1399
.....	7 , 57 , 78 , 110 , 207 , 269		<code>\siunitx_number_adjust_exponent:nn</code>	37 , 1399
<code>\siunitx_command_create:</code>	...	37 , 177	<code>\l_siunitx_number_bracket_-</code>	<code>ambiguous_bool</code>
<code>\siunitx_complex_number:n</code>	188	63 , 240 , 1423 , 1428 , 1518	
<code>\siunitx_complex_quantity:nn</code>	..	197	<code>\l_siunitx_number_comparator_tl</code>	.	38
<code>\siunitx_compound_number:n</code>		<code>\l_siunitx_number_exponent_tl</code>	...	38
.....	19 , 84 , 380 , 429 , 476		<code>\siunitx_number_format:nN</code>	
<code>\siunitx_compound_quantity:nn</code>	14 , 37 , 115 , 153 , 728	
.....	19 , 230 , 387 , 436 , 483		<code>\l_siunitx_number_input_comparator_-</code>	<code>tl</code> 29 , 250 , 1809
<code>\siunitx_declare_power:NNn</code>		<code>\l_siunitx_number_input_decimal_-</code>	<code>tl</code>	28 , 38 , 40 , 346 , 387 , 390 , 405 , 1807
.....	40 , 66 , 132 , 503 , 508 , 1078 , 1079		<code>\l_siunitx_number_input_exponent_-</code>	<code>tl</code> 29 , 261 , 269 , 270 , 1811
<code>\siunitx_declare_power:NnN</code>	...	132	<code>\l_siunitx_number_input_sign_tl</code> 29 , 309 , 417 , 564 , 1814
<code>\siunitx_declare_prefix:Nn</code>	...	2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 49 , 132 , 132 , 499	<code>\siunitx_number_list:nn</code> 19 , 19 , 144 , 376
<code>\siunitx_declare_prefix:Nnn</code>	.	49 , 70 , 116 , 132 , 132 , 1018 , 1019 , 1020 , 1021 , 1022 , 1023 , 1024 , 1025 , 1026 , 1027 , 1028 , 1029 , 1030 , 1031 , 1032 , 1033 , 1034 , 1035 , 1036 , 1037 , 1038	<code>\siunitx_number_normalize_-</code>	<code>symbols:N</code> 37 , 77 , 126
<code>\siunitx_declare_qualifier:Nn</code> 64 , 74 , 132	<code>\siunitx_number_output:N</code> 22 , 37 , 37 , 115 , 135 , 743 , 1484
<code>\siunitx_declare_unit:Nn</code>	2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 10 , 11 , 11 , 12 , 13 , 14 , 15 , 16 , 17 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 25 , 26 , 27 , 28 , 29 , 30 , 31 , 32 , 33 , 34 , 35 , 36 , 37 , 38 , 39 , 40 , 41 , 42 , 43 , 44 , 45 , 46 , 47 , 48 , 49 , 50 , 51 , 52 , 53 , 53 , 54 , 55 , 56 , 57 , 58 , 60 , 61 , 62 , 63 , 64 , 65 , 66 , 67 , 68 , 69 , 70 , 70 , 71 , 72 , 73 , 74 , 75 , 76 , 77 , 78 , 79 , 79 , 80 , 81 , 82 , 83 , 84 , 85 , 86 , 86 , 87 , 88 , 89 , 90 , 96 , 132 , 157 , 1009 , 1010 , 1011 , 1012 , 1013 , 1014 , 1015 , 1016 , 1017 , 1039 , 1040 , 1041 , 1042 , 1043 , 1044 , 1045 , 1046 , 1047 , 1048 , 1049 , 1050 , 1051 , 1052 , 1053 , 1054 , 1055 , 1056 , 1057 , 1058 , 1059 , 1060 , 1061 , 1062 , 1063 , 1064 , 1065 , 1066 , 1067 , 1068 , 1069 , 1070 , 1071 , 1072 , 1073 , 1074 , 1075 , 1076 , 1077	<code>\siunitx_declare_unit:Nnn</code> 58 , 66 , 70 , 80 , 132 , 133 , 178
<code>\siunitx_format_number:nN</code>	11	<code>\siunitx_if_number:nTF</code>	37 , 1786
<code>\siunitx_if_number_p:n</code>	37	<code>\siunitx_number_product:n</code> 19 , 19 , 163 , 425

\siunitx_number_range:nn	siunitx internal commands:
..... 19, 19, 180, 472	__siunitx_angle:n 279, 328
\l_siunitx_number_sign_tl 38	__siunitx_angle:nnn 107, 212
\siunitx_print:nn	__siunitx_angle:w 328
..... 27, 47, 83, 84, 86, 86, 87,	__siunitx_angle_arc_convert:n .. 39
87, 87, 88, 99, 99, 116, 125, 126,	__siunitx_angle_arc_print:nnn ..
141, 146, 200, 207, 214, 241, 251, 47, 128, 166
253, 264, 270, 351, 358, 476, 479,	__siunitx_angle_arc_print_-
517, 528, 562, 596, 610, 619, 686, 729	auxi:nnn 166
\siunitx_print_match:n 86, 99	__siunitx_angle_arc_print_-
\siunitx_print_math:n .. 86, 102, 118	auxii:nw 178, 194
\l_siunitx_print_series_prop ... 88	__siunitx_angle_arc_print_-
\siunitx_print_text:n	auxii:w 166
... 68, 86, 88, 103, 103, 118, 156, 248	__siunitx_angle_arc_print_-
\siunitx_quantity_print:nn 139	auxiii:n 166
\siunitx_quantity:nn 40, 86, 99, 99, 531	__siunitx_angle_arc_print_-
\siunitx_quantity_list:nn	auxiv:NN 166
..... 19, 19, 136, 376, 541	__siunitx_angle_arc_print_-
\l_siunitx_quantity_prefix_mode_-	auxv:w 166
tl 9, 65, 164, 168, 242	__siunitx_angle_arc_print_-
\siunitx_quantity_print:nn 55, 99,	auxvi:n 166
104, 108, 121, 123, 127, 139, 149, 353	__siunitx_angle_arc_sign:nn ... 85
\siunitx_quantity_product:nn ...	__siunitx_angle_arc_sign:nnn 60, 85
..... 19, 154, 425	\l_siunitx_angle_astronomy_bool
\siunitx_quantity_range:nnn 6, 177
..... 19, 172, 472, 550	\l__siunitx_angle_degrees_tl ...
\l_siunitx_range_phrase_tl 44, 45, 46, 48, 81, 129
..... 20, 299, 301, 460, 491	__siunitx_angle_extract_-
\l_siunitx_unit_font_tl	sign:nnnnnnnn 85
..... 107, 110, 122,	\l_siunitx_angle_fill_degrees_-
132, 303, 313, 803, 815, 826, 839, 910	bool 6
\siunitx_unit_format:nN	\l_siunitx_angle_fill_minutes_-
..... 38, 46, 54, 79,	bool 6
81, 92, 99, 125, 130, 130, 131, 131,	\l_siunitx_angle_fill_seconds_-
131, 132, 144, 213, 219, 269, 516, 527	bool 6
\siunitx_unit_format_combine_-	\l_siunitx_angle_force_arc_bool
exponent:nnN 75, 131, 132, 175 6, 41
\siunitx_unit_format_extract_-	\l_siunitx_angle_force_decimal_-
prefixes:nN 80, 131, 132, 198	bool 6, 55
\siunitx_unit_format_multiply:nnN	\l_siunitx_angle_marker_box ...
..... 131, 132, 224	164, 204, 218, 222, 228, 230, 234, 239
\siunitx_unit_format_multiply_-	\l_siunitx_angle_minutes_tl 81, 130
combine_exponent:nnnN 131, 132, 183	\l_siunitx_angle_product_tl 6, 268
\siunitx_unit_format_multiply_-	\l_siunitx_angle_seconds_tl 81, 131
extract_prefixes:nnNN 131, 132, 204	\l_siunitx_angle_separator_tl 6, 186
\siunitx_unit_options_apply:n ...	__siunitx_angle_sign:nnnnnnn ... 85
..... 43, 79, 89, 97, 123, 132,	\l_siunitx_angle_sign_tl
133, 134, 152, 170, 195, 341, 539, 548 84, 95, 99, 108, 157
\siunitx_unit_pdfstring_context:	\l_siunitx_angle_tmp_bool
..... 133, 321, 322 3, 198, 199, 246
\siunitx_unit_power_set:NnN ... 143	\l_siunitx_angle_tmp_dim
\l_siunitx_unit_seq .. 22, 75, 91, 133 3, 231, 253, 255
\l_siunitx_unit_symbolic_seq 21,	\l_siunitx_angle_tmp_tl
29, 53, 66, 133, 133, 179, 324, 504, 509	... 3, 146, 147, 153, 213, 214, 269, 270

\l__siunitx_angle_unit_box	__siunitx_compound_format:nn
. 164 , 209 , 219 , 223 , 227 , 236 87 , 91 , 241
__siunitx_bookmark_cmd:Nn 272	__siunitx_compound_format:nnn 84
__siunitx_bookmark_cmd:Nnn	__siunitx_compound_format_-
. 272 , 278 , 279 ,	combine-exponent:n 160
280 , 281 , 282 , 289 , 296 , 297 , 298 , 300	__siunitx_compound_format_-
\c__siunitx_bookmark_seq 302 , 314	combine-exponent:nn 160
\l__siunitx_column_type_tl 48 , 262	__siunitx_compound_format_-
__siunitx_command_create: 37	combine-exponent_aux:n 160
__siunitx_command_create:N 37	__siunitx_compound_format_-
\l__siunitx_command_create_bool	extract-exponent:n 160
. 4 , 39	__siunitx_compound_format_-
\l__siunitx_command_optarg_bool	extract-exponent:nn 160
. 4 , 71 , 75	__siunitx_compound_format_-
\l__siunitx_command_overwrite_-	extract-exponent_aux:n 160
bool 4 , 64	__siunitx_compound_format_-
\l__siunitx_command_prespace_-	input:n 160
bool 4 , 80	__siunitx_compound_format_-
\l__siunitx_command_tmp_tl 3 , 82 , 84	input:nn 221
\l__siunitx_command_xspace_bool	__siunitx_compound_format_-
. 4 , 62 , 88	units:nn 108 , 160
\l__siunitx_compound_bracket_-	__siunitx_compound_parsed:n 118 , 151
close_tl 13 , 253 , 269	__siunitx_compound_print:N
\l__siunitx_compound_bracket_- 88 , 245 , 252 , 255 , 260
open_tl 13 , 251 , 267	__siunitx_compound_print:nnN 260
\l__siunitx_compound_count_int	__siunitx_compound_print:nnnN 260
. 11 , 308 , 331 , 336	__siunitx_compound_print_aux:n 260
\l__siunitx_compound_end_tl	__siunitx_compound_print_aux:nn 260
. 9 , 310 , 341	__siunitx_compound_print_-
\l__siunitx_compound_exp_-	number:n 88 , 252 , 255 , 260
bracket_bool 17 , 234 , 263	__siunitx_compound_print_-
\l__siunitx_compound_exp_-	quantity:n 245 , 256 , 352
combine_bool 17 , 110 , 189 , 214 , 236	__siunitx_compound_print_-
\l__siunitx_compound_exp_tl	separator:n
. 8 , 137 , 243 , 249 , 264 , 270 , 274 299 , 329 , 334 , 347 , 354 , 360
__siunitx_compound_extract_-	\l__siunitx_compound_separator_-
exp:nN 160	final_tl 17 , 334
__siunitx_compound_extract_-	\l__siunitx_compound_separator_-
exp:nnnnnnN 160	pair_tl 17 , 299
__siunitx_compound_extract_-	\l__siunitx_compound_separator_-
exponents: 112 , 120	text_bool 17 , 356
__siunitx_compound_extract_-	\l__siunitx_compound_separator_-
exponents:N 120	tl 17 , 329 , 347
__siunitx_compound_extract_-	\l__siunitx_compound_start_tl
exponents_auxi:w 120 9 , 309 , 326
__siunitx_compound_extract_-	\l__siunitx_compound_tmp_fp
exponents_auxii:nw 120 4 , 192 , 193 , 210 , 212
__siunitx_compound_extract_-	\l__siunitx_compound_tmp_seq
exponents_auxiii:nnnnnn 120 4 , 93 , 116 ,
\l__siunitx_compound_first_tl	136 , 154 , 158 , 279 , 290 , 297 , 302 , 313
. 7 , 107 ,	\l__siunitx_compound_tmp_tl
115 , 123 , 139 , 190 , 192 , 213 , 218 , 223 4 , 105 , 107 , 114 , 116 , 122 , 125 ,
__siunitx_compound_format:n 84	153 , 154 , 190 , 211 , 212 , 213 , 218 , 223

\l__siunitx_compound_unit_- bracket_bool 17 , 233 , 238 , 248	__siunitx_number_drop_uncertainty:nnnnnnn 940
\l__siunitx_compound_unit_power_- bool 21 , 56 , 62 , 68 , 74 , 80 , 162	\l__siunitx_number_drop_uncertainty_- bool 578 , 942
\l__siunitx_compound_unit_- repeat_bool 17 , 235 , 239 , 244	\l__siunitx_number_drop_zero_- decimal_bool 578 , 1383
\l__siunitx_compound_unit_tl 12 , 193 , 210 , 219 , 224 , 353	\l__siunitx_number_explicit_- plus_bool 29 , 318 , 569
__siunitx_compound_unparsed:n 101 , 151	__siunitx_number_exponent:NN 654 , 658
__siunitx_declare_column:Nnn .. 223	\l__siunitx_number_exponent_- base_tl 1425 , 1770
__siunitx_list_aux: 376	__siunitx_number_exponent_- engineering:nnnnnnn 658
__siunitx_list_count:n 345	__siunitx_number_exponent_- engineering:nnNw 658
__siunitx_list_count:w 345	__siunitx_number_exponent_- engineering_0:nnnn 658
\l__siunitx_list_exp_tl 362 , 394	__siunitx_number_exponent_- engineering_1:nnnn 658
\l__siunitx_list_units_tl .. 362 , 402	__siunitx_number_exponent_- engineering_2:nnnn 658
__siunitx_list_use:nnnnn 284 , 291 , 345	__siunitx_number_exponent_- engineering_aux:nnnnnnn 658
__siunitx_list_use_aux:nnnnn .. 345	__siunitx_number_exponent_- engineering_uncert:nn 658
__siunitx_list_use_auxi:nw 358 , 359 , 368	__siunitx_number_exponent_- engineering_uncert_S:nnn 658
__siunitx_list_use_auxii:w 345	__siunitx_number_exponent_- finalise:n 658 , 1180
__siunitx_list_use_auxiii:nnw .. 345	__siunitx_number_exponent_- fixed:nnnnnnn 658
__siunitx_load_check: 27	__siunitx_number_exponent_- fixed:nnnnnnnn 658
__siunitx_load_check:n ... 30 , 38 , 42	\l__siunitx_number_exponent_- fixed_int 578 , 676 , 684
__siunitx_number_adjust_exp:nn 1399	__siunitx_number_exponent_- input:nnnnnnn 658
__siunitx_number_adjust_- exp:nnnnnnnn 1399	\l__siunitx_number_exponent_- mode_tl 578 , 663 , 667 , 1140
__siunitx_number_adjust_exp:nNw 1399	\l__siunitx_number_exponent_- product_tl 1425 , 1767
\l__siunitx_number_arg_tl ... 48 , 50 , 69 , 119 , 125 , 126 , 127 , 246 , 253 , 256 , 272 , 287 , 299 , 373 , 560 , 566 , 574	__siunitx_number_exponent_- scientific:nnnnnnn 658
\l__siunitx_number_bracket_- close_tl 1419 , 1532 , 1587	__siunitx_number_exponent_- scientific:nnnnnnnn 658
\l__siunitx_number_bracket_- negative_bool 1425 , 1553	__siunitx_number_exponent_- scientific:nnnw 658
\l__siunitx_number_bracket_open_- tl 1419 , 1530 , 1585	__siunitx_number_exponent_- shift:nnn 658 , 1174
\l__siunitx_number_comparator_tl 70 , 252 , 255 , 356	__siunitx_number_exponent_- shift_down:nnn 658
__siunitx_number_digits:NN 642 , 906	__siunitx_number_exponent_- shift_down:nnnw 658
__siunitx_number_digits:Nn ... 906	
__siunitx_number_digits:nnnnnnn 906	
__siunitx_number_drop_exponent:NN 640 , 930	
__siunitx_number_drop_exponent:nnnnnnn 930	
\l__siunitx_number_drop_exponent_- bool 578 , 932	
__siunitx_number_drop_uncertainty:NN 638 , 940	

_siunitx_number_output_sign- brackets:w	1484	_siunitx_number_parse_exponent- auxiv:nn	259
_siunitx_number_output_sign- color:w	1484	_siunitx_number_parse_exponent- check:N	259
\l_siunitx_number_output- uncert_close_tl	1425, 1708	_siunitx_number_parse_exponent- cleanup:N	259
\l_siunitx_number_output- uncert_open_tl	1425, 1706	_siunitx_number_parse_exponent- cleanup:wN	345, 347
_siunitx_number_output- uncertainty:nnn	1484	_siunitx_number_parse_exponent- zero_test:N	259
_siunitx_number_output- uncertainty_S:nnnw	1484	_siunitx_number_parse_finalise:	164, 350
_siunitx_number_output- uncertainty_S:nnw	1484	_siunitx_number_parse_finalise:nw	350
_siunitx_number_output- uncertainty_S_aux:nnn	1484	_siunitx_number_parse_loop:	265, 305, 368
_siunitx_number_output- uncertainty_S_aux:nnnw	1716, 1732, 1739, 1746	_siunitx_number_parse_loop- after_decimal:NNN	368
_siunitx_number_output- uncertainty_S_aux:nnw .	1737, 1747	_siunitx_number_parse_loop- after_uncert:N	368
_siunitx_number_output- uncertainty_unaligned:n ...	1484	_siunitx_number_parse_loop- break:wN	368
\l_siunitx_number_outputted_tl	8, 21, 24, 26	_siunitx_number_parse_loop- first:N	368
_siunitx_number_parse:nN	108	_siunitx_number_parse_loop- first:NNN	371, 376, 466
_siunitx_number_parse_check:	129, 133	_siunitx_number_parse_loop- main:NNNNN	368
_siunitx_number_parse_combine- uncert:	151, 166	_siunitx_number_parse_loop- main_decimal:NN	368
_siunitx_number_parse_combine- uncert_auxi:nnnnnnnn	166	_siunitx_number_parse_loop- main_digit:NNNNN	368
_siunitx_number_parse_combine- uncert_auxii:nnnnn	166	_siunitx_number_parse_loop- main_end:NN	368
_siunitx_number_parse_combine- uncert_auxiii:nnnnnn	166	_siunitx_number_parse_loop- main_sign:NNN	368
_siunitx_number_parse_combine- uncert_auxiv:nnnn	166	_siunitx_number_parse_loop- main_store:NNN	368
_siunitx_number_parse_combine- uncert_auxv:w	166	_siunitx_number_parse_loop- main_uncert:NNN	368
_siunitx_number_parse_combine- uncert_auxvi:w	166	_siunitx_number_parse_loop- root_swap:NNwNN	368
_siunitx_number_parse_comparator:	128, 243	_siunitx_number_parse_loop- uncert:NNNNN	368
_siunitx_number_parse_comparator- aux:Nw	243	_siunitx_number_parse_sign:	257, 557
_siunitx_number_parse_exponent:	259, 576	_siunitx_number_parse_sign- aux:Nw	557
_siunitx_number_parse_exponent- auxi:w	259	\l_siunitx_number_parsed_tl	19, 20, 22, 46, 73, 118, 131, 140, 152, 154, 156, 170, 177, 212, 214, 264, 301, 304, 313, 323, 349, 352, 354, 357, 372, 555, 570, 571, 573, 575, 1792, 1793
_siunitx_number_parse_exponent- auxii:nn	259		
_siunitx_number_parse_exponent- auxiii:Nw	259		

\l_siunitx_number_partial_tl ...	_siunitx_number_round_if_-
. 75, 370, 432, 433, 436, 446, 471,	half:N 1197
472, 475, 479, 489, 509, 524, 535, 536	_siunitx_number_round_if_-
_siunitx_number_process:nnnnnnnNN	half:n 1197
..... 633	_siunitx_number_round_if_half_-
_siunitx_number_round:NN . 655, 952	p:n 1034, 1052, 1197
_siunitx_number_round:nnn	_siunitx_number_round_input:nn 964
964, 1252, 1291, 1301, 1362, 1371, 1376	\l_siunitx_number_round_min_tl .
_siunitx_number_round_auxi:nnnN 613, 621, 1319, 1327
..... 964	\l_siunitx_number_round_mode_tl
_siunitx_number_round_auxii:nnnN 578, 957, 1137, 1183
..... 964	_siunitx_number_round_none:nnnnnnn
_siunitx_number_round_auxiii:nnnN 952
..... 964	_siunitx_number_round_pad:nnn .
_siunitx_number_round_auxiv:nnN 1214, 1257, 1286
..... 964	\l_siunitx_number_round_pad_-
_siunitx_number_round_auxiv:nnnN	bool 578, 1219
..... 985, 999, 1009, 1015	_siunitx_number_round_places:nnnnnnn
_siunitx_number_round_auxv:nnN 964 1264
_siunitx_number_round_auxvi:nN 964	_siunitx_number_round_places_-
_siunitx_number_round_auxvi:nnN	decimal:nn 1264
..... 1024	_siunitx_number_round_places_-
_siunitx_number_round_auxvi:nnnN	end:nn 1179, 1264
..... 1018, 1041	_siunitx_number_round_places_-
_siunitx_number_round_auxvii:nnN	finalise:n 1264
..... 964	_siunitx_number_round_places_-
_siunitx_number_round_auxviii:nnN	finalise:nnnnn 1264
..... 964	_siunitx_number_round_places_-
_siunitx_number_round_engineering:nn	finalise:nnnnnnn 1264
..... 964	_siunitx_number_round_places_-
_siunitx_number_round_engineering:nnN	integer:nn 1264
..... 964	\l_siunitx_number_round_-
_siunitx_number_round_engineering:NNNNn	precision_int 578,
..... 964	1160, 1227, 1250, 1253, 1258, 1271,
_siunitx_number_round_figures:nnnnnnn	1284, 1287, 1294, 1304, 1345, 1363
..... 1223	_siunitx_number_round_scientific:nn
_siunitx_number_round_figures_- 1172
count:nnN 1223	_siunitx_number_round_scientificc:nn
_siunitx_number_round_figures_- 964
count:nnnN 1223	_siunitx_number_round_truncate:n
_siunitx_number_round_final:nn 964 964
_siunitx_number_round_final_-	_siunitx_number_round_truncate:nnN
decimal:nnw 964 964
_siunitx_number_round_final_-	_siunitx_number_round_truncate_-
integer:nnw 964	direct:n 964, 1378
_siunitx_number_round_final_-	_siunitx_number_round_uncertainty:nnn
output:nn 964 1341
_siunitx_number_round_final_-	_siunitx_number_round_uncertainty:nnnnn
shift:nn 964 1341
_siunitx_number_round_final_-	_siunitx_number_round_uncertainty:nnnnnnn
shift:Nw 964 1341
_siunitx_number_round_fixed:nn 964	_siunitx_number_set_round_-
\l_siunitx_number_round_half_-	min:n 601, 614
even_bool 578, 1032, 1050	

_siunitx_number_set_round_- min:nnnnnnn	614	_siunitx_option_table_format:n	331, 369, 371, 373, 375, 377, 379, 381
\l_siunitx_number_tight_bool	1425, 1569, 1764	_siunitx_option_table_sign-exponent:nnnnnnnn	331
\l_siunitx_number_tmp_tl	7, 147, 150, 268, 273, 279, 280, 288, 289, 616, 617	_siunitx_option_table_sign-mantissa:nnnnnnnn	331
_siunitx_number_token_auxi:NN	1806, 1819, 1823	_siunitx_print_convert_- series:n	118
_siunitx_number_token_auxii:NN	1822, 1825	_siunitx_print_extract_- series:Nw	118
_siunitx_number_token_auxiii:NN	1827, 1830, 1841	_siunitx_print_math_aux:Mn	118
\l_siunitx_number_uncert_- separate_bool	1425, 1517, 1695	_siunitx_print_math_auxi:n	118
\l_siunitx_number_uncert_- separator_tl	1425, 1705	_siunitx_print_math_auxii:n	118
\l_siunitx_number_unity_- mantissa_bool	1425, 1597, 1761	_siunitx_print_math_auxiii:n	118
\l_siunitx_number_valid_tl	1785	_siunitx_print_math_auxiv:n	118
\l_siunitx_number_validate_bool	76, 158, 1790	_siunitx_print_math_auxv:n	118
_siunitx_number_zero_decimal:NN	641, 1381	\l_siunitx_print_math_family_- bool	32, 168
_siunitx_number_zero_decimal:nnnnnnn	1381	\l_siunitx_print_math_font_bool	32, 183
\l_siunitx_number_zero_exponent_- bool	1425, 1522, 1756	_siunitx_print_math_script:n	118
\l_siunitx_number_zero_uncert_- bool	29, 233, 529	_siunitx_print_math_sub:n	118
_siunitx_option_deprecated:nn	11, 63, 69, 75, 164, 170, 182, 188, 197, 232, 238, 244, 250, 396, 402, 410, 425, 443, 449, 457, 463, 469, 483, 489	_siunitx_print_math_super:n	118
_siunitx_option_deprecated:nnn	11, 36, 44, 52, 84, 205, 215, 223, 257, 268, 276, 293, 387, 434, 475	_siunitx_print_math_text:n	118
_siunitx_option_removed:n	22, 30, 104, 128, 148, 157, 176, 178, 201, 212, 393, 407	_siunitx_print_math_version:nn	118
_siunitx_option_table_comparator:nnnnnnn	331	\l_siunitx_print_math_version_- bool	32, 145
_siunitx_option_table_comparator:nnnnnnnn	346	\l_siunitx_print_math_weight_- bool	32, 120
_siunitx_option_table_figures-decimal:nnnnnnn	331	\c_siunitx_print_mathrm_int	8, 199
_siunitx_option_table_figures-exponent:nnnnnnn	331	\c_siunitx_print_mathsf_int	8, 187
_siunitx_option_table_figures-integer:nnnnnnn	331	\c_siunitx_print_mathtt_int	8, 188
_siunitx_option_table_figures-uncertainty:nnnnnnn	331	\l_siunitx_print_number_color_- tl	32
		\l_siunitx_print_number_mode_tl	32
		_siunitx_print_replace_font:N	105, 208, 230, 280
		_siunitx_print_store_fam:n	8
		\l_siunitx_print_text_family_- bool	53, 252
		\l_siunitx_print_text_family_tl	32
		_siunitx_print_text_replace:N	248
		_siunitx_print_text_replace:n	248
		_siunitx_print_text_replace:NNn	248
		_siunitx_print_text_scripts:	248
		_siunitx_print_text_scripts:NnN	248
		_siunitx_print_text_scripts_- one:Nn	338, 345, 376
		_siunitx_print_text_scripts_- one:NnN	248
		_siunitx_print_text_scripts_- two:n	248

__siunitx_print_text_scripts_- two:nm	248	\l__siunitx_quantity_break_bool	9, 145
__siunitx_print_text_scripts_- two:NnNn	248	__siunitx_quantity_extract_- exp:nNN	73, 131
\l__siunitx_print_text_series_- bool	55, 257	__siunitx_quantity_extract_- exp:nnnnnnnnNN	131
\l__siunitx_print_text_series_tl	32	__siunitx_quantity_non_latin:n	157, 179
\l__siunitx_print_text_shape_- bool	57, 262	__siunitx_quantity_non_latin:nnnn	157
\l__siunitx_print_text_shape_tl	32	\l__siunitx_quantity_number_tl	38, 53, 56, 64, 66, 67, 68, 72, 74, 82, 85, 87, 91
__siunitx_print_text_sub:n	248	__siunitx_quantity_parsed:nn	40
__siunitx_print_text_super:n	248	__siunitx_quantity_parsed_- aux:nnnn	40
\l__siunitx_print_tmp_box	6, 22	__siunitx_quantity_parsed_- aux:nnnw	40
\l__siunitx_print_tmp_tl	6, 122, 124, 126, 207, 208, 209, 212, 215, 229, 230, 231, 240, 241, 243, 273, 274, 275, 312, 313, 314, 348, 349, 351	__siunitx_quantity_parsed_- aux:nnw	40
\l__siunitx_print_unit_color_tl	32	__siunitx_quantity_parsed_aux:w	40
\l__siunitx_print_unit_mode_tl	32	__siunitx_quantity_parsed_- combine-exponent:n	40
\l__siunitx_print_version_b_tl	71	__siunitx_quantity_parsed_- input:n	40
\l__siunitx_print_version_eb_tl	71	\l__siunitx_quantity_product_tl	9, 144
\l__siunitx_print_version_el_tl	71	\l__siunitx_quantity_tmp_fp	3, 74, 76, 81, 85
\l__siunitx_print_version_l_tl	71	\l__siunitx_quantity_tmp_tl	3
\l__siunitx_print_version_m_tl	71	\l__siunitx_quantity_uncert_- bracket_bool	9, 106
\l__siunitx_print_version_sb_tl	71	\l__siunitx_quantity_uncert_- repeat_bool	9, 119
\l__siunitx_print_version_sl_tl	71	\l__siunitx_quantity_unit_tl	38, 46, 47, 54, 56, 76, 81, 92, 104, 116, 122, 124, 127
\l__siunitx_print_version_ub_tl	71	__siunitx_range_aux:	472
\l__siunitx_print_version_ul_tl	71	\l__siunitx_range_exp_tl	460, 490
\c__siunitx_print_weight_b_tl	116	\l__siunitx_range_units_tl	460, 493
\c__siunitx_print_weight_c_tl	114	__siunitx_symbol_if_replace:Nn	31
\c__siunitx_print_weight_ecl_tl	114	__siunitx_symbol_if_replace:NnTF	31, 51, 56, 61, 94, 114
\c__siunitx_print_weight_ex_tl	114	__siunitx_symbol_non_latin:n	10, 34, 69, 75, 89, 108, 123, 158
\c__siunitx_print_weight_l_tl	116	__siunitx_symbol_non_latin:nnnn	10
\c__siunitx_print_weight_m_tl	116	\l__siunitx_symbol_tmpa_tl	3, 34, 35, 36, 39, 75, 76, 77, 80, 132, 134, 135, 137
\c__siunitx_print_weight_sc_tl	114	\l__siunitx_symbol_tmpb_tl	3, 38, 39, 79, 80, 136, 137
\c__siunitx_print_weight_sx_tl	114	\l__siunitx_table_after_box	416, 443, 445, 449, 456, 459, 470, 533, 546
\c__siunitx_print_weight_uc_tl	114	\l__siunitx_table_after_model_tl	271, 284, 321, 532
\c__siunitx_print_weight_ux_tl	114		
\c__siunitx_print_weight_x_tl	114		
__siunitx_product_aux:	425		
__siunitx_product_aux:n	425		
\l__siunitx_product_exp_tl	406, 449		
\l__siunitx_product_phrase_bool	406, 441, 454		
\l__siunitx_product_phrase_tl	406, 442		
\l__siunitx_product_symbol_tl	406, 443		
\l__siunitx_product_units_tl	406, 455		
\l__siunitx_quantity_bracket_- close_tl	5, 113		
\l__siunitx_quantity_bracket_- open_tl	5, 111		

<code>\l__siunitx_table_after_tl</code>	<code>\l__siunitx_table_column_width-</code>
. 101 , 112 , 119 , 147	dim 176 , 192
<code>\l__siunitx_table_align_after-</code>	<code>\l__siunitx_table_decimal_box</code> . . .
bool 419 , 536	. 229 , 239 , 241 , 244 , 250 , 341 , 356 ,
<code>__siunitx_table_align_auxi:nn</code> . 183	366 , 380 , 398 , 399 , 411 , 477 , 484 ,
<code>__siunitx_table_align_auxii:nn</code> 183	545 , 640 , 644 , 693 , 695 , 700 , 711 , 713
<code>\l__siunitx_table_align_before-</code>	<code>__siunitx_table_direct_begin:</code> . .
bool 419 , 555 , 600 12 , 323
<code>__siunitx_table_align_center:n</code> 183	<code>__siunitx_table_direct_begin:w</code> 323
<code>\l__siunitx_table_align_comparator-</code>	<code>__siunitx_table_direct_end:</code> 20 , 323
bool 419 , 585	<code>__siunitx_table_direct_format:</code> 323
<code>\l__siunitx_table_align_exponent-</code>	<code>__siunitx_table_direct_format:nnnnnnn</code>
bool 419 , 678 323
<code>__siunitx_table_align_left:n</code> . . 183	<code>__siunitx_table_direct_format:w</code> 323
<code>\l__siunitx_table_align_mode_tl</code> .	<code>__siunitx_table_direct_format-</code>
. 257 , 332 , 336 , 433	aux:w 372 , 375
<code>\l__siunitx_table_align_number-</code>	<code>__siunitx_table_direct_format-</code>
tl 257 , 408 , 541 , 725	end: 323
<code>__siunitx_table_align_right:n</code> . 183	<code>__siunitx_table_direct_format-</code>
<code>\l__siunitx_table_align_text_tl</code> .	switch: 323
. 216 , 226	<code>__siunitx_table_direct_marker:</code> 323
<code>\l__siunitx_table_align_uncertainty-</code>	<code>__siunitx_table_direct_marker-</code>
bool 419 , 667	end: 323
<code>\l__siunitx_table_auto_round-</code>	<code>__siunitx_table_direct_marker-</code>
bool 257 , 510	switch: 323
<code>\l__siunitx_table_before_box</code> . . .	<code>__siunitx_table_direct_none:</code> . . 323
. . . . 416 , 437 , 438 , 440 , 446 , 448 ,	<code>__siunitx_table_direct_none-</code>
452 , 457 , 463 , 496 , 497 , 499 , 502 ,	end: 323
506 , 543 , 567 , 575 , 621 , 623 , 630	<code>__siunitx_table_fil:</code> . . 231 , 245 ,
<code>\l__siunitx_table_before_model-</code>	252 , 343 , 382 , 392 , 406 , 451 , 460 ,
tl 269 , 282 , 328 , 495	505 , 539 , 560 , 574 , 595 , 608 , 629 , 701
<code>\l__siunitx_table_before_tl</code>	<code>\l__siunitx_table_fixed_width-</code>
. 101 , 110 , 114 , 117	bool 176 , 191
<code>\l__siunitx_table_carry_dim</code>	<code>\l__siunitx_table_format_tl</code> 281 ,
. . . . 418 , 534 , 537 , 641 , 696 , 704 , 716	290 , 292 , 293 , 296 , 336 , 340 , 344 , 518
<code>__siunitx_table_center_marker:</code> .	<code>__siunitx_table_generate-</code>
. 235 , 364 , 482	model:n 272 , 285
<code>__siunitx_table_cleanup-</code>	<code>__siunitx_table_generate-</code>
decimal:w 232 , 379	model:nnnnnnn 285 , 343
<code>__siunitx_table_collect_begin:</code> .	<code>__siunitx_table_generate_model-</code>
. 11 , 24	S:nw 285
<code>__siunitx_table_collect_begin:N</code> 120	<code>\l__siunitx_table_integer_box</code> . . .
<code>__siunitx_table_collect_begin:w</code> 24 229 , 238 , 242 , 249 ,
<code>__siunitx_table_collect_end:</code> 19 , 104	253 , 344 , 365 , 384 , 410 , 475 , 483 ,
<code>__siunitx_table_collect_group:n</code> 38	544 , 557 , 566 , 571 , 586 , 588 , 590 ,
<code>__siunitx_table_collect_loop:</code> . .	594 , 602 , 604 , 609 , 615 , 616 , 618 , 626
. 30 , 37 , 38	<code>\l__siunitx_table_model_tl</code>
<code>__siunitx_table_collect-</code> 270 , 272 , 281 , 301 , 371 , 526
search:NnTF 38	<code>\l__siunitx_table_number_tl</code>
<code>__siunitx_table_collect_search-</code> 101 , 111 , 113 , 118
aux:NNn 38	<code>__siunitx_table_print:nnn</code> . 116 , 432
<code>\l__siunitx_table_collect_tl</code> . . .	<code>__siunitx_table_print_format:nnn</code>
. . . . 23 , 26 , 46 , 60 , 81 , 106 , 107 , 109 432
<code>__siunitx_table_collect_token:N</code> 38	

_siunitx_table_print_format:nnnnn	\l_siunitx_table_tmp_dim
..... 517, 549	.. 3, 126, 614, 625, 665, 676, 706, 718
_siunitx_table_print_format_-	\l_siunitx_table_tmp_tl
after:N 432 3,
_siunitx_table_print_format_-	370, 373, 464, 465, 466, 467, 469,
auxi:w 530, 551	508, 521, 523, 524, 528, 531, 728, 729
_siunitx_table_print_format_-	_siunitx_tmp:w
auxii:w 578, 580 231, 233, 503, 504, 508, 509
_siunitx_table_print_format_-	\l_siunitx_tmp_tl
auxiii:w 634, 636 45, 115,
_siunitx_table_print_format_-	116, 125, 126, 188, 516, 517, 527, 528
auxiv:w 646, 648	\l_siunitx_unit_autofrac_bool
_siunitx_table_print_format_- 497, 504, 511, 518, 525, 532, 551, 929
auxv:w 652, 656	\l_siunitx_unit_bracket_bool
_siunitx_table_print_format_- 545, 583, 676, 746, 853, 874
auxvi:w 653, 660	\l_siunitx_unit_bracket_close_-
_siunitx_table_print_format_-	tl 546, 680, 805
auxvii:w 659, 668, 670	\l_siunitx_unit_bracket_open_tl
_siunitx_table_print_format_- 546, 678, 802
box:Nn 432	\l_siunitx_unit_combine_exp_fp
_siunitx_table_print_marker:nnn 135,
..... 432	142, 149, 156, 164, 172, 557, 572, 607
_siunitx_table_print_marker:w 432	\l_siunitx_unit_current_tl
_siunitx_table_print_marker_- 561,
aux:w 432	584, 722, 724, 740, 742, 782, 784,
_siunitx_table_print_marker_-	787, 795, 833, 840, 848, 900, 908, 911
auxi:w 432	\l_siunitx_unit_denominator_-
_siunitx_table_print_marker_-	bracket_bool 484, 872
auxii:w 432	\l_siunitx_unit_denominator_tl
_siunitx_table_print_marker_- 563, 568, 873, 918, 959, 968, 977, 984
auxiii:w 432	\l_siunitx_unit_font_bool
_siunitx_table_print_none:nnn 432 550, 585, 837, 843, 906, 913
_siunitx_table_print_text:n ...	\l_siunitx_unit_forbid_literal_-
..... 114, 223, 329	bool 192, 484
_siunitx_table_skip:n	_siunitx_unit_format:nnn
..... 171, 195, 197, 209, 211 132
_siunitx_table_split:nNNN	_siunitx_unit_format_aux:
..... 108, 122, 268 132
_siunitx_table_split_group:NNNn	_siunitx_unit_format_bracket:N
..... 122 674, 724, 742, 968
_siunitx_table_split_loop:NNN 122	_siunitx_unit_format_combine_-
_siunitx_table_split_tidy:N ...	exp: 573, 602
..... 128, 129, 160	_siunitx_unit_format_finalise:
_siunitx_table_split_tidy:Nn . 160 592, 916
_siunitx_table_split_token:NNNN	_siunitx_unit_format_finalise_-
..... 122	autofrac: 916
\l_siunitx_table_text_bool	_siunitx_unit_format_finalise_-
..... 6, 9, 16, 225	fraction: 934, 940, 953
\l_siunitx_table_tmp_box	_siunitx_unit_format_finalise_-
3, 339,	fractional: 916
342, 378, 381, 383, 385, 495, 503,	_siunitx_unit_format_finalise_-
532, 534, 554, 558, 570, 583, 591,	power: 916
605, 624, 639, 643, 664, 665, 666,	_siunitx_unit_format_finalise_-
675, 676, 677, 697, 702, 707, 714, 719	symbol: 933, 943, 962
	_siunitx_unit_format_font:
 686, 799, 811, 821, 852, 904
	_siunitx_unit_format_literal:n
 194, 197, 211

<code>_siunitx_unit_format_literal-</code>		<code>_siunitx_unit_format_qualifier:</code>	
<code>auxi:w</code>	211	788
<code>_siunitx_unit_format_literal-</code>		<code>_siunitx_unit_format_qualifier-</code>	
<code>auxii:n</code>	251 , 255	<code>bracket:</code>	788
<code>_siunitx_unit_format_literal-</code>		<code>_siunitx_unit_format_qualifier-</code>	
<code>auxii:w</code>	211	<code>combine:</code>	788
<code>_siunitx_unit_format_literal-</code>		<code>_siunitx_unit_format_qualifier-</code>	
<code>auxiii:w</code>	211	<code>phrase:</code>	788
<code>_siunitx_unit_format_literal-</code>		<code>_siunitx_unit_format_qualifier-</code>	
<code>auxiv:w</code>	211	<code>subscript:</code>	788
<code>_siunitx_unit_format_literal-</code>		<code>_siunitx_unit_format_special:</code>	831
<code>auxv:w</code>	211	<code>_siunitx_unit_format_unit:</code> ..	845
<code>_siunitx_unit_format_literal-</code>		<code>\l_siunitx_unit_formatted_tl</code> ...	
<code>subscript:</code>	211 131 , 155 , 181 , 201 , 232 , 240 ,	
<code>_siunitx_unit_format_literal-</code>		241 , 290 , 293 , 295 , 569 , 881 , 882 ,	
<code>superscript:</code>	211	927 , 928 , 942 , 944 , 948 , 949 , 950 ,	
<code>_siunitx_unit_format_literal-</code>		955 , 958 , 964 , 966 , 973 , 976 , 980 , 982	
<code>tilde:</code>	211	<code>\l_siunitx_unit_fraction-</code>	
<code>_siunitx_unit_format_mass_to-</code>		<code>function_tl</code>	484 , 957
<code>kilogram:</code>	579 , 650	<code>_siunitx_unit_if_symbolic:n</code> ...	11
<code>_siunitx_unit_format_multiply:</code>		<code>_siunitx_unit_if_symbolic:nTF</code> .	
.....	575 , 634	11 , 79 , 185
<code>_siunitx_unit_format_output:</code> ..		<code>_siunitx_unit_literal_power:nn</code>	
.....	590 , 850	43 , 117 , 209
<code>_siunitx_unit_format_output-</code>		<code>_siunitx_unit_literal_special:nN</code>	
<code>aux:</code>	850	108 , 111 , 210
<code>_siunitx_unit_format_output-</code>		<code>\l_siunitx_unit_mass_kilogram-</code>	
<code>aux:nn</code>	850	<code>bool</code>	122 , 578 , 759
<code>_siunitx_unit_format_output-</code>		<code>\c_siunitx_unit_math_subscript-</code>	
<code>denominator:</code>	850	<code>tl</code>	7 , 221 , 245 , 274 , 275 ,
<code>_siunitx_unit_format_parsed:</code> ..		278 , 279 , 283 , 284 , 286 , 287 , 310 , 824	
.....	189 , 565	<code>\l_siunitx_unit_multiple_fp</code> 136 ,	
<code>_siunitx_unit_format_parsed-</code>		143 , 150 , 157 , 165 , 173 , 560 , 574 , 641	
<code>aux:n</code>	565	<code>_siunitx_unit_non_latin:n</code>	
<code>_siunitx_unit_format_power:</code> ..	684	988 , 1024 , 1040 , 1051 , 1074 , 1075 , 1076	
<code>_siunitx_unit_format_power-</code>		<code>_siunitx_unit_non_latin:nnnn</code> .	988
<code>aux:wTF</code>	684	<code>\l_siunitx_unit_numerator_bool</code> .	
<code>_siunitx_unit_format_power-</code>		162 , 555 , 586 , 698 , 857
<code>negative:</code>	684	<code>\l_siunitx_unit_options_bool</code> ...	
<code>_siunitx_unit_format_power-</code>		88 , 91 , 102
<code>negative_aux:w</code>	684	<code>_siunitx_unit_parse:n</code>	187 , 334
<code>_siunitx_unit_format_power-</code>		<code>_siunitx_unit_parse_add:nnnn</code> ..	
<code>positive:</code>	684	347 ,
<code>_siunitx_unit_format_power-</code>		364 , 378 , 396 , 406 , 415 , 419 , 424 , 438	
<code>superscript:w</code>	715 , 718	<code>\l_siunitx_unit_parse_bool</code> 183 , 484	
<code>_siunitx_unit_format_power-</code>		<code>_siunitx_unit_parse_finalise:</code> .	
<code>superscript:</code>	684	345 , 474
<code>_siunitx_unit_format_prefix:</code> .	748	<code>_siunitx_unit_parse_finalise:n</code>	
<code>_siunitx_unit_format_prefix-</code>		344 , 443
<code>exp:</code>	748	<code>_siunitx_unit_parse_per:</code> .	106 , 429
<code>_siunitx_unit_format_prefix-</code>		<code>_siunitx_unit_parse_power:nnN</code> .	
<code>gram:</code>	748	44 , 47 , 118 , 121 , 361
<code>_siunitx_unit_format_prefix-</code>		<code>_siunitx_unit_parse_prefix:Nn</code> .	
<code>symbol:</code>	748	53 , 361

__siunitx_unit_parse_qualifier:nn	68, 115, 361
__siunitx_unit_parse_special:n	109, 112, 361
__siunitx_unit_parse_unit:Nn	81, 410
\l__siunitx_unit_parsed_prop	148, 188, 331, 336, 350, 357, 375, 394, 446, 448, 452, 460, 464, 469, 480, 598, 604, 608, 613, 623, 627, 636, 638, 643, 645, 654, 656, 663, 665, 763, 768
\l__siunitx_unit_parsing_bool	9, 34, 216, 323, 337, 659, 781
\l__siunitx_unit_part_tl	454, 456, 457, 458, 465, 561 , 599, 688, 701, 704, 706, 716, 757, 772, 778, 779, 787, 795, 800, 804, 812, 816, 822, 827, 835, 848
\l__siunitx_unit_per_bool	151, 331 , 338, 422, 433
\l__siunitx_unit_per_symbol_bool	498, 505, 512, 519, 526, 533, 551 , 880, 886, 932
\l__siunitx_unit_per_symbol_tl	484, 967
\l__siunitx_unit_position_int	155, 331 , 340, 343, 363, 370, 381, 393, 397, 407, 412, 416, 420, 425, 439, 479, 567, 571, 581, 587, 597, 762, 767
\l__siunitx_unit_powers_positive_bool	499, 506, 513, 520, 527, 534, 551 , 699, 920
\l__siunitx_unit_prefix_exp_bool	134, 141, 148, 155, 163, 171, 558 , 577, 750
\l__siunitx_unit_prefix_fp	182, 203, 559 , 570, 668, 669, 771
\l__siunitx_unit_prefixes_forward_prop	49, 610, 756
\l__siunitx_unit_prefixes_reverse_prop	49, 625
\l__siunitx_unit_product_tl	122, 252, 865, 876, 983
\l__siunitx_unit_qualifier_mode_tl	484, 556 , 793
\l__siunitx_unit_qualifier_phrase_tl	484, 814
\l__siunitx_unit_separator_tl	211
__siunitx_unit_set_symbolic:Nnn	23, 42, 45, 51, 66, 76, 104
__siunitx_unit_set_symbolic:Nnnn	23
__siunitx_unit_set_symbolic:Npnn	23, 107, 110, 113, 116, 119
\l__siunitx_unit_sticky_per_bool	151, 327 , 431
\l__siunitx_unit_test_bool	10, 14, 32, 339
\l__siunitx_unit_tmp_fp	4, 137, 151, 158, 174, 606, 621, 640, 642, 646
\l__siunitx_unit_tmp_int	4, 363, 365
\l__siunitx_unit_tmp_tl	4, 15, 17, 217, 218, 220, 230, 231, 233, 236, 349, 351, 358, 369, 375, 392, 394, 445, 446, 449, 450, 453, 461, 465, 470, 478, 480, 596, 599, 604, 605, 607, 608, 611, 613, 615, 616, 619, 620, 621, 622, 626, 627, 630, 638, 639, 641, 660, 662, 664, 666, 667, 669, 729, 743, 761, 763, 766, 769, 770, 772, 942, 947
\l__siunitx_unit_total_int	564, 567, 581, 652
\l__siunitx_unit_two_part_bool	500, 507, 514, 521, 528, 535, 551 , 869
skip commands:	
\skip_horizontal:N	240
\skip_horizontal:n	173, 215, 537
\c_zero_skip	174
\sp	159
space-before-unit	177
\SplitArgument	102
\SplitList	130, 139, 148, 158, 535
\square	135, 1078
\squared	136, 1078
\steradian	135, 1050
sticky-per	138
str commands:	
\str_case_e:nnTF	170
\str_if_eq:nnTF	39, 80, 151, 167, 195, 236, 242, 304, 333, 667, 699, 706, 716, 761, 771, 801, 903, 1063, 1137, 1183, 1322, 1335, 1348, 1392, 1417, 1549, 1606, 1649, 1833
\str_if_eq_p:nn	319, 444, 507, 624, 626, 648, 650, 1313, 1315, 1523, 1594, 1596, 1757, 1760
sys commands:	
\sys_if_engine luatex_p:	11, 106, 121, 158, 989
\sys_if_engine xetex:TF	303
\sys_if_engine xetex_p:	12, 107, 122, 159, 990
T	
table-align-comparator	110

table-align-exponent	110	\textsubscript	86, 324, 355
table-align-text-after	110	\textsuperscript	86, 329
table-align-text-before	110	\texttimes	143, 145
table-align-uncertainty	110	\the	233, 238
table-alignment	110	\THz	8, 170
table-alignment-mode	110	tight-spacing	40
table-auto-round	110	\times	14, 20, 26, 32, 136, 143, 145, 512, 1859
table-column-width	110	tl commands:	
table-fixed-width	110	\c_empty_tl	49, 50, 1659
table-format	111	\c_space_tl	275, 317, 336, 341, 380, 388
table-number-alignment	111	\tl_clear:N	
table-text-alignment	111	..	26, 95, 112, 118, 124, 125, 126,
\tablenum	200		143, 152, 154, 177, 178, 181, 232,
\tabularnewline	54, 56, 75, 77		243, 255, 301, 313, 323, 349, 370,
\tebi	2, 176		479, 554, 555, 568, 569, 575, 584, 636
\tera	13, 57, 134, 1028	\tl_clear_new:N	83
\tesla	135, 1050	\tl_const:Nn	7, 91, 115, 117
\TeV	42, 171	\tl_count:n	111, 165, 175,
TeX and L ^A T _E X 2 _ε commands:			182, 628, 651, 691, 927, 928, 1148,
\@ifl@t@r	14, 44		1284, 1287, 1294, 1304, 1316, 1363,
\@ifpackagelater	1		1372, 1377, 1394, 1620, 1631, 1701
\@ifpackageloaded	32, 42, 44, 63,	\tl_head:n	97, 214,
	66, 82, 83, 152, 202, 236, 248, 307, 310		269, 309, 1028, 1046, 1199, 1348, 1687
\@ifundefined	11	\tl_head:w	861, 885
\@maybe@unskip	72	\tl_if_blank:nTF	3, 17, 44, 103, 106,
\@temptokena	236, 238		136, 142, 152, 175, 183, 201, 300,
\f@family	170		300, 306, 307, 308, 322, 332, 334,
\f@series	123		336, 339, 341, 347, 369, 387, 483,
\FB@fg	323		490, 561, 651, 673, 685, 720, 739,
\m@th	331, 362		751, 757, 779, 795, 855, 869, 879,
\math@version	151		892, 983, 1093, 1225, 1266, 1369,
\NC@do	231, 232		1537, 1543, 1563, 1604, 1684, 1773
\NC@find	241	\tl_if_blank_p:n	4,
\NC@list	7, 232, 233		139, 143, 218, 219, 1344, 1519, 1595
\newcommand	178	\tl_if_empty:NnTF	86,
\protected@edef			99, 103, 107, 113, 119, 124, 127,
	15, 35, 76, 106, 119, 146, 230		135, 139, 147, 152, 156, 162, 233,
\sf@size	375		256, 261, 264, 293, 294, 304, 312,
\tab@setcr	73		352, 471, 524, 574, 635, 898, 918,
\z@	375		927, 973, 1319, 1403, 1494, 1551, 1793
tex commands:		\tl_if_empty:nTF	84, 693, 1499
\tex_cr:D	31	\tl_if_empty_p:N	249, 264, 432, 873, 881
\tex_hfil:D	231, 237	\tl_if_eq:NNTF	137
\tex_hss:D	257, 259	\tl_if_eq:nnTF	413
\tex_kern:D	174	\tl_if_exist:NnTF	95
\text	86, 96, 155, 250, 357	\tl_if_head_eq_charcode:nNTF	732
text-family-to-math	87	\tl_if_in:NnTF	5,
text-weight-to-math	88		56, 149, 250, 309, 340, 384, 390,
\textcolor	86,		402, 405, 412, 417, 487, 504, 519, 564
	86, 87, 88, 88, 111, 112, 130, 136, 1576	\tl_if_novalue:nTF	214, 217
\textminus	86, 288	\tl_if_single_token:nTF	93
\textmu	124	\tl_map_function:nN	101, 118
\textohm	109	\tl_map_inline:Nn	262, 270, 346, 387
\textpm	86, 284	\tl_map_inline:nn	54

\yotta	<i>134</i> , <u>1028</u>	\zepto	<i>134</i> , <u>1018</u>
	Z	\zetta	<i>134</i> , <u>1028</u>
\zebi	<u>2</u> , <i>176</i>		