

siunitx – A comprehensive (SI) units package*

Joseph Wright[†]

Released 2021-05-18

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	12
1	Formatting angles	12
1.1	Key-value options	12
2	siunitx-angle implementation	13
III	siunitx-compound – Compound numbers and quantities	20
1	siunitx-compound implementation	22
1.1	General mechanism	22
1.2	Lists	30
1.3	Products	31
1.4	Ranges	32
1.5	Standard settings for module options	33

*This file describes v3.0.1, last revised 2021-05-18.

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

IV	siunitx-locale – Localisation	35
1	siunitx-locale implementation	35
1.1	Locales	35
1.2	Localisation	36
V	siunitx-number – Parsing and formatting numbers	37
1	Formatting numbers	37
1.1	Key-value options	39
2	siunitx-number implementation	42
2.1	Initial set-up	42
2.2	Main formatting routine	42
2.3	Parsing numbers	43
2.4	Processing numbers	58
2.5	Number modification	76
2.6	Outputting parsed numbers	77
2.7	Miscellaneous tools	85
2.8	Messages	87
2.9	Standard settings for module options	87
VI	siunitx-print – Printing material with font control	89
1	Printing quantities	89
1.1	Key-value options	89
2	siunitx-print implementation	91
2.1	Initial set up	92
2.2	Printing routines	93
2.3	Standard settings for module options	101
VII	siunitx-quantity – Quantities	102
1	siunitx-quantity implementation	102
1.1	Initial set-up	103
1.2	Main formatting routine	103
1.3	Standard settings for module options	106
1.4	Adjustments to units	106
VIII	siunitx-symbol – Symbol-related settings	108
1	siunitx-symbol implementation	108
1.1	Bookmark definitions	111
IX	siunitx-table – Formatting numbers in tables	113

1	Numbers in tables	113
1.1	Key-value options	113
2	siunitx-table implementation	115
2.1	Interface functions	115
2.2	Collecting tokens	115
2.3	Separating collected material	117
2.4	Printing numbers in cells: spacing	119
2.5	Printing just text	121
2.6	Number alignment: core ideas	121
2.7	Directly printing without collection	124
2.8	Printing numbers in cells: main functions	126
2.9	Standard settings for module options	133
X	siunitx-unit – Parsing and formatting units	135
1	Formatting units	135
2	Defining symbolic units	137
3	Per-unit options	138
4	Units in (PDF) strings	138
5	Pre-defined symbolic unit components	138
5.1	Key-value options	141
6	siunitx-unit implementation	143
6.1	Initial set up	143
6.2	Defining symbolic unit	144
6.3	Applying unit options	146
6.4	Non-standard symbolic units	147
6.5	Main formatting routine	148
6.6	Formatting literal units	150
6.7	(PDF) String creation	153
6.8	Parsing symbolic units	153
6.9	Formatting parsed units	158
6.10	Non-Latin character support	170
6.11	Pre-defined unit components	171
6.12	Messages	173
6.13	Standard settings for module options	174
XI	siunitx-abbreviations – Abbreviations	175
1	siunitx-abbreviation implementation	177
XII	siunitx-binary – Binary units	181

1	siunitx-binary implementation	181
XIII	siunitx-command – Units as document command	182
1	Creating units as document commands	182
1.1	Key–value options	182
2	siunitx-command implementation	183
2.1	Options	183
2.2	Creation of unit document commands	184
2.3	Standard settings for module options	185
XIV	siunitx-emulation – Emulation	186
1	siunitx-emulation implementation	186
1.1	Load-time option	187
1.2	Angle options	187
1.3	Combination functions options	187
1.4	Command options	188
1.5	Print options	188
1.6	Symbol options	190
1.7	Number options	190
1.7.1	Table options	193
1.8	Unit options	195
1.9	Quantity units	196
1.10	Preamble commands	197
1.11	Document commands	198
1.12	Symbol commands	199
	Index	202

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}{2010-05-23}{siunitx-v2.sty}
7 \DeclareRelease{v2}{2010-05-23}{siunitx-v2.sty}
8 \DeclareCurrentRelease{}{2021-05-17}
```

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

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

```
12 \@ifl@t@r\ExplFileDate{2018-06-01}
13  {}
14  {%
15    \PackageError{siunitx}{Support package expl3 too old}
16    {%
17      You need to update your installation of the bundles 'l3kernel' and
18      'l3packages'.\MessageBreak
19      Loading~siunitx~will~abort!%
20    }%
21  \endinput
22  }%
```

Identify the package and give the over all version information.

```
23 \ProvidesExplPackage {siunitx} {2021-05-18} {3.0.1}
24  {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.

```

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

```

(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.

```

42 \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.

```

43 \tl_new:N \l__siunitx_tmp_tl

```

(End definition for \l__siunitx_tmp_tl.)

```

44 </init>

```

```

45 <*options>

```

1.5 Load time options

`\l__siunitx_column_type_tl`

```

46 \keys_define:nn { siunitx }
47   {
48     table-column-type .tl_set:N =
49       \l__siunitx_column_type_tl
50   }
51 \keys_set:nn { siunitx }
52   {
53     table-column-type = S
54   }

```

(End definition for \l__siunitx_column_type_tl.)

1.6 Option handling

```

55 \RequirePackage { l3keys2e }
56 \ProcessKeysOptions { siunitx }
57 \</options>

```

1.7 User interfaces

```

58 \<*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

```

59 \IfFormatAtLeastTF { 2020-10-01 }
60 { }
61 { \RequirePackage { xparse } }

```

1.7.1 Preamble commands

<pre> \DeclareSIPower \DeclareSIPrefix \DeclareSIQualifier \DeclareSIUnit </pre>	<pre> Pass data to the code layer. 62 \NewDocumentCommand \DeclareSIPower { +m +m m } 63 { 64 \siunitx_declare_power:Nnn #1 #2 {#3} 65 } 66 \NewDocumentCommand \DeclareSIPrefix { +m m m } 67 { 68 \siunitx_declare_prefix:Nnn #1 {#2} {#3} 69 } 70 \NewDocumentCommand \DeclareSIQualifier { +m m m } 71 { 72 \siunitx_declare_qualifier:Nn #1 {#2} 73 } 74 \NewDocumentCommand \DeclareSIUnit { o +m m } 75 { 76 \IfNoValueTF {#1} 77 { \siunitx_declare_unit:Nn #2 {#3} } 78 { \siunitx_declare_unit:Nnn #2 {#3} {#1} } 79 } </pre>
--	---

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

1.7.2 Document commands

```

\qty
80 \@ifpackageloaded { physics }
81 {
82   \msg_new:nnn { siunitx } { physics-pkg }
83   {
84     Detected~the~"physics"~package: \\\
85     Omitting~definition~of~\token_to_str:N \qty.
86   }
87   \msg_warning:nn { siunitx } { physics-pkg }
88   \use_none:nnnn
89 }

```

```

90 { }
91 \NewDocumentCommand \qty { 0 { } m } > { \TrimSpaces } m }
92 {
93   \mode_leave_vertical:
94   \group_begin:
95     \siunitx_unit_options_apply:n {#3}
96     \keys_set:nn { siunitx } {#1}
97     \siunitx_quantity:nn {#2} {#3}
98   \group_end:
99 }

```

(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 100 \NewDocumentCommand \ang { 0 { } } > { \SplitArgument { 2 } { ; } } m }
101 {
102   \mode_leave_vertical:
103   \group_begin:
104     \keys_set:nn { siunitx } {#1}
105     \_siunitx_angle:nnn #2
106   \group_end:
107 }
108 \NewDocumentCommand \num { 0 { } m }
109 {
110   \mode_leave_vertical:
111   \group_begin:
112     \keys_set:nn { siunitx } {#1}
113     \siunitx_number_format:nN {#2} \l__siunitx_tmp_tl
114     \siunitx_print_number:V \l__siunitx_tmp_tl
115   \group_end:
116 }
117 \NewDocumentCommand \unit { 0 { } } > { \TrimSpaces } m }
118 {
119   \mode_leave_vertical:
120   \group_begin:
121     \siunitx_unit_options_apply:n {#2}
122     \keys_set:nn { siunitx } {#1}
123     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
124     \siunitx_print_unit:V \l__siunitx_tmp_tl
125   \group_end:
126 }

```

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

\qtylist Interfaces for compound values.
\numlist 127 \NewDocumentCommand \qtylist
\qtyproduct 128 { 0 { } } > { \SplitList { ; } } m > { \TrimSpaces } m }
\numproduct 129 {
\qtyrange 130 \mode_leave_vertical:
131 \group_begin:
132 \siunitx_unit_options_apply:n {#3}
133 \keys_set:nn { siunitx } {#1}
134 \siunitx_quantity_list:nn {#2} {#3}


```

135     \group_end:
136   }
137 \NewDocumentCommand \numlist { O { } } > { \SplitList { ; } } m }
138 {
139   \mode_leave_vertical:
140   \group_begin:
141     \keys_set:nn { siunitx } {#1}
142     \siunitx_number_list:nn {#2}
143   \group_end:
144 }
145 \NewDocumentCommand \qtyproduct
146 { O { } } > { \SplitList { x } } m > { \TrimSpaces } m }
147 {
148   \mode_leave_vertical:
149   \group_begin:
150     \siunitx_unit_options_apply:n {#3}
151     \keys_set:nn { siunitx } {#1}
152     \siunitx_quantity_product:nn {#2} {#3}
153   \group_end:
154 }
155 \NewDocumentCommand \numproduct
156 { O { } } > { \SplitList { x } } m > { \TrimSpaces } m }
157 {
158   \mode_leave_vertical:
159   \group_begin:
160     \keys_set:nn { siunitx } {#1}
161     \siunitx_number_product:n {#2}
162   \group_end:
163 }
164 \NewDocumentCommand \qtyrange { O { } } m m > { \TrimSpaces } m }
165 {
166   \mode_leave_vertical:
167   \group_begin:
168     \siunitx_unit_options_apply:n {#4}
169     \keys_set:nn { siunitx } {#1}
170     \siunitx_quantity_range:nnn {#2} {#3} {#4}
171   \group_end:
172 }
173 \NewDocumentCommand \numrange { O { } } m m }
174 {
175   \mode_leave_vertical:
176   \group_begin:
177     \keys_set:nn { siunitx } {#1}
178     \siunitx_number_range:nn {#2} {#3}
179   \group_end:
180 }

```

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

\complexnum Interfaces for complex numbers.

```

\complexqty 181 \NewDocumentCommand \complexnum { O { } } m }
182 {
183   \mode_leave_vertical:
184   \group_begin:

```

```

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

```

(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`.

```

198 \NewDocumentCommand \tablenum { 0 { } m }
199 {
200   \mode_leave_vertical:
201   \group_begin:
202     \keys_set:nn { siunitx } {#1}
203     \siunitx_cell_begin:w
204     \ignorespaces #2
205     \siunitx_cell_end:
206   \group_end:
207 }

```

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

`\sisetup` A very thin wrapper.

```

208 \NewDocumentCommand \sisetup { m }
209 { \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.

```

210 \cs_new_protected:Npn \__siunitx_angle:nnn #1#2#3
211 {
212   \tl_if_novalue:nTF {#2}
213     { \siunitx_angle:n {#1} }
214     {
215       \tl_if_novalue:nTF {#3}
216         { \siunitx_angle:nnn {#1} {#2} { } }
217         { \siunitx_angle:nnn {#1} {#2} {#3} }
218     }
219 }

```

(End definition for `__siunitx_angle:nnn`.)

1.9 Table column

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

```
220 \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.

```
221 \cs_new_protected:Npn \_siunitx_declare_column:Nnn #1#2#3
222 {
223   \cs_if_exist:cT { NC@find@ #1 }
224   {
225     \cs_undefine:c { NC@find@ #1 }
226     \msg_warning:nnn { siunitx } { column-overwritten } {#1}
227   }
228   \newcolumnntype {#1} { }
229   \cs_set_protected:Npn \_siunitx_tmp:w \NC@do ##1##2 \NC@do #1
230   { \NC@list { \NC@do ##1 \NC@do #1 ##2 } }
231   \exp_after:wN \_siunitx_tmp:w \the \NC@list
232   \exp_args:NNc \renewcommand * { NC@rewrite@ #1 } [ 1 ] [ ]
233   {
234     \@temptokena \expandafter
235     {
236       \the \@temptokena
237       > {#2} c < {#3}
238     }
239     \NC@find
240   }
241 }
242 \msg_new:nnn { siunitx } { column-overwritten }
243 { Tabular~column~type~"#1"~overwritten~with~siunitx-definition. }
```

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

```
244 \AtBeginDocument
245 {
246   \@ifpackageloaded { mdwtab }
247   {
248     \cs_set_protected:Npn \_siunitx_declare_column:Nnn #1#2#3
249     {
250       \cs_if_exist:cT { NC@find@ #1 }
251       {
252         \cs_undefine:c { NC@find@ #1 }
253         \msg_warning:nnn { siunitx } { column-overwritten } {#1}
254       }
255       \newcolumnntype {#1} [ 1 ] [ ]
256       { > {#2} c < {#3} }
257     }
258   }
259 }
260 \tl_map_inline:Nn \l__siunitx_column_type_tl
261 {
```

```

262     \__siunitx_declare_column:Nnn #1
263     {
264         \keys_set:nn { siunitx } {##1}
265         \siunitx_cell_begin:w
266     }
267     { \siunitx_cell_end: }
268 }
269 }

```

(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.

```

270 \cs_new_protected:Npn \__siunitx_bookmark_cmd:Nnn #1#2#3
271 {
272     \exp_args:Nc \DeclareExpandableDocumentCommand
273     { \cs_to_str:N #1 \c_space_tl ( pdfstring ~ context ) }
274     {#2} {#3}
275 }
276 \__siunitx_bookmark_cmd:Nnn \qty { o m m } { #2 ~ #3 }
277 \__siunitx_bookmark_cmd:Nnn \ang { m } { \__siunitx_angle:n {#1} }
278 \__siunitx_bookmark_cmd:Nnn \num { o m } { #2 }
279 \__siunitx_bookmark_cmd:Nnn \unit { o m } { #2 }
280 \__siunitx_bookmark_cmd:Nnn \numlist { o m }
281 {
282     \__siunitx_list_use:nnVVV {#2} { }
283     \l_siunitx_list_separator_pair_tl
284     \l_siunitx_list_separator_tl
285     \l_siunitx_list_separator_final_tl
286 }
287 \__siunitx_bookmark_cmd:Nnn \qtylist { o m m }
288 {
289     \__siunitx_list_use:nnVVV {#2} {#3}
290     \l_siunitx_list_separator_pair_tl
291     \l_siunitx_list_separator_tl
292     \l_siunitx_list_separator_final_tl
293 }
294 \__siunitx_bookmark_cmd:Nnn \numproduct { o m } { }
295 \__siunitx_bookmark_cmd:Nnn \qtyproduct { o m m } { }
296 \__siunitx_bookmark_cmd:Nnn \numrange { o m m }
297 { #2 \tl_use:N \l_siunitx_range_phrase_tl #3 }
298 \__siunitx_bookmark_cmd:Nnn \qtyrange { o m m m }
299 { #2 ~ #4 \tl_use:N \l_siunitx_range_phrase_tl #3 ~ #4 }

```

(End definition for __siunitx_bookmark_cmd:Nn.)

We also need the v2 names.

```

300 \__siunitx_bookmark_cmd:Nnn \si { o m } { #2 }
301 \__siunitx_bookmark_cmd:Nnn \SI { o m 0 { } m } { #3 #2 ~ #4 }

```

```

302 \__siunitx_bookmark_cmd:Nnn \SIlist { o m m }
303 {
304   \__siunitx_list_use:nnVVV {#2} {#3}
305   \l_siunitx_list_separator_pair_tl
306   \l_siunitx_list_separator_tl
307   \l_siunitx_list_separator_final_tl
308 }
309 \__siunitx_bookmark_cmd:Nnn \SIrange { o m m m }
310 { #2 ~ #4 \tl_use:N \l_siunitx_range_phrase_tl #3 ~ #4 }

```

\c__siunitx_bookmark_seq Commands usable in bookmarks

```

311 \seq_const_from_clist:Nn \c__siunitx_bookmark_seq
312 {
313   \ang , \qty , \num , \unit ,
314   \numlist , \qtylist ,
315   \numrange , \qtyrange ,
316   \si , \SI , \SIlist , \SIrange
317 }

```

(End definition for \c__siunitx_bookmark_seq.)

Activate the document commands here: the unit macros are handled in siunitx-final.

```

318 \AtBeginDocument
319 {
320   \@ifpackageloaded { hyperref }
321   {
322     \pdfstringdefDisableCommands
323     {
324       \seq_map_inline:Nn \c__siunitx_bookmark_seq
325       {
326         \cs_set_eq:Nc #1
327         { \cs_to_str:N #1 \c_space_tl ( pdfstring ~ context ) }
328       }
329     }
330     \pdfstringdefDisableCommands
331     {
332       \siunitx_unit_pdfstring_context:
333       \cs_if_exist:NT \FB@fg { \def \fg { \FB@fg } }
334       \edef \H
335       {
336         \exp_not:c { PU-cmd }
337         \exp_not:N \H
338         \exp_not:c { PU \token_to_str:N \H }
339       }
340     }
341   }
342   { }
343 }

```

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

```

\__siunitx_angle:w
344 \cs_new:Npn \__siunitx_angle:n #1
345 { \__siunitx_angle:w #1 ; ; ; \q_stop }
346 \cs_new:Npn \__siunitx_angle:w #1 ; #2 ; #3 ; #4 \q_stop
347 {
348   \tl_if_blank:nF {#1}

```

```

349     { #1 \degree }
350   \tl_if_blank:nF {#2}
351   {
352     \tl_if_blank:nF {#1} { \c_space_tl }
353     #2 \arcminute
354   }
355   \tl_if_blank:nF {#3}
356   {
357     \tl_if_blank:nF {#1#2} { \c_space_tl }
358     #3 \arcsecond
359   }
360 }

```

(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 361 \cs_new:Npn \_siunitx_list_use:nnnnn #1#2#3#4#5
\_siunitx_list_use_auxi:w 362 {
\_siunitx_list_use_auxii:nnw 363   \tl_if_blank:nTF {#2}
\_siunitx_list_use_auxiii:nnw 364   { \_siunitx_list_use_aux:nnnnn {#1} { } }
\_siunitx_list_count:n 365   { \_siunitx_list_use_aux:nnnnn {#1} { ~ #2 } }
\_siunitx_list_count:w 366   {#3} {#4} {#5}
367 }
368 \cs_generate_variant:Nn \_siunitx_list_use:nnnnn { nnVVV }
369 \cs_new:Npn \_siunitx_list_use_aux:nnnnn #1#2#3#4#5
370 {
371   \int_case:nnF { \_siunitx_list_count:n {#1} }
372   {
373     { 0 } { }
374     { 1 } { \_siunitx_list_use_auxi:nw {#2} #1 ; ; { } }
375     { 2 } { \_siunitx_list_use_auxi:nw {#2} #1 ; {#3} }
376   }
377   {
378     \_siunitx_list_use_auxii:nnw {#2} { } #1 ;
379     \q_mark ; { \_siunitx_list_use_auxii:nnw {#2} {#4} }
380     \q_mark ; { \_siunitx_list_use_auxiii:nnw {#2} {#5} }
381     \q_stop { }
382   }
383 }
384 \cs_new:Npn \_siunitx_list_use_auxi:nw #1#2 ; #3 ; #4
385   { #2 #1 #4 #3 \tl_if_blank:nF {#3} {#1} }
386 \cs_new:Npn \_siunitx_list_use_auxii:nnw
387   #1#2#3 ; #4 ; #5 ; #6 \q_mark ; #7#8 \q_stop #9
388   { #7 {#4} ; {#5} ; #6 \q_mark ; {#7} #8 \q_stop { #9 #2 #3 #1 } }
389 \cs_new:Npn \_siunitx_list_use_auxiii:nnw #1#2#3 ; #4 \q_stop #5
390   { #5 #2 #3 #1 }
391 \cs_new:Npx \_siunitx_list_count:n #1
392 {
393   \exp_not:N \int_eval:n
394   {
395     0
396     \exp_not:N \_siunitx_list_count:w \c_space_tl
397     #1 \exp_not:n { ; \q_recursion_tail ; \q_recursion_stop }

```

```

398     }
399   }
400   \cs_new:Npx \__siunitx_list_count:w #1 ;
401   {
402     \exp_not:n { \exp_args:Nf \quark_if_recursion_tail_stop:n } {#1}
403     \exp_not:N \tl_if_blank:nF {#1} { + 1 }
404     \exp_not:N \__siunitx_list_count:w \c_space_tl
405   }

(End definition for \__siunitx_list_use:nnnnn and others.)

406 \end{interfaces}
407 \end{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`.

`angle-symbol-degree`
`angle-symbol-minute`
`angle-symbol-second`

`angle-symbol-degree = ⟨symbol⟩`

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

`angle-symbol-over-decimal`

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

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

`arc-separator`

`arc-separator = ⟨separator⟩`

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

`fill-angle-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-angle-minutes`

`fill-arc-minutes = true|false`

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

<hr/> <hr/>	<code>fill-arc-seconds = true false</code>
	Determines whether a missing seconds part is zero-filled when printing an arc. The standard setting is <code>false</code> .
<hr/> <hr/>	<code>number-angle-product = $\langle separator \rangle$</code>
	Inserted between the value of an angle and the unit (degree, minute or second component). The standard setting is <code>\,</code> .

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
\l__siunitx_angle_tmp_dim
\l__siunitx_angle_tmp_tl
```

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_symbol_degree_tl
\l__siunitx_angle_symbol_minute_tl
\l__siunitx_angle_symbol_second_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-degree .tl_set:N =
25     \l__siunitx_angle_symbol_degree_tl ,
26   angle-symbol-minute .tl_set:N =
27     \l__siunitx_angle_symbol_minute_tl ,
28   angle-symbol-second .tl_set:N =
29     \l__siunitx_angle_symbol_second_tl ,
30   angle-symbol-over-decimal .bool_set:N =
31     \l__siunitx_angle_astronomy_bool ,
```

```

32 angle-separator .tl_set:N =
33   \l__siunitx_angle_separator_tl ,
34 fill-angle-degrees .bool_set:N =
35   \l__siunitx_angle_fill_degrees_bool ,
36 fill-angle-minutes .bool_set:N =
37   \l__siunitx_angle_fill_minutes_bool ,
38 fill-angle-seconds .bool_set:N =
39   \l__siunitx_angle_fill_seconds_bool ,
40 number-angle-product .tl_set:N =
41   \l__siunitx_angle_product_tl
42 }
43 \bool_new:N \l__siunitx_angle_force_arc_bool
44 \bool_new:N \l__siunitx_angle_force_decimal_bool

```

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

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

```

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

```

```

80             - trunc((#1 - trunc(#1,0)) * 60,0)
81             )
82             * 60
83         }
84     }
85 }
86 }

```

(End definition for `\siunitx_angle:n`, `\siunitx_angle:nnn`, and `__siunitx_angle_arc_convert:n`. These functions are documented on page [12](#).)

```

\l__siunitx_angle_degrees_tl Space for formatting parsed numbers.
\l__siunitx_angle_minutes_tl 87 \tl_new:N \l__siunitx_angle_degrees_tl
\l__siunitx_angle_seconds_tl 88 \tl_new:N \l__siunitx_angle_minutes_tl
                             89 \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”.
                             90 \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 shuffling. That means doing processing of each number manually.

```

\__siunitx_angle_arc_sign:nn 91 \cs_new_protected:Npn \__siunitx_angle_arc_sign:nnn #1#2#3
\__siunitx_angle_extract_sign:nnnnnnnn 92 {
\__siunitx_angle_sign:nnnnnnnn 93   \group_begin:
94   \keys_set:nn { siunitx }
95   {
96     input-close-uncertainty = ,
97     input-exponent-markers = ,
98     input-open-uncertainty = ,
99     input-uncertainty-signs =
100   }
101   \tl_clear:N \l__siunitx_angle_sign_tl
102   \__siunitx_angle_arc_sign:nn {#1} { degrees }
103   \__siunitx_angle_arc_sign:nn {#2} { minutes }
104   \__siunitx_angle_arc_sign:nn {#3} { seconds }
105   \tl_if_empty:NF \l__siunitx_angle_sign_tl
106   {
107     \clist_map_inline:nn { degrees , minutes , seconds }
108     {
109       \tl_if_empty:cF { l__siunitx_angle_ ##1 _tl }
110       {
111         \tl_set:cx { l__siunitx_angle_ ##1 _tl }
112         {
113           { }
114           { \exp_not:V \l__siunitx_angle_sign_tl }
115           \exp_after:wN \exp_after:wN \exp_after:wN
116             \__siunitx_angle_sign:nnnnnnn
117             \cs:w l__siunitx_angle_ ##1 _tl \cs_end:
118         }
119       }
120     }
121   }
122   \clist_map_break:

```

```

120         }
121     }
122 }
123 \clist_map_inline:nn { degrees , minutes , seconds }
124 {
125     \tl_if_empty:cF { l__siunitx_angle_ ##1 _tl }
126     {
127         \tl_set:cx { l__siunitx_angle_ ##1 _tl }
128         {
129             \exp_args:Nc \siunitx_number_output:NN
130             { l__siunitx_angle_ ##1 _tl } \q_nil
131         }
132     }
133 }
134 \__siunitx_angle_arc_print:VVV
135 \l__siunitx_angle_degrees_tl
136 \l__siunitx_angle_minutes_tl
137 \l__siunitx_angle_seconds_tl
138 \group_end:
139 }
140 \cs_new_protected:Npn \__siunitx_angle_arc_sign:nn #1#2
141 {
142     \tl_if_blank:nTF {#1}
143     {
144         \bool_if:cTF { l__siunitx_angle_fill_ #2 _bool }
145         {
146             \tl_set:cn { l__siunitx_angle_ #2 _tl }
147             { { } { } { 0 } { } { } { } { } { 0 } }
148         }
149         { \tl_clear:c { l__siunitx_angle_ #2 _tl } }
150     }
151     {
152         \siunitx_number_parse:nN {#1} \l__siunitx_angle_tmp_tl
153         \exp_after:wN \__siunitx_angle_extract_sign:nnnnnnnn \l__siunitx_angle_tmp_tl {#2}
154     }
155 }
156 \cs_new_protected:Npn \__siunitx_angle_extract_sign:nnnnnnnn #1#2#3#4#5#6#7#8
157 {
158     \tl_if_blank:nTF {#2}
159     { \tl_set_eq:cN { l__siunitx_angle_ #8 _tl } \l__siunitx_angle_tmp_tl }
160     {
161         \tl_set:cn { l__siunitx_angle_ #8 _tl }
162         { {#1} { } {#3} {#4} {#5} {#6} {#7} }
163         \tl_set:Nn \l__siunitx_angle_sign_tl {#2}
164         \keys_set:nn { siunitx }
165         { input-comparators = , input-signs = }
166     }
167 }
168 \cs_new:Npn \__siunitx_angle_sign:nnnnnnnn #1#2#3#4#5#6#7
169 { \exp_not:n { {#3} {#4} {#5} {#6} {#7} } }

```

(End definition for __siunitx_angle_arc_sign:nnn and others.)

\l__siunitx_angle_marker_box For “astronomy style” angles.
\l__siunitx_angle_unit_box

```

170 \box_new:N \l__siunitx_angle_marker_box
171 \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_auxi:nVn
\__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.

```

172 \cs_new_protected:Npn \__siunitx_angle_arc_print:nnn #1#2#3
173 {
174   \__siunitx_angle_arc_print_auxi:nVn {#1}
175   \l__siunitx_angle_symbol_degree_tl {#2#3}
176   \__siunitx_angle_arc_print_auxi:nVn {#2}
177   \l__siunitx_angle_symbol_minute_tl {#3}
178   \__siunitx_angle_arc_print_auxi:nVn {#3}
179   \l__siunitx_angle_symbol_second_tl { }
180 }
181 \cs_generate_variant:Nn \__siunitx_angle_arc_print:nnn { VVV }
182 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxi:nnn #1#2#3
183 {
184   \tl_if_blank:nF {#1}
185   {
186     \bool_if:NTF \l__siunitx_angle_astronomy_bool
187     { \__siunitx_angle_arc_print_auxii:nw {#2} #1 \q_stop }
188     {
189       \__siunitx_angle_arc_print_auxv:w #1 \q_stop
190       \__siunitx_angle_arc_print_auxvi:n {#2}
191     }
192     \tl_if_blank:nF {#3}
193     {
194       \nobreak
195       \l__siunitx_angle_separator_tl
196     }
197   }
198 }
199 \cs_generate_variant:Nn \__siunitx_angle_arc_print_auxi:nnn { nV }
200 % \end{macrocode}
201 % To align the two parts of the astronomy-style marker, we need to allow
202 % for the |\scriptspace|.
203 % \begin{macrocode}
204 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxii:nw
205 #1#2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
206 {
207   \mode_if_math:TF
208   { \bool_set_true:N \l__siunitx_angle_tmp_bool }
209   { \bool_set_false:N \l__siunitx_angle_tmp_bool }
210   \siunitx_print_number:n {#2#3#4}
211   \tl_if_blank:nTF {#6}
212   { \__siunitx_angle_arc_print_auxvi:n {#1} }
213   {
214     \hbox_set:Nn \l__siunitx_angle_marker_box
215     {
216       \__siunitx_angle_arc_print_auxiii:n
217       { \siunitx_print_number:n {#5} }

```

```

218     }
219     \hbox_set:Nn \l__siunitx_angle_unit_box
220     {
221         \__siunitx_angle_arc_print_auxiii:n
222         {
223             \siunitx_unit_format:nN {#1} \l__siunitx_angle_tmp_tl
224             \siunitx_print_unit:V \l__siunitx_angle_tmp_tl
225             \skip_horizontal:n { -\scriptspace }
226         }
227     }
228     \dim_compare:nNnTF { \box_wd:N \l__siunitx_angle_marker_box } >
229     { \box_wd:N \l__siunitx_angle_unit_box }
230     {
231         \__siunitx_angle_arc_print_auxiv:NN
232         \l__siunitx_angle_marker_box
233         \l__siunitx_angle_unit_box
234     }
235     {
236         \__siunitx_angle_arc_print_auxiv:NN
237         \l__siunitx_angle_unit_box
238         \l__siunitx_angle_marker_box
239     }
240     \hbox_set_to_wd:Nnn \l__siunitx_angle_marker_box
241     \l__siunitx_angle_tmp_dim
242     {
243         \hbox_overlap_right:n
244         { \box_use_drop:N \l__siunitx_angle_marker_box }
245         \hbox_overlap_right:n
246         { \box_use_drop:N \l__siunitx_angle_unit_box }
247         \tex_hfil:D
248     }
249     \box_use:N \l__siunitx_angle_marker_box
250     \skip_horizontal:N \scriptspace
251     \siunitx_print_number:n {#6}
252 }
253 }
254 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxiii:n #1
255 {
256     \bool_if:NTF \l__siunitx_angle_tmp_bool
257     { \ensuremath }
258     { \use:n }
259     {#1}
260 }
261 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxiv:NN #1#2
262 {
263     \dim_set:Nn \l__siunitx_angle_tmp_dim { \box_wd:N #1 }
264     \hbox_set_to_wd:Nnn #2
265     \l__siunitx_angle_tmp_dim
266     {
267         \tex_hss:D
268         \hbox_unpack_drop:N #2
269         \tex_hss:D
270     }
271 }

```

```

272 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxv:w
273   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_stop
274   { \siunitx_print_number:n {#1#2#3#4#5} }
275 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxvi:n #1
276   {
277     \nobreak
278     \l__siunitx_angle_product_tl
279     \siunitx_unit_format:nN {#1} \l__siunitx_angle_tmp_tl
280     \siunitx_print_unit:V \l__siunitx_angle_tmp_tl
281   }

```

(End definition for __siunitx_angle_arc_print:nnn and others.)

```

282 \keys_set:nn { siunitx }
283   {
284     angle-mode           = input      ,
285     angle-symbol-degree  = \degree    ,
286     angle-symbol-minute  = \arcminute ,
287     angle-symbol-over-decimal = false  ,
288     angle-symbol-second  = \arcsecond ,
289     angle-separator      =            ,
290     fill-angle-degrees   = false      ,
291     fill-angle-minutes   = false      ,
292     fill-angle-seconds   = false      ,
293     number-angle-product =            =
294   }
295 \end{package}

```

Part III

siunitx-compound – Compound numbers and quantities

<hr/> <hr/> <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 <code>\siunitx_number_list:nn</code> , this function may semantically take any form
<hr/> <hr/> <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 <code>\siunitx_quantity_list:nn</code> , this function may semantically take any form
<hr/> <hr/> <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 $\langle balanced text \rangle$.
<hr/> <hr/> <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 $\langle balanced text \rangle$.
<hr/> <hr/> <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 $\langle balanced text \rangle$.
<hr/> <hr/> <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 $\langle balanced text \rangle$.
<hr/> <hr/> <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 $\langle end \rangle$.
<hr/> <hr/> <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 $\langle end \rangle$.
<hr/> <hr/> <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

range-phrase range-phrase = $\langle text \rangle$

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

Start the DocStrip guards.

1 $\langle *package \rangle$

1 siunitx-compound implementation

2 $\backslash cs_generate_variant:Nn \backslash keys_set:nn \{ nx \}$

1.1 General mechanism

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

3 $\langle @@=siunitx_compound \rangle$

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 $\backslash fp_new:N \backslash l_siunitx_compound_tmp_fp$
5 $\backslash seq_new:N \backslash l_siunitx_compound_tmp_seq$
6 $\backslash tl_new:N \backslash l_siunitx_compound_tmp_tl$

(End definition for $\backslash l_siunitx_compound_tmp_fp$, $\backslash l_siunitx_compound_tmp_seq$, and $\backslash l_siunitx_compound_tmp_tl$.)

The first number in the list in internal format.

7 $\backslash tl_new:N \backslash l_siunitx_compound_first_tl$

(End definition for $\backslash l_siunitx_compound_first_tl$.)

For storing the combined exponent, if present.

8 $\backslash tl_new:N \backslash l_siunitx_compound_exp_tl$

(End definition for $\backslash l_siunitx_compound_exp_tl$.)

Data on the end-of-list.

9 $\backslash tl_new:N \backslash l_siunitx_compound_start_tl$
10 $\backslash tl_new:N \backslash l_siunitx_compound_end_tl$

(End definition for $\backslash l_siunitx_compound_start_tl$ and $\backslash l_siunitx_compound_end_tl$.)

Data on the length-of-list.

11 $\backslash int_new:N \backslash l_siunitx_compound_count_int$

(End definition for $\backslash l_siunitx_compound_count_int$.)

$\backslash l_siunitx_compound_unit_tl$

12 $\backslash tl_new:N \backslash l_siunitx_compound_unit_tl$

(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_print_number:x
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 20.)

`__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
\_siunitx_compound_format_combine-exponent:n
\_siunitx_compound_format_extract-exponent:n
  \_siunitx_compound_format_input:n
\_siunitx_compound_format_combine-exponent:nn
\_siunitx_compound_format_extract-exponent:nn
\_siunitx_compound_format_combine-exponent_aux:n
\_siunitx_compound_format_extract-exponent_aux:n
  \_siunitx_compound_extract_exp:nN
\_siunitx_compound_extract_exp:nnnnnnN
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_number:V \l__siunitx_compound_bracket_open_tl
252     \__siunitx_compound_print:N \siunitx_print_number:x
253     \siunitx_print_number:V \l__siunitx_compound_bracket_close_tl
254   }
255   { \__siunitx_compound_print:N \siunitx_print_number:x }
256   \__siunitx_compound_print_quantity:n { }
257 }
258 \group_end:
259 }

```

(End definition for \siunitx_compound_quantity:nn. This function is documented on page 20.)

```

\__siunitx_compound_print:N We now need to know how many entries there are: the reason we don't use \seq_
  \__siunitx_compound_print:nnN use:Nnnn is that we want to be able to insert \siunitx_print...:n in a controlled
  \__siunitx_compound_print:xxN way.
  \__siunitx_compound_print:nnnN
  \__siunitx_compound_print_aux:n
  \__siunitx_compound_print_aux:nn
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_quantity:n #1
351 { \siunitx_quantity_print:nV {#1} \l__siunitx_compound_unit_tl }
352 \cs_new_protected:Npn \__siunitx_compound_print_separator:n #1
353 {
354     \bool_if:NTF \l__siunitx_compound_separator_text_bool
355     { #1 }
356     { \siunitx_print_number:n {#1} }
357 }
358 \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.

```

359 <@@=siunitx_list>

```

Options for products.

```

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

```

(End definition for `\l_siunitx_list_separator_tl` and others. These variables are documented on page 20.)

Simply recover the settings and use as a list.

```

\siunitx_number_list:nn
\siunitx_quantity_list:nn
\__siunitx_list_aux:
374 \cs_new_protected:Npn \siunitx_number_list:nn #1
375 {
376     \group_begin:
377     \__siunitx_list_aux:
378     \siunitx_compound_number:n {#1}
379     \group_end:
380 }

```

```

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

```

(End definition for `\siunitx_number_list:nn`, `\siunitx_quantity_list:nn`, and `__siunitx_list_aux:`. These functions are documented on page 20.)

1.3 Products

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

```

403 <@@=siunitx_product>

```

```

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

```

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

`\siunitx_number_product:n` Simply recover the settings and use as a list.

`\siunitx_quantity_product:nn`

```

423 \cs_new_protected:Npn \siunitx_number_product:n #1
424 {
425   \group_begin:
426     \__siunitx_product_aux:
427     \siunitx_compound_number:n {#1}
428   \group_end:
429 }
430 \cs_new_protected:Npn \siunitx_quantity_product:nn #1#2
431 {
432   \group_begin:
433     \__siunitx_product_aux:
434     \siunitx_compound_quantity:nn {#1} {#2}
435   \group_end:
436 }
437 \cs_new_protected:Npn \__siunitx_product_aux:
438 {
439   \bool_if:NTF \l__siunitx_product_phrase_bool
440   { \__siunitx_product_aux:x { \exp_not:N \l__siunitx_product_phrase_tl } }
441   { \__siunitx_product_aux:x { { } \exp_not:N \l__siunitx_product_symbol_tl { } } }
442 }
443 \cs_new_protected:Npn \__siunitx_product_aux:n #1
444 {
445   \keys_set:nx { siunitx }
446   {
447     compound-exponents      = \l__siunitx_product_exp_tl ,
448     compound-final-separator = { \exp_not:n {#1} } ,
449     compound-pair-separator  = { \exp_not:n {#1} } ,
450     compound-separator       = { \exp_not:n {#1} } ,
451     compound-separator-mode  =
452       \bool_if:NTF \l__siunitx_product_phrase_bool { text } { number } ,
453     compound-units           = \l__siunitx_product_units_tl
454   }
455 }
456 \cs_generate_variant:Nn \__siunitx_product_aux:n { x }

```

(End definition for `\siunitx_number_product:n` and others. These functions are documented on page 20.)

1.4 Ranges

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

```

457 <@@=siunitx_range>

```

Options for products.

```

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

```

```

466   range-units .choices:nn =
467     { bracket , repeat , single }
468     { \tl_set_eq:NN \l__siunitx_range_units_tl \l_keys_choice_tl }
469   }

```

(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 21.)

`\siunitx_number_range:nn`
`\siunitx_quantity_range:nnn`
`__siunitx_range_aux:`

Simply recover the settings and use as a list.

```

470 \cs_new_protected:Npn \siunitx_number_range:nn #1#2
471 {
472   \group_begin:
473     \__siunitx_range_aux:
474     \siunitx_compound_number:n { {#1} {#2} }
475   \group_end:
476 }
477 \cs_new_protected:Npn \siunitx_quantity_range:nnn #1#2#3
478 {
479   \group_begin:
480     \__siunitx_range_aux:
481     \siunitx_compound_quantity:nn { {#1} {#2} } {#3}
482   \group_end:
483 }
484 \cs_new_protected:Npn \__siunitx_range_aux:
485 {
486   \keys_set:nx { siunitx }
487   {
488     compound-exponents      = \l__siunitx_range_exp_tl ,
489     compound-pair-separator = { \exp_not:V \l_siunitx_range_phrase_tl } ,
490     compound-separator-mode = text ,
491     compound-units          = \l__siunitx_range_units_tl
492   }
493 }

```

(End definition for `\siunitx_number_range:nn`, `\siunitx_quantity_range:nnn`, and `__siunitx_range_aux:`. These functions are documented on page 20.)

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.

```

494 \keys_set:nn { siunitx }
495 {
496   compound-exponents      = individual ,
497   compound-final-separator =
498   {
499     \ifmmode \ \else \space \fi
500     \text { and }
501     \ifmmode \ \else \space \fi
502   } ,
503   compound-pair-separator =
504   {
505     \ifmmode \ \else \space \fi
506     \text { and }

```

```

507     \ifmmode \ \else \space \fi
508   } ,
509   compound-separator      =
510   { , \ifmmode \ \else \space \fi } ,
511   compound-separator-mode = text      ,
512   compound-units          = repeat    ,
513   list-exponents         = individual ,
514   list-final-separator   =
515   {
516     \ifmmode \ \else \space \fi
517     \text { and }
518     \ifmmode \ \else \space \fi
519   } ,
520   list-pair-separator     =
521   {
522     \ifmmode \ \else \space \fi
523     \text { and }
524     \ifmmode \ \else \space \fi
525   } ,
526   list-separator          =
527   { , \ifmmode \ \else \space \fi } ,
528   list-units              = repeat    ,
529   product-exponents      = individual ,
530   product-mode            = symbol    ,
531   product-phrase         =
532   {
533     \ifmmode \ \else \space \fi
534     \text { by }
535     \ifmmode \ \else \space \fi
536   } ,
537   product-symbol          = \times    ,
538   product-units           = repeat    ,
539   range-exponents        = individual ,
540   range-phrase            =
541   {
542     \ifmmode \ \else \space \fi
543     \text { to }
544     \ifmmode \ \else \space \fi
545   } ,
546   range-units             = repeat
547 }
548 \end{package}

```

Part IV

siunitx-locale — Localisation

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

locale

```
locale = <locale>
```

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

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) } { à }
42     \DeclareTranslation { German } { to~(numerical~range) } { bis }
43     \DeclareTranslation { Spanish } { to~(numerical~range) } { a }
44     \keys_set:nn { siunitx }
45     {
46         list-final-separator =
47         {
48             \ifmmode \ \else \space \fi
49             \text { \GetTranslation { and } }
50             \ifmmode \ \else \space \fi
51         } ,
52         list-pair-separator =
53         {
54             \ifmmode \ \else \space \fi
55             \text { \GetTranslation { and } }
56             \ifmmode \ \else \space \fi
57         } ,
58         range-phrase =
59         {
60             \ifmmode \ \else \space \fi
61             \text { \GetTranslation { to~(numerical~range) } }
62             \ifmmode \ \else \space \fi
63         }
64     }
65 }

```


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 = $\langle base \rangle$`

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

`exponent-product` `exponent-product = $\langle symbol \rangle$`

`expression` `expression = $\langle expression \rangle$`

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

2 siunitx-number implementation

Start the DocStrip guards.

```
1 <*package>
```

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

```
2 <@@=siunitx_number>
```

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 39.)

`\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 39.)

`\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 38.)

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:

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

where 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 $\langle \text{integer} \rangle$ and $\langle \text{exponent} \rangle$ parts.

A non-empty $\langle \text{uncertainty} \rangle$ 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 39.)

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

```

\l_siunitx_number_input_uncert_close_tl
\l_siunitx_number_input_comparator_tl
\l_siunitx_number_input_digit_tl
\l_siunitx_number_input_exponent_tl
\l_siunitx_number_input_ignore_tl
\l_siunitx_number_input_uncert_open_tl
\l_siunitx_number_input_sign_tl
\l_siunitx_number_input_uncert_sign_tl
\l_siunitx_number_explicit_plus_bool
\l_siunitx_number_zero_uncert_bool
\l_siunitx_number_expression:n
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 38.)

```

\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 37.)

```

\__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:nnx { 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:
\__siunitx_number_parse_combine_uncert_auxi:nnnnnnnn
\__siunitx_number_parse_combine_uncert_auxii:nnnnnn
\__siunitx_number_parse_combine_uncert_auxiii:fnnnn
\__siunitx_number_parse_combine_uncert_auxiiii:nnnnnn
\__siunitx_number_parse_combine_uncert_auxv:nnnn
\__siunitx_number_parse_combine_uncert_auxvi:w
\__siunitx_number_parse_combine_uncert_auxvii: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         \bool_lazy_and:nnTF
218         { \tl_if_blank_p:n {#2} }
219         { ! \tl_if_blank_p:n {#4} }
220         { 0 }
221         { \exp_not:n {#2} }
222       }
223       {
224         \__siunitx_number_parse_combine_uncert_auxv:w #3#4
225         \q_recursion_tail \q_recursion_stop
226       }
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: 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:NNNNN
    \_siunitx_number_parse_loop_main_end:NN
    siunitx_number_parse_loop_main_digit:NNNNN
    \_siunitx_number_parse_loop_main_decimal:NN
    \_siunitx_number_parse_loop_main_uncert:NNN
    \_siunitx_number_parse_loop_main_sign:NNN
    \_siunitx_number_parse_loop_main_store:NNN
    siunitx_number_parse_loop_after_decimal:NNN
    \_siunitx_number_parse_loop_root_swap:NNwNN
    \_siunitx_number_parse_loop_break:wN

```

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

```

368 \cs_new_protected:Npn \_siunitx_number_parse_loop:
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:NNNN #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:NNNN #1 #2 \c_true_bool #4
448     }
449     { \__siunitx_number_parse_loop_main:NNNN #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_uncert:NN #1
460 }

```

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.

```

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

```

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.

```

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

```

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.

```

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

```

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.

```

499 \cs_new_protected:Npn \__siunitx_number_parse_loop_break:wN
500 #1 \q_recursion_stop
501 {
502   \tl_clear:N \l__siunitx_number_flex_tl
503   \tl_clear:N \l__siunitx_number_parsed_tl
504 }

```

(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.

```

505 \cs_new_protected:Npn \__siunitx_number_parse_sign:

```

```

506 {
507   \exp_after:wN \_siunitx_number_parse_sign_aux:Nw
508   \l__siunitx_number_arg_tl \q_stop
509 }
510 \cs_new_protected:Npn \_siunitx_number_parse_sign_aux:Nw #1#2 \q_stop
511 {
512   \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#1}
513   {
514     \tl_set:Nn \l__siunitx_number_arg_tl {#2}
515     \bool_lazy_and:nnTF
516     { \token_if_eq_charcode_p:NN #1 + }
517     { ! \l__siunitx_number_explicit_plus_bool }
518     { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
519     { \tl_set:Nn \l__siunitx_number_parsed_tl { {#1} } }
520   }
521   { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
522   \tl_if_empty:NnTF \l__siunitx_number_arg_tl
523   { \tl_clear:N \l__siunitx_number_parsed_tl }
524   { \_siunitx_number_parse_exponent: }
525 }

```

(End definition for _siunitx_number_parse_sign: and _siunitx_number_parse_sign_aux:Nw.)

```

\_siunitx_number_parse_uncert:NN
\_siunitx_number_parse_uncert:NNNN
\_siunitx_number_parse_uncert_auxi:NN
\_siunitx_number_parse_uncert_auxii:NN
\_siunitx_number_parse_uncert_auxii:N
\_siunitx_number_parse_uncert_marker:N
\_siunitx_number_parse_uncert_after:N

```

Parsing a combined uncertainty has a very restricted range of allowed tokens. A closing uncertainty token in the first place is an error, so we filter that out explicitly. After that, we check for digits, which require checking for significant digits. The non-digit function is separate to make the flow clearer.

```

526 \cs_new_protected:Npn \_siunitx_number_parse_uncert:NN #1#2
527 {
528   \quark_if_recursion_tail_stop_do:Nn #2
529   { \_siunitx_number_parse_loop_break:wN \q_recursion_stop }
530   \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#2}
531   { \_siunitx_number_parse_loop_break:wN }
532   {
533     \_siunitx_number_parse_uncert:NNNN
534     #1 \c_false_bool \_siunitx_number_parse_uncert_auxi:NN #2
535   }
536 }

```

Deal with digits: a simple question of whether they are significant.

```

537 \cs_new_protected:Npn \_siunitx_number_parse_uncert:NNNN #1#2#3#4
538 {
539   \quark_if_recursion_tail_stop_do:Nn #4
540   { \_siunitx_number_parse_loop_break:wN \q_recursion_stop }
541   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#4}
542   {
543     \bool_lazy_or:nnTF
544     {#2} { ! \str_if_eq_p:nn {#4} { 0 } }
545     {
546       \tl_put_right:Nn \l__siunitx_number_partial_tl {#4}
547       \_siunitx_number_parse_uncert:NNNN #1 \c_true_bool #3
548     }
549     { \_siunitx_number_parse_uncert:NNNN #1 \c_false_bool #3 }
550   }

```

```

551     { #3 #1#4 }
552 }

```

For the two auxiliaries, the difference is the handling of a decimal marker: one may be present, but only exactly one.

```

553 \cs_new_protected:Npn \__siunitx_number_parse_uncert_auxi:NN #1#2
554 {
555   \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#2}
556   {
557     \__siunitx_number_parse_uncert_auxiii:N #1
558     \__siunitx_number_parse_uncert_after:N
559   }
560   {
561     \tl_if_in:NnTF \l__siunitx_number_input_decimal_tl {#2}
562     { \__siunitx_number_parse_uncert_marker:N #1 }
563     { \__siunitx_number_parse_loop_break:wN }
564   }
565 }
566 \cs_new_protected:Npn \__siunitx_number_parse_uncert_auxii:NN #1#2
567 {
568   \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#2}
569   {
570     \__siunitx_number_parse_uncert_auxiii:N #1
571     \__siunitx_number_parse_uncert_after:N
572   }
573   { \__siunitx_number_parse_loop_break:wN }
574 }

```

Deal with the closing bracket, which might leave us with nothing if there were no significant digits.

```

575 \cs_new_protected:Npn \__siunitx_number_parse_uncert_auxiii:N #1
576 {
577   \tl_if_empty:NnTF \l__siunitx_number_partial_tl
578   {
579     \tl_put_right:Nx #1
580     {
581       {
582         \bool_if:NT \l__siunitx_number_zero_uncert_bool
583         { { S } { 0 } }
584       }
585     }
586   }
587   {
588     \tl_set:Nx \l__siunitx_number_partial_tl
589     { { S } { \exp_not:V \l__siunitx_number_partial_tl } }
590     \__siunitx_number_parse_loop_main_store:NNN #1
591     \c_false_bool \c_false_bool
592   }
593 }

```

Handling a decimal marker in the uncertainty is a bit tricky: we need to make sure it's valid. First, we need to be sure that the integer part of the captured uncertainty is not too long. Then we need to check that the decimal part is not too long. Both of these require data from the collected partial number, so we extract that first. Checking the decimal part needs the length of the not-yet-collected uncertainty. Handily, we know that

it should be a set of digits then a closing marker. So we can use that as a length: if it's too long we can stop.

```

594 \cs_new_protected:Npn \__siunitx_number_parse_uncert_marker:N #1
595   { \exp_after:wN \__siunitx_number_parse_uncert_marker:nnnN #1 #1 }
596 \cs_new_protected:Npn \__siunitx_number_parse_uncert_marker:nnnN #1#2#3#4
597   {
598     \int_compare:nNnTF
599       { \tl_count:N \l__siunitx_number_partial_tl } > { \tl_count:n {#2} }
600       { \__siunitx_number_parse_loop_break:wN }
601       { \__siunitx_number_parse_uncert_marker:nNw {#3} #4 }
602   }
603 \cs_new_protected:Npn \__siunitx_number_parse_uncert_marker:nNw
604   #1#2#3 \q_recursion_tail \q_recursion_stop
605   {
606     \int_compare:nNnTF
607       { \tl_count:n {#3} - 1 } = { \tl_count:n {#1} }
608       {
609         \str_if_eq:eeTF
610           { \exp_not:V \l__siunitx_number_partial_tl }
611           { \prg_replicate:nn { \tl_count:N \l__siunitx_number_partial_tl } { 0 } }
612           {
613             \__siunitx_number_parse_uncert:NNNN
614             #2 \c_false_bool
615           }
616           {
617             \__siunitx_number_parse_uncert:NNNN
618             #2 \c_true_bool
619           }
620           \__siunitx_number_parse_uncert_auxii:NN
621         }
622         { \__siunitx_number_parse_loop_break:wN }
623       #3 \q_recursion_tail \q_recursion_stop
624   }

```

No further tokens are allowed after an uncertainty in parenthesis.

```

625 \cs_new_protected:Npn \__siunitx_number_parse_uncert_after:N #1
626   {
627     \quark_if_recursion_tail_stop:N #1
628     \__siunitx_number_parse_loop_break:wN
629   }

```

(End definition for __siunitx_number_parse_uncert:NN and others.)

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
630 \keys_define:nn { siunitx }
631   {
632     drop-exponent .bool_set:N =
633       \l__siunitx_number_drop_exponent_bool ,
634     drop-uncertainty .bool_set:N =
635       \l__siunitx_number_drop_uncertainty_bool ,
636     drop-zero-decimal .bool_set:N =
637       \l__siunitx_number_drop_zero_decimal_bool ,

```

```

638 exponent-mode .choices:nn =
639   { engineering , fixed , input , scientific }
640   { \tl_set_eq:NN \l__siunitx_number_exponent_mode_tl \l_keys_choice_tl } ,
641 fixed-exponent .int_set:N =
642   \l__siunitx_number_exponent_fixed_int ,
643 minimum-decimal-digits .int_set:N =
644   \l__siunitx_number_min_decimal_int ,
645 minimum-integer-digits .int_set:N =
646   \l__siunitx_number_min_integer_int ,
647 round-half .choice: ,
648 round-half / even .code:n =
649   { \bool_set_true:N \l__siunitx_number_round_half_even_bool } ,
650 round-half / up .code:n =
651   { \bool_set_false:N \l__siunitx_number_round_half_even_bool } ,
652 round-minimum .code:n =
653   { \__siunitx_number_set_round_min:n {#1} } ,
654 round-mode .choices:nn =
655   { figures , none , places , uncertainty }
656   { \tl_set_eq:NN \l__siunitx_number_round_mode_tl \l_keys_choice_tl } ,
657 round-pad .bool_set:N =
658   \l__siunitx_number_round_pad_bool ,
659 round-precision .int_set:N =
660   \l__siunitx_number_round_precision_int ,
661 }
662 \bool_new:N \l__siunitx_number_round_half_even_bool
663 \tl_new:N \l__siunitx_number_exponent_mode_tl
664 \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.

```

665 \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.
 __siunitx_number_set_round_min:nnnnnnn That's mainly a question of checking if there are any significant digits in the mantissa given.

```

666 \cs_new_protected:Npn \__siunitx_number_set_round_min:n #1
667 {
668   \siunitx_number_parse:nN {#1} \l__siunitx_number_tmp_tl
669   \exp_after:wN \__siunitx_number_set_round_min:nnnnnnn \l__siunitx_number_tmp_tl
670 }
671 \cs_new:Npn \__siunitx_number_set_round_min:nnnnnnn #1#2#3#4#5#6#7
672 {
673   \tl_set:Nx \l__siunitx_number_round_min_tl
674   {
675     \bool_lazy_and:nnF
676     { \str_if_eq_p:nn {#3} { 0 } }
677     {
678       \str_if_eq_p:ee
679       { \exp_not:n {#4} }
680       { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
681     }
682   }

```

```

682         { \exp_not:n { {#3} {#4} } }
683     }
684 }

```

(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`

```

685 \cs_new_protected:Npn \siunitx_number_process:NN #1#2
686 {
687     \tl_if_empty:NTF #1
688     { \tl_clear:N #2 }
689     {
690         \__siunitx_number_drop_uncertainty:NN #1 #2
691         \exp_after:wN \__siunitx_number_process:nnnnnnnNN #2 #2 #2
692         \__siunitx_number_drop_exponent:NN #2 #2
693         \__siunitx_number_zero_decimal:NN #2 #2
694         \__siunitx_number_digits:NN #2 #2
695     }
696 }
697 \cs_new_protected:Npn \__siunitx_number_process:nnnnnnnNN #1#2#3#4#5#6#7#8#9
698 {
699     \bool_lazy_and:nnF
700     { \str_if_eq_p:nn {#3} { 0 } }
701     {
702         \str_if_eq_p:ee
703         { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
704     }
705     {
706         \__siunitx_number_exponent:NN #8 #9
707         \__siunitx_number_round:NN #9 #9
708     }
709 }

```

(End definition for `\siunitx_number_process:NN` and `__siunitx_number_process:nnnnnnnNN`. This function is documented on page 37.)

`__siunitx_number_exponent:NN`

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

`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:fnn`

`__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`

`siunitx_number_exponent_engineering_aux:nnnnnnn`

`siunitx_number_exponent_engineering_0:nnnn`

`siunitx_number_exponent_engineering_1:nnnn`

`siunitx_number_exponent_engineering_2:nnnn`

```

710 \cs_new_protected:Npn \__siunitx_number_exponent:NN #1#2
711 {

```

```

712     \tl_set:Nx #2
713     {

```

```

714         \cs:w

```

```

715         __siunitx_number_exponent_ \l__siunitx_number_exponent_mode_tl :nnnnnnn

```

```

716         \exp_after:wN

```

```

717         \cs_end: #1
718     }

```

```

719     \str_if_eq:VnT \l__siunitx_number_exponent_mode_tl { engineering }
720     {

```

```

721         \tl_set:Nx #2

```

```

722         { \exp_after:wN \__siunitx_number_exponent_engineering_aux:nnnnnnn #2 }
723     }

```



```

724 }
725 \cs_new:Npn \__siunitx_number_exponent_fixed:nnnnnnn #1#2#3#4#5#6#7
726 {
727   \exp_args:Nf \__siunitx_number_exponent_fixed:nnnnnnnn
728   { \int_eval:n { \l__siunitx_number_exponent_fixed_int - (#6#7) } }
729   {#1} {#2} {#3} {#4} {#5} {#6} {#7}
730 }
731 \cs_new:Npn \__siunitx_number_exponent_fixed:nnnnnnnn #1#2#3#4#5#6#7#8
732 {
733   \exp_not:n { {#2} {#3} }
734   \__siunitx_number_exponent_shift:nnn {#1} {#4} {#5}
735   \__siunitx_number_exponent_uncert:n {#6}
736   \exp_not:n { {#7} } { \int_use:N \l__siunitx_number_exponent_fixed_int }
737 }
738 \cs_new:Npn \__siunitx_number_exponent_input:nnnnnnn #1#2#3#4#5#6#7
739 { \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.

```

740 \cs_new:Npn \__siunitx_number_exponent_scientific:nnnnnnn #1#2#3#4#5#6#7
741 {
742   \exp_args:Nf \__siunitx_number_exponent_scientific:nnnnnnnn
743   { \int_eval:n { \tl_count:n {#3} } }
744   {#1} {#2} {#3} {#4} {#5} {#6} {#7}
745 }
746 \cs_new:Npn \__siunitx_number_exponent_scientific:nnnnnnnn #1#2#3#4#5#6#7#8
747 {
748   \exp_not:n { {#2} {#3} }
749   \int_compare:nNnTF {#1} = 1
750   {
751     \str_if_eq:nnTF {#4} { 0 }
752     {
753       \__siunitx_number_exponent_scientific:nnnw
754       { 0 } {#6} { #7#8 } #5 \q_stop
755     }
756     { \exp_not:n { {#4} {#5} {#6} {#7} {#8} } }
757   }
758   {
759     \__siunitx_number_exponent_shift:nnn { #1 - 1 } {#4} {#5}
760     \__siunitx_number_exponent_uncert:n {#6}
761     \__siunitx_number_exponent_finalise:n { #1 + #7#8 - 1 }
762   }
763 }
764 \cs_new_eq:NN \__siunitx_number_exponent_engineering:nnnnnnn
765 \__siunitx_number_exponent_scientific:nnnnnnn
766 \cs_new:Npn \__siunitx_number_exponent_scientific:nnnw #1#2#3#4#5 \q_stop
767 {
768   \str_if_eq:nnTF {#4} { 0 }
769   {
770     \__siunitx_number_exponent_scientific:nnnw
771     { #1 - 1 } {#2} {#3} #5 \q_stop
772   }
773   {

```

```

774         \exp_not:n { #4 } { #5 } { #2 } }
775         \__siunitx_number_exponent_finalise:n { #1 + #3 - 1 }
776     }
777 }

```

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

```

778 \cs_new:Npn \__siunitx_number_exponent_shift:nnn #1#2#3
779 {
780     \int_compare:nNnTF {#1} > 0
781     { \__siunitx_number_exponent_shift_down:nnnw {#1} {#3} { } #2 \q_stop }
782     {
783         \int_compare:nNnTF {#1} < 0
784         { \__siunitx_number_exponent_shift_up:nnn {#1} {#2} {#3} }
785         { {#2} {#3} }
786     }
787 }
788 \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.

```

789 \cs_new:Npn \__siunitx_number_exponent_shift_down:nnnw #1#2#3#4#5 \q_stop
790 {
791     \tl_if_blank:nTF {#5}
792     { \__siunitx_number_exponent_shift_down:nnn {#1} { #4 #3 } {#2} }
793     { \__siunitx_number_exponent_shift_down:nnnw {#1} {#2} { #4 #3 } #5 \q_stop }
794 }
795 \cs_new:Npn \__siunitx_number_exponent_shift_down:nnn #1#2#3
796 {
797     \int_compare:nNnTF {#1} = 0
798     { { \tl_reverse:n {#2} } \exp_not:n { {#3} } }
799     { \__siunitx_number_exponent_shift_down:nw {#1} #2 \q_stop {#3} }
800 }
801 \cs_new:Npn \__siunitx_number_exponent_shift_down:nw #1#2#3 \q_stop #4
802 {
803     \tl_if_blank:nTF {#3}
804     { \__siunitx_number_exponent_shift_down:nnn { #1 - 1 } { 0 } { #2#4 } }
805     { \__siunitx_number_exponent_shift_down:nnn { #1 - 1 } {#3} { #2#4 } }
806 }

```

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.

```

807 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnn #1#2#3
808 {
809     \tl_if_blank:nTF {#3}
810     {
811         \__siunitx_number_exponent_shift_up_aux:ffn
812         { \int_eval:n { #1 + 1 } }
813         { \str_if_eq:nnF {#2} { 0 } {#2} 0 }
814         { }
815         \__siunitx_number_exponent_shift_uncert:nw { 1 }
816     }
817     { \__siunitx_number_exponent_shift_up:nnw {#1} {#2} #3 \q_stop }

```

```

818 }
819 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnw #1#2#3#4 \q_stop
820 {
821   \__siunitx_number_exponent_shift_up_aux:ffn
822   { \int_eval:n { #1 + 1 } }
823   { \str_if_eq:nnF {#2} { 0 } {#2} #3 }
824   {#4}
825 }
826 \cs_new:Npn \__siunitx_number_exponent_shift_up_aux:nnn #1#2#3
827 {
828   \int_compare:nNnTF {#1} = 0
829   { \exp_not:n { {#2} {#3} } }
830   {
831     \tl_if_blank:nTF {#3}
832     {
833       {
834         \exp_not:n {#2}
835         \prg_replicate:nn { \int_abs:n {#1} } { 0 }
836       }
837       { }
838       \__siunitx_number_exponent_shift_uncert:nw { \int_abs:n {#1} }
839     }
840     { \__siunitx_number_exponent_shift_up:nnn {#1} {#2} {#3} }
841   }
842 }
843 \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.

```

844 \cs_new:Npn \__siunitx_number_exponent_shift_uncert:nw
845   #1#2 \__siunitx_number_exponent_uncert:n #3
846   {
847     \tl_if_blank:nTF {#3}
848     {
849       #2
850       \__siunitx_number_exponent_uncert:n { }
851     }
852     {
853       \str_if_eq:nnTF {#3} { 0 }
854       {
855         #2
856         \__siunitx_number_exponent_uncert:n { { S } { 0 } }
857       }
858       {
859         \use:c { __siunitx_number_exponent_shift_uncert_ \use_i:nn #3 :fnnn }
860         { \prg_replicate:nn {#1} { 0 } }
861         {#2}
862         #3
863       }
864     }
865   }
866 \cs_new:Npn \__siunitx_number_exponent_shift_uncert_S:nnnn #1#2#3#4
867 {

```

```

868     #2
869     \__siunitx_number_exponent_uncert:n { { S } { #4#1 } }
870   }
871   \cs_generate_variant:Nn \__siunitx_number_exponent_shift_uncert_S:nnnn { f }
872   \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.

```

873   \cs_new:Npn \__siunitx_number_exponent_finalise:n #1
874   {
875     \int_compare:nNnTF {#1} < 0
876       { { - } }
877       { { } }
878     { \int_abs:n {#1} }
879   }

```

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.

```

880   \cs_new:Npn \__siunitx_number_exponent_engineering_aux:nnnnnnn #1#2#3#4#5#6#7
881   {
882     \exp_not:n { {#1} {#2} }
883     \use:c
884     {
885       __siunitx_number_exponent_engineering_
886       \int_compare:nNnTF {#6#7} < 0
887       {
888         \int_case:nnF { \int_mod:nn { #7 } { 3 } }
889         {
890           { 1 } { 2 }
891           { 2 } { 1 }
892         }
893         { 0 }
894       }
895       { \int_mod:nn {#7} { 3 } }
896       :nnnn
897     }
898     {#3} {#4} {#5} {#6#7}
899   }
900   \cs_new:cpn { __siunitx_number_exponent_engineering_0:nnnn } #1#2#3#4
901   {
902     \exp_not:n { {#1} {#2} {#3} }
903     \__siunitx_number_exponent_finalise:n {#4}
904   }
905   \cs_new:cpn { __siunitx_number_exponent_engineering_1:nnnn } #1#2#3#4
906   {
907     \tl_if_blank:nTF {#2}
908     {
909       { \exp_not:n { #1 0 } } { }
910       { \__siunitx_number_exponent_engineering_uncert:nn {#3} { 0 } }
911     }
912     {
913       { \exp_not:n {#1} \exp_not:o { \tl_head:w #2 \q_stop } }
914       { \exp_not:f { \tl_tail:n {#2} } }
915       { \exp_not:n {#3} }

```

```

916     }
917     \__siunitx_number_exponent_finalise:n { #4 - 1 }
918   }
919   \cs_new:cpn { __siunitx_number_exponent_engineering_2:nnnn } #1#2#3#4
920   {
921     \tl_if_blank:nTF {#2}
922     {
923       { \exp_not:n { #1 00 } } { }
924       { \__siunitx_number_exponent_engineering_uncert:nn {#3} { 00 } }
925     }
926     { \__siunitx_number_exponent_engineering:nnNw {#1} {#3} #2 \q_stop }
927     \__siunitx_number_exponent_finalise:n { #4 - 2 }
928   }
929   \cs_new:Npn \__siunitx_number_exponent_engineering:nnNw #1#2#3#4 \q_stop
930   {
931     \tl_if_blank:nTF {#4}
932     {
933       { \exp_not:n { #1#3 0 } } { }
934       { { \__siunitx_number_exponent_engineering_uncert:nn {#2} { 0 } } }
935     }
936     {
937       { \exp_not:n {#1#3} \exp_not:o { \tl_head:w #4 \q_stop } }
938       { \exp_not:f { \tl_tail:n {#4} } }
939       { \exp_not:n {#2} }
940     }
941   }
942   \cs_new:Npn \__siunitx_number_exponent_engineering_uncert:nn #1#2
943   {
944     \tl_if_blank:nF {#1}
945     {
946       \use:c { __siunitx_number_exponent_engineering_uncert_ \use_i:nn #1 :nnn }
947       #1 {#2}
948     }
949   }
950   \cs_new:Npn \__siunitx_number_exponent_engineering_uncert_S:nnn #1#2#3
951   {
952     { S }
953     {
954       \exp_not:n {#2}
955       \str_if_eq:nnF {#2} { 0 } {#3}
956     }
957   }

```

(End definition for __siunitx_number_exponent:NN and others.)

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

```

\__siunitx_number_digits:Nn
\__siunitx_number_digits:nn
\__siunitx_number_digits_S:n
958 \cs_new_protected:Npn \__siunitx_number_digits:NN #1#2
959 {
960   \tl_set:Nx #2
961   { \exp_after:wN \__siunitx_number_digits:nnnnnnn #1 }
962 }
963 \cs_new:Npn \__siunitx_number_digits:nnnnnnn #1#2#3#4#5#6#7

```

```

964 {
965   \exp_not:n { {#1} {#2} }
966   {
967     \__siunitx_number_digits:Nn \l__siunitx_number_min_integer_int {#3}
968     \exp_not:n {#3}
969   }
970   {
971     \exp_not:n {#4}
972     \__siunitx_number_digits:Nn \l__siunitx_number_min_decimal_int {#4}
973   }
974   { \tl_if_blank:nF {#5} { \__siunitx_number_digits_uncert:nn #5 } }
975   \exp_not:n { {#6} {#7} }
976 }
977 \cs_new:Npn \__siunitx_number_digits:Nn #1#2
978 {
979   \int_compare:nNnT
980     { #1 - \tl_count:n {#2} } > 0
981     { \prg_replicate:nn { #1 - \tl_count:n {#2} } { 0 } }
982 }
983 \cs_new:Npn \__siunitx_number_digits_uncert:nn #1#2
984 {
985   { #1 }
986   { \use:c { __siunitx_number_digits_uncert_ #1 :n } {#2} }
987 }
988 \cs_new:Npn \__siunitx_number_digits_uncert_S:n #1
989 {
990   \exp_not:n {#1}
991   \__siunitx_number_digits:Nn \l__siunitx_number_min_decimal_int {#1}
992 }

```

(End definition for __siunitx_number_digits:NN and others.)

__siunitx_number_drop_exponent:NN
 __siunitx_number_drop_exponent:nnnnnnn

Simple stripping of the exponent.

```

993 \cs_new_protected:Npn \__siunitx_number_drop_exponent:NN #1#2
994 {
995   \bool_if:NT \l__siunitx_number_drop_exponent_bool
996   {
997     \tl_set:Nx #2
998       { \exp_after:wN \__siunitx_number_drop_exponent:nnnnnnn #1 }
999   }
1000 }
1001 \cs_new:Npn \__siunitx_number_drop_exponent:nnnnnnn #1#2#3#4#5#6#7
1002 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} { } { 0 } } }

```

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

__siunitx_number_drop_uncertainty:NN
 __siunitx_number_drop_uncertainty:nnnnnnn

Simple stripping of the uncertainty.

```

1003 \cs_new_protected:Npn \__siunitx_number_drop_uncertainty:NN #1#2
1004 {
1005   \bool_if:NTF \l__siunitx_number_drop_uncertainty_bool
1006   {
1007     \tl_set:Nx #2
1008       { \exp_after:wN \__siunitx_number_drop_uncertainty:nnnnnnn #1 }
1009   }
1010   { \tl_set_eq:NN #2 #1 }

```

```

1011 }
1012 }
1013 \cs_new:Npn \__siunitx_number_drop_uncertainty:nnnnnnn #1#2#3#4#5#6#7
1014 { \exp_not:n { {#1} {#2} {#3} {#4} { } {#6} {#7} } }

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

```

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

```

1015 \cs_new_protected:Npn \__siunitx_number_round:NN #1#2
1016 {
1017   \tl_set:Nx #2
1018   {
1019     \cs:w
1020     __siunitx_number_round_ \l__siunitx_number_round_mode_tl :nnnnnnn
1021     \exp_after:wN
1022     \cs_end: #1
1023   }
1024 }
1025 \cs_new:Npn \__siunitx_number_round_none:nnnnnnn #1#2#3#4#5#6#7
1026 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }

```

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

__siunitx_number_round:nnn 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.

```

1027 \cs_new:Npn \__siunitx_number_round:nnn #1#2#3
1028 {
1029   \__siunitx_number_round_auxi:nnnN {#1} {#2} { }
1030   #3 \q_recursion_tail \q_recursion_stop
1031 }
1032 \cs_generate_variant:Nn \__siunitx_number_round:nnn { f }
1033 \cs_new:Npn \__siunitx_number_round_auxi:nnnN #1#2#3#4
1034 {
1035   \quark_if_recursion_tail_stop_do:Nn #4
1036   {
1037     \__siunitx_number_round_auxii:nnnN {#1} {#3} { } #2
1038     \q_recursion_tail \q_recursion_stop
1039   }
1040   \__siunitx_number_round_auxi:nnnN {#1} {#2} {#4#3}
1041 }
1042 \cs_new:Npn \__siunitx_number_round_auxii:nnnN #1#2#3#4
1043 {
1044   \quark_if_recursion_tail_stop_do:Nn #4
1045   {
1046     \tl_if_blank:nTF {#2}
1047     {
1048       \__siunitx_number_round_auxiv:nnnN {#1} { } { } #3
1049       \q_recursion_tail \q_recursion_stop
1050     }
1051     {
1052       \__siunitx_number_round_auxiii:nnnN {#1} {#3} { } #2

```

```

1053         \q_recursion_tail \q_recursion_stop
1054     }
1055 }
1056 \__siunitx_number_round_auxii:nnnN {#1} {#2} {#4#3}
1057 }

```

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.

```

1058 \cs_new:Npn \__siunitx_number_round_auxiii:nnnN #1#2#3#4
1059 {
1060     \quark_if_recursion_tail_stop_do:Nn #4
1061     {
1062         \__siunitx_number_round_auxiv:nnnN {#1} { } {#3} #2
1063         \q_recursion_tail \q_recursion_stop
1064     }
1065     \int_compare:nNnTF {#1} > 0
1066     {
1067         \exp_args:Nf \__siunitx_number_round_auxiii:nnnN
1068         { \int_eval:n { #1 - 1 } } {#2} { #4#3 }
1069     }
1070     { \__siunitx_number_round_auxv:nnN {#3} {#2} #4 }
1071 }
1072 \cs_new:Npn \__siunitx_number_round_auxiv:nnnN #1#2#3#4
1073 {
1074     \quark_if_recursion_tail_stop_do:Nn #4
1075     { { 0 } { } }
1076     \int_compare:nNnTF {#1} > 0
1077     {
1078         \exp_args:Nf \__siunitx_number_round_auxiv:nnnN
1079         { \int_eval:n { #1 - 1 } } { #2 0 } { #4#3 }
1080     }
1081     { \__siunitx_number_round_auxvi:nnnN {#3} {#2} #4 }
1082 }

```

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.

```

1083 \cs_new:Npn \__siunitx_number_round_auxv:nnN #1#2#3
1084 {
1085     \quark_if_recursion_tail_stop_do:Nn #3
1086     {
1087         \__siunitx_number_round_auxvi:nnN
1088         {#1} { } #2 \q_recursion_tail \q_recursion_stop
1089     }
1090     \bool_lazy_or:nnTF
1091     { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
1092     {
1093         \bool_lazy_all_p:n
1094         {
1095             { \l__siunitx_number_round_half_even_bool }
1096             { \int_if_odd_p:n {#3} }
1097             { \__siunitx_number_round_if_half_p:n {#1} }

```



```

1098     }
1099   }
1100   { \_siunitx_number_round_final_decimal:nnw }
1101   { \_siunitx_number_round_auxvii:nnN }
1102     {#2} { } #3
1103 }
1104 \cs_new:Npn \_siunitx_number_round_auxvi:nnnN #1#2#3
1105 {
1106   \quark_if_recursion_tail_stop_do:Nn #3
1107     { { 0 } { } }
1108   \bool_lazy_or:nnTF
1109     { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
1110     {
1111       \bool_lazy_all_p:n
1112         {
1113           { \l_siunitx_number_round_half_even_bool }
1114           { \int_if_odd_p:n {#3} }
1115           { \_siunitx_number_round_if_half_p:n {#1} }
1116         }
1117     }
1118   { \_siunitx_number_round_final_integer:nnw }
1119   { \_siunitx_number_round_auxviii:nnN }
1120     { } {#2} #3
1121 }

```

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.

```

1122 \cs_new:Npn \_siunitx_number_round_auxvii:nnN #1#2#3
1123 {
1124   \quark_if_recursion_tail_stop_do:Nn #3
1125   {
1126     \str_if_eq:nnTF {#1} { 0 }
1127     {
1128       \_siunitx_number_round_final_output:ff
1129         { 1 }
1130         { \_siunitx_number_round_truncate:n {#2} }
1131     }
1132     {
1133       \_siunitx_number_round_auxviii:nnN {#2} { } #1
1134       \q_recursion_tail \q_recursion_stop
1135     }
1136   }
1137   \int_compare:nNnTF {#3} = 9
1138   { \_siunitx_number_round_auxvii:nnN {#1} { 0 #2 } }
1139   {
1140     \int_compare:nNnTF {#3} = 0
1141     {
1142       \_siunitx_number_round_final_decimal:nnw
1143         {#1} { 1 \_siunitx_number_round_truncate:n {#2} }
1144     }
1145     {

```

```

1146         \_siunitx_number_round_final:fn
1147         { \int_eval:n { #3 + 1 } }
1148         { \_siunitx_number_round_final_decimal:nnw {#1} {#2} }
1149     }
1150 }
1151 }
1152 \cs_new:Npn \_siunitx_number_round_auxviii:nnN #1#2#3
1153 {
1154     \quark_if_recursion_tail_stop_do:Nn #3
1155     {
1156         \tl_if_blank:nTF {#1}
1157         {
1158             \_siunitx_number_round_final_shift:ff
1159             {
1160                 \exp_last_unbraced:Nf 1
1161                 { \_siunitx_number_round_truncate_direct:n {#2} } 0
1162             }
1163             { }
1164         }
1165         {
1166             \_siunitx_number_round_final_shift:ff
1167             { 1 #2 }
1168             { \_siunitx_number_round_truncate:n {#1} }
1169         }
1170     }
1171     \int_compare:nNnTF {#3} = 9
1172     { \_siunitx_number_round_auxviii:nnN {#1} { 0 #2 } }
1173     {
1174         \_siunitx_number_round_final:fn
1175         { \int_eval:n { #3 + 1 } }
1176         { \_siunitx_number_round_final_integer:nnw {#1} {#2} }
1177     }
1178 }

```

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

```

1179 \cs_new:Npn \_siunitx_number_round_final_decimal:nnw
1180 #1#2#3 \q_recursion_tail \q_recursion_stop
1181 {
1182     \_siunitx_number_round_final_output:ff
1183     { \tl_reverse:n {#1} }
1184     { \tl_reverse:n {#3} #2 }
1185 }
1186 \cs_new:Npn \_siunitx_number_round_final_integer:nnw
1187 #1#2#3 \q_recursion_tail \q_recursion_stop
1188 {
1189     \_siunitx_number_round_final_output:ff
1190     { \tl_reverse:n {#3} #2 }
1191     {#1}
1192 }
1193 \cs_new:Npn \_siunitx_number_round_final_output:nn #1#2 { {#1} {#2} }
1194 \cs_generate_variant:Nn \_siunitx_number_round_final_output:nn { ff }
1195 \cs_new:Npn \_siunitx_number_round_final:nn #1#2
1196 { #2 #1 }
1197 \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.

```

1198 \cs_new:Npn \__siunitx_number_round_final_shift:nn #1#2
1199 {
1200   \str_if_eq:VnTF \l__siunitx_number_round_mode_tl { places }
1201   {
1202     \use:c
1203     { __siunitx_number_round_ \l__siunitx_number_exponent_mode_tl :nn }
1204     {#1} {#2}
1205   }
1206   { {#1} {#2} }
1207 }
1208 \cs_generate_variant:Nn \__siunitx_number_round_final_shift:nn { ff }
1209 \cs_new:Npn \__siunitx_number_round_engineering:nn #1#2
1210 {
1211   \int_compare:nNnTF { \tl_count:n {#1} } = 4
1212   {
1213     \__siunitx_number_round_engineering:NNNNn #1 {#2}
1214     { }
1215     \__siunitx_number_round_final_shift:Nw 3
1216   }
1217   { {#1} {#2} }
1218 }
1219 \cs_new:Npn \__siunitx_number_round_engineering:NNNNn #1#2#3#4#5
1220 {
1221   {#1}
1222   \exp_args:NV \__siunitx_number_round_engineering:nnN
1223   { \l__siunitx_number_round_precision_int } { }
1224   #2#3#4#5 \q_recursion_tail \q_recursion_stop
1225 }
1226 \cs_new:Npn \__siunitx_number_round_engineering:nnN #1#2#3
1227 {
1228   \quark_if_recursion_tail_stop_do:Nn #3 { {#2} }
1229   \int_compare:nNnTF {#1} = { 0 }
1230   { \use_i_delimit_by_q_recursion_stop:nw { {#2} } }
1231   { \__siunitx_number_round_engineering:nnN { #1 - 1 } { #2#3 } }
1232 }
1233 \cs_new:Npn \__siunitx_number_round_fixed:nn #1#2 { {#1} {#2} }
1234 \cs_new:Npn \__siunitx_number_round_input:nn #1#2 { {#1} {#2} }
1235 \cs_new:Npn \__siunitx_number_round_scientific:nn #1#2
1236 {
1237   \__siunitx_number_exponent_shift:nnf
1238   { 1 } {#1} { \__siunitx_number_round_truncate_direct:n {#2} }
1239   { }
1240   \__siunitx_number_round_final_shift:Nw 1
1241 }
1242 \cs_new:Npn \__siunitx_number_round_final_shift:Nw #1#2 \__siunitx_number_round_places_end:nn
1243 { \__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.

```

1244 \cs_new:Npn \__siunitx_number_round_truncate:n #1
1245 {
1246   \str_if_eq:VnTF \l__siunitx_number_round_mode_tl { figures }
1247   { \__siunitx_number_round_truncate_direct:n {#1} }
1248   {#1}
1249 }
1250 \cs_new:Npn \__siunitx_number_round_truncate_direct:n #1
1251 {
1252   \__siunitx_number_round_truncate:nnN { } { }
1253   #1 \q_recursion_tail \q_recursion_stop
1254 }
1255 \cs_new:Npn \__siunitx_number_round_truncate:nnN #1#2#3
1256 {
1257   \quark_if_recursion_tail_stop_do:Nn #3 { #1 }
1258   \__siunitx_number_round_truncate:nnN {#1#2} {#3}
1259 }

```

(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.

```

1260 \prg_new_conditional:Npnn \__siunitx_number_round_if_half:n #1 { p }
1261 {
1262   \int_compare:nNnTF { \tl_head:n { #1 0 } } = 5
1263   {
1264     \exp_after:wN \__siunitx_number_round_if_half:N \use_none:n #1 0
1265     \q_recursion_tail \q_recursion_stop
1266   }
1267   { \prg_return_false: }
1268 }
1269 \cs_new:Npn \__siunitx_number_round_if_half:N #1
1270 {
1271   \quark_if_recursion_tail_stop_do:Nn #1
1272   { \prg_return_true: }
1273   \int_compare:nNnTF {#1} = 0
1274   { \__siunitx_number_round_if_half:N }
1275   { \use_i_delimit_by_q_recursion_stop:nw { \prg_return_false: } }
1276 }

```

(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.

```

1277 \cs_new:Npn \__siunitx_number_round_pad:nnn #1#2#3
1278 {
1279   {#2}
1280   {
1281     #3
1282     \bool_if:NT \l__siunitx_number_round_pad_bool
1283     { \prg_replicate:nn {#1} { 0 } }
1284   }
1285 }

```

(End definition for __siunitx_number_round_pad:nnn.)

```

\__siunitx_number_round_figures:nnnnnnn
\__siunitx_number_round_figures_count:nnN
\__siunitx_number_round_figures_count:nnnN

```

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.

```

1286 \cs_new:Npn \__siunitx_number_round_figures:nnnnnnn #1#2#3#4#5#6#7
1287 {
1288   \tl_if_blank:nTF {#5}
1289   {
1290     \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
1291     {
1292       \exp_not:n { {#1} {#2} }
1293       \__siunitx_number_round_figures_count:nnN {#3} {#4} #3#4
1294       \q_recursion_tail \q_recursion_stop
1295       \exp_not:n { { } {#6} {#7} }
1296     }
1297     { { } { } { } { 0 } { } { } { } { } { 0 } }
1298   }
1299   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1300 }

```

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

```

1301 \cs_new:Npn \__siunitx_number_round_figures_count:nnN #1#2#3
1302 {
1303   \quark_if_recursion_tail_stop_do:Nn #3
1304   { { } { } { } { 0 } { } { } { } { } { 0 } }
1305   \int_compare:nNnTF {#3} = 0
1306   { \__siunitx_number_round_figures_count:nnN {#1} {#2} }
1307   { \__siunitx_number_round_figures_count:nnnN { 1 } {#1} {#2} }
1308 }
1309 \cs_new:Npn \__siunitx_number_round_figures_count:nnnN #1#2#3#4
1310 {
1311   \quark_if_recursion_tail_stop_do:Nn #4
1312   {
1313     \int_compare:nNnTF {#1} > \l__siunitx_number_round_precision_int
1314     {
1315       \__siunitx_number_round:fnn
1316       { \int_eval:n { #1 - \l__siunitx_number_round_precision_int } }
1317       {#2} {#3}
1318     }
1319     {
1320       \__siunitx_number_round_pad:nnn
1321       { \l__siunitx_number_round_precision_int - (#1) } {#2} {#3}
1322     }
1323   }
1324   \exp_args:Nf \__siunitx_number_round_figures_count:nnnN
1325   { \int_eval:n { #1 + 1 } } {#2} {#3}
1326 }

```

(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.

```

1327 \cs_new:Npn \__siunitx_number_round_places:nnnnnnn #1#2#3#4#5#6#7
1328 {
1329   \tl_if_blank:nTF {#5}
1330   {
1331     \exp_args:Ne \__siunitx_number_round_places_finalise:n
1332     {
1333       \exp_not:n { {#1} {#2} }
1334       \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
1335       { \__siunitx_number_round_places_decimal:nn }
1336       { \__siunitx_number_round_places_integer:nn }
1337       {#3} {#4}
1338       \__siunitx_number_round_places_end:nn {#6} {#7}
1339     }
1340   }
1341   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1342 }
1343 \cs_new:Npn \__siunitx_number_round_places_end:nn #1#2 { { } \exp_not:n { {#1} {#2} } }
1344 \cs_new:Npn \__siunitx_number_round_places_decimal:nn #1#2
1345 {
1346   \int_compare:nNnTF
1347   { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} } > 0
1348   {
1349     \__siunitx_number_round_pad:nnn
1350     { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} }
1351     {#1} {#2}
1352   }
1353   {
1354     \__siunitx_number_round:fnn
1355     {
1356       \int_eval:n
1357       { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }
1358     }
1359     {#1} {#2}
1360   }
1361 }
1362 \cs_new:Npn \__siunitx_number_round_places_integer:nn #1#2
1363 {
1364   \__siunitx_number_round:fnn
1365   {
1366     \int_eval:n
1367     { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }
1368   }
1369   {#1} {#2}
1370 }

```

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.

```

1371 \cs_new:Npn \__siunitx_number_round_places_finalise:n #1
1372 { \__siunitx_number_round_places_finalise:nnnnnnn #1 }
1373 \cs_new:Npn \__siunitx_number_round_places_finalise:nnnnnnn #1#2#3#4#5#6#7
1374 {

```

```

1375 \bool_lazy_and:nnTF
1376 { \str_if_eq_p:nn {#3} { 0 } }
1377 {
1378   \str_if_eq_p:ee
1379   { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1380 }
1381 {
1382   \tl_if_empty:NTF \l__siunitx_number_round_min_tl
1383   {
1384     \exp_not:n { {#1} }
1385     { \str_if_eq:nnF {#2} { - } { \exp_not:n {#2} } }
1386     \exp_not:n { {#3} {#4} {#5} {#6} {#7} }
1387   }
1388   {
1389     \exp_after:wN \__siunitx_number_round_places_finalise:nnnnn
1390     \l__siunitx_number_round_min_tl {#2} {#6} {#7}
1391   }
1392 }
1393 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1394 }
1395 \cs_new:Npn \__siunitx_number_round_places_finalise:nnnnn #1#2#3#4#5
1396 {
1397   {
1398     \str_if_eq:nnTF {#3} { - }
1399     { > }
1400     { < }
1401   }
1402   \exp_not:n { {#3} {#1} {#2} { } {#4} {#5} }
1403 }

```

(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 used for rounding.

```

1404 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnnnn #1#2#3#4#5#6#7
1405 {
1406   \bool_lazy_or:nnTF
1407   { \tl_if_blank_p:n {#5} }
1408   { ! \int_compare_p:nNn \l__siunitx_number_round_precision_int > 0 }
1409   { \exp_not:n { {#1} #2 {#3} {#4} { } #6 {#7} } }
1410   {
1411     \str_if_eq:eeTF { \tl_head:n {#5} } { S }
1412     {
1413       \exp_not:n { {#1} {#2} }
1414       \exp_args:Nnno \__siunitx_number_round_uncertainty:nnn
1415       {#3} {#4} { \use_ii:nn #5 }
1416       \exp_not:n { {#6} {#7} }
1417     }
1418     { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1419   }
1420 }

```

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.

```

1421 \cs_new:Npn \__siunitx_number_round_uncertainty:nnn #1#2#3
1422 {
1423   \exp_last_unbraced:Nf \__siunitx_number_round_uncertainty:nnnnn
1424   {
1425     \__siunitx_number_round:fnn
1426     { \tl_count:n {#3} - \l__siunitx_number_round_precision_int } { } {#3}
1427   }
1428   {#1} {#2} {#3}
1429 }
1430 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnn #1#2#3#4#5
1431 {
1432   \tl_if_blank:nTF {#1}
1433   {
1434     \__siunitx_number_round:fnn
1435     { \tl_count:n {#5} - \tl_count:n {#2} } {#3} {#4}
1436     { { S } {#2} }
1437   }
1438   {
1439     \__siunitx_number_round:fnn
1440     { \tl_count:n {#5} - \tl_count:n {#2} + 1 } {#3} {#4}
1441     { { S } { #1 \__siunitx_number_round_truncate_direct:n {#2} } }
1442   }
1443 }

```

(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
1444 \cs_new_protected:Npn \__siunitx_number_zero_decimal:NN #1#2
1445 {
1446   \bool_if:NT \l__siunitx_number_drop_zero_decimal_bool
1447   {
1448     \tl_set:Nx #2
1449     { \exp_after:wN \__siunitx_number_zero_decimal:nnnnnnn #1 }
1450   }
1451 }
1452 \cs_new:Npn \__siunitx_number_zero_decimal:nnnnnnn #1#2#3#4#5#6#7
1453 {
1454   \exp_not:n { {#1} {#2} {#3} }
1455   \str_if_eq:eeTF
1456   { \exp_not:n {#4} }
1457   { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1458   { { } }
1459   { \exp_not:n { {#4} } }
1460   \exp_not:n { {#5} {#6} {#7} }
1461 }

```

(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.

```

1462 \cs_new:Npn \siunitx_number_adjust_exponent:nn #1#2

```

```

\siunitx_number_adjust_exponent:nn
\siunitx_number_adjust_exponent:Nn
\__siunitx_number_adjust_exp:nnnnnnnn
\__siunitx_number_adjust_exp:nn
\__siunitx_number_adjust_exp:nNw

```



```

1463 { \__siunitx_number_adjust_exp:nnnnnnnn #1 {#2} }
1464 \cs_new:Npn \siunitx_number_adjust_exponent:Nn #1#2
1465 {
1466   \tl_if_empty:NF #1
1467   { \exp_args:NV \siunitx_number_adjust_exp:nn #1 {#2} }
1468 }
1469 \cs_new:Npn \__siunitx_number_adjust_exp:nnnnnnnn #1#2#3#4#5#6#7#8
1470 {
1471   \exp_not:n { {#1} {#2} {#3} {#4} {#5} }
1472   \exp_args:Ne \__siunitx_number_adjust_exp:nn { \fp_eval:n { #6#7 + #8 } } {#6}
1473 }
1474 \cs_new:Npn \__siunitx_number_adjust_exp:nn #1#2
1475 { \__siunitx_number_adjust_exp:nNw {#2} #1 \q_stop }
1476 \cs_new:Npn \__siunitx_number_adjust_exp:nNw #1#2#3 \q_stop
1477 {
1478   \token_if_eq_meaning:NNTF #2 -
1479   { { - } { \exp_not:n {#3} } }
1480   { { \str_if_eq:nnT {#1} { + } { + } } { \exp_not:n {#2#3} } }
1481 }

```

(End definition for `\siunitx_number_adjust_exponent:nn` and others. These functions are documented on page 38.)

2.6 Outputting parsed numbers

`\l_siunitx_number_bracket_close_tl`
`\l_siunitx_number_bracket_open_tl`

Purely internal for the present.

```

1482 \tl_new:N \l_siunitx_number_bracket_close_tl
1483 \tl_new:N \l_siunitx_number_bracket_open_tl
1484 \tl_set:Nn \l_siunitx_number_bracket_open_tl { ( }
1485 \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`

```

1486 \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`

```

1487 \tl_new:N \l_siunitx_number_output_decimal_tl

```

(End definition for `\l_siunitx_number_output_decimal_tl`. This variable is documented on page 39.)

`\l_siunitx_number_bracket_negative_bool`

Keys producing tokens in the output.

`\l_siunitx_number_implicit_plus_bool`
`\l_siunitx_number_exponent_base_tl`
`\l_siunitx_number_exponent_product_tl`
`\l_siunitx_number_group_decimal_bool`
`\l_siunitx_number_group_integer_bool`
`\l_siunitx_number_group_minimum_int`
`\l_siunitx_number_group_separator_tl`
`\l_siunitx_number_negative_color_tl`
`\l_siunitx_number_output_uncert_close_tl`
`\l_siunitx_number_output_uncert_open_tl`
`\l_siunitx_number_uncert_mode_tl`
`\l_siunitx_number_uncert_separator_tl`
`\l_siunitx_number_tight_bool`
`\l_siunitx_number_unity_mantissa_bool`
`\l_siunitx_number_zero_exponent_bool`

```

1488 \keys_define:nn { siunitx }
1489 {
1490   bracket-ambiguous-numbers .bool_set:N =
1491     \l_siunitx_number_bracket_ambiguous_bool ,
1492   bracket-negative-numbers .bool_set:N =
1493     \l_siunitx_number_bracket_negative_bool ,
1494   exponent-base .tl_set:N =
1495     \l_siunitx_number_exponent_base_tl ,
1496   exponent-product .tl_set:N =

```

```

1497     \l__siunitx_number_exponent_product_tl ,
1498     group-digits .choice: ,
1499     group-digits / all .code:n =
1500     {
1501         \bool_set_true:N \l__siunitx_number_group_decimal_bool
1502         \bool_set_true:N \l__siunitx_number_group_integer_bool
1503     } ,
1504     group-digits / decimal .code:n =
1505     {
1506         \bool_set_true:N \l__siunitx_number_group_decimal_bool
1507         \bool_set_false:N \l__siunitx_number_group_integer_bool
1508     } ,
1509     group-digits / integer .code:n =
1510     {
1511         \bool_set_false:N \l__siunitx_number_group_decimal_bool
1512         \bool_set_true:N \l__siunitx_number_group_integer_bool
1513     } ,
1514     group-digits / none .code:n =
1515     {
1516         \bool_set_false:N \l__siunitx_number_group_decimal_bool
1517         \bool_set_false:N \l__siunitx_number_group_integer_bool
1518     } ,
1519     group-digits .default:n = all ,
1520     group-minimum-digits .int_set:N =
1521     \l__siunitx_number_group_minimum_int ,
1522     group-separator .tl_set:N =
1523     \l__siunitx_number_group_separator_tl ,
1524     negative-color .tl_set:N =
1525     \l__siunitx_number_negative_color_tl ,
1526     output-close-uncertainty .tl_set:N =
1527     \l__siunitx_number_output_uncert_close_tl ,
1528     output-decimal-marker .tl_set:N =
1529     \l__siunitx_number_output_decimal_tl ,
1530     output-open-uncertainty .tl_set:N =
1531     \l__siunitx_number_output_uncert_open_tl ,
1532     print-implicit-plus .bool_set:N =
1533     \l__siunitx_number_implicit_plus_bool ,
1534     print-unity-mantissa .bool_set:N =
1535     \l__siunitx_number_unity_mantissa_bool ,
1536     print-zero-exponent .bool_set:N =
1537     \l__siunitx_number_zero_exponent_bool ,
1538     tight-spacing .bool_set:N =
1539     \l__siunitx_number_tight_bool ,
1540     uncertainty-mode .choices:nn =
1541     { compact , compact-marker , full , separate }
1542     { \tl_set_eq:NN \l__siunitx_number_uncert_mode_tl \l_keys_choice_tl } ,
1543     uncertainty-separator .tl_set:N =
1544     \l__siunitx_number_uncert_separator_tl
1545 }
1546 \bool_new:N \l__siunitx_number_group_decimal_bool
1547 \bool_new:N \l__siunitx_number_group_integer_bool
1548 \tl_new:N \l__siunitx_number_uncert_mode_tl

```

(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
1549 \cs_new:Npn \siunitx_number_output:N #1
1550 { \_siunitx_number_output:Nn #1 { } }
1551 \cs_new:Npn \siunitx_number_output:n #1
1552 { \_siunitx_number_output:nn #1 { } }
1553 \cs_new:Npn \siunitx_number_output:NN #1#2
1554 { \_siunitx_number_output:Nn #1 {#2} }
1555 \cs_new:Npn \siunitx_number_output:nN #1#2
1556 { \_siunitx_number_output:nn #1 {#2} }
1557 \cs_new:Npn \_siunitx_number_output:Nn #1#2
1558 {
1559   \tl_if_empty:NF #1
1560   { \exp_after:wN \_siunitx_number_output:nnnnnnn #1 {#2} }
1561 }
1562 \cs_new:Npn \_siunitx_number_output:nn #1#2
1563 {
1564   \tl_if_empty:NF {#1}
1565   { \_siunitx_number_output:nnnnnnn #1 {#2} }
1566 }
1567 \cs_new:Npn \_siunitx_number_output:nnnnnnn #1#2#3#4#5#6#7#8
1568 {
1569   \_siunitx_number_output_comparator:nn {#1} {#8}
1570   \_siunitx_number_output_bracket:nn {#5} {#7}
1571   \_siunitx_number_output_sign:nnn {#1} {#2} {#8}
1572   \_siunitx_number_output_integer:nnn {#3} {#4} {#7}
1573   \_siunitx_number_output_decimal:nn {#4} {#8}
1574   \_siunitx_number_output_uncertainty:nnn {#5} {#4} {#8}
1575   \_siunitx_number_output_exponent:nnnn {#6} {#7} { #3 . #4 } {#8}
1576   \_siunitx_number_output_end:
1577 }
1578 \cs_new:Npn \_siunitx_number_output_bracket:nn #1#2
1579 {
1580   \bool_lazy_all:nT
1581   {
1582     { \str_if_eq_p:Vn \l__siunitx_number_uncert_mode_tl { separate } }
1583     { \l_siunitx_number_bracket_ambiguous_bool }
1584     { ! \tl_if_blank_p:n {#1} }
1585   }
1586   \bool_lazy_or_p:nn
1587   { \l__siunitx_number_zero_exponent_bool }
1588   { ! \str_if_eq_p:nn {#2} { 0 } }
1589 }
1590 }
1591 \_siunitx_number_output_bracket:w
1592 }
1593 \cs_new:Npn \_siunitx_number_output_bracket:w #1 \_siunitx_number_output_exponent:nnnn
1594 {
1595   \exp_not:V \l__siunitx_number_bracket_open_tl

```

```

1596 #1
1597 \exp_not:V \l__siunitx_number_bracket_close_tl
1598 \__siunitx_number_output_exponent:nnnn
1599 }

```

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

```

1600 \cs_new:Npn \__siunitx_number_output_comparator:nn #1#2
1601 {
1602   \tl_if_blank:nF {#1}
1603     { \exp_not:n { \mathord {#1} } }
1604   \exp_not:n {#2}
1605 }

```

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.

```

1606 \cs_new:Npn \__siunitx_number_output_sign:nnn #1#2#3
1607 {
1608   \tl_if_blank:nTF {#2}
1609   {
1610     \bool_if:NT \l__siunitx_number_implicit_plus_bool
1611       { \__siunitx_number_output_sign:nN {#1} + }
1612   }
1613   {
1614     \str_if_eq:nnTF {#2} { - }
1615     {
1616       \tl_if_empty:NF \l__siunitx_number_negative_color_tl
1617       { \__siunitx_number_output_sign_color:w }
1618       \bool_if:NTF \l__siunitx_number_bracket_negative_bool
1619       { \__siunitx_number_output_sign_brackets:w }
1620       { \__siunitx_number_output_sign:nN {#1} #2 }
1621     }
1622     { \__siunitx_number_output_sign:nN {#1} #2 }
1623   }
1624   \exp_not:n {#3}
1625 }
1626 \cs_new:Npn \__siunitx_number_output_sign:nN #1#2
1627 {
1628   \tl_if_blank:nTF {#1}
1629   { \__siunitx_number_output_sign:N #2 }
1630   { \exp_not:n { \mathord {#2} } }
1631 }
1632 \cs_new:Npn \__siunitx_number_output_sign:N #1
1633 {
1634   \bool_if:NTF \l__siunitx_number_tight_bool
1635   { \exp_not:n { \mathord {#1} } }
1636   { \exp_not:n {#1} }
1637 }
1638 \cs_new:Npn
1639   \__siunitx_number_output_sign_color:w #1 \__siunitx_number_output_end:
1640 {
1641   \exp_not:N \textcolor { \exp_not:V \l__siunitx_number_negative_color_tl }
1642   {

```

```

1643         #1
1644         \_siunitx_number_output_end:
1645     }
1646 }
1647 \cs_new:Npn
1648   \_siunitx_number_output_sign_brackets:w #1 \_siunitx_number_output_end:
1649 {
1650   \exp_not:V \l_siunitx_number_bracket_open_tl
1651   #1
1652   \exp_not:V \l_siunitx_number_bracket_close_tl
1653   \_siunitx_number_output_end:
1654 }

```

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.

```

1655 \cs_new:Npn \_siunitx_number_output_integer:nnn #1#2#3
1656 {
1657   \bool_lazy_all:nF
1658   {
1659     { \str_if_eq_p:nn {#1} { 1 } }
1660     { \tl_if_blank_p:n {#2} }
1661     { ! \str_if_eq_p:nn {#3} { 0 } }
1662     { ! \l_siunitx_number_unity_mantissa_bool }
1663   }
1664   { \_siunitx_number_output_digits:nn { integer } {#1} }
1665 }
1666 \cs_new:Npn \_siunitx_number_output_decimal:nn #1#2
1667 {
1668   \exp_not:n {#2}
1669   \tl_if_blank:nF {#1}
1670   {
1671     \str_if_eq:VnTF \l_siunitx_number_output_decimal_tl { , }
1672     { \exp_not:N \mathord }
1673     { \use:n }
1674     { \exp_not:V \l_siunitx_number_output_decimal_tl }
1675   }
1676   \exp_not:n {#2}
1677   \_siunitx_number_output_digits:nn { decimal } {#1}
1678 }
1679 \cs_generate_variant:Nn \_siunitx_number_output_decimal:nn { f }
1680 \cs_new:Npn \_siunitx_number_output_digits:nn #1#2
1681 {
1682   \bool_if:cTF { l_siunitx_number_group_ #1 _ bool }
1683   {
1684     \int_compare:nNnTF
1685     { \tl_count:n {#2} } < \l_siunitx_number_group_minimum_int
1686     { \exp_not:n {#2} }
1687     { \use:c { __siunitx_number_output_ #1 _aux:n } {#2} }
1688   }
1689   { \exp_not:n {#2} }
1690 }

```

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.

```

1691 \cs_new:Npn \__siunitx_number_output_integer_aux:n #1
1692 {
1693   \use:c
1694   {
1695     __siunitx_number_output_integer_aux_
1696     \int_eval:n { \int_mod:nn { \tl_count:n {#1} } { 3 } }
1697     :n
1698   } {#1}
1699 }
1700 \cs_new:cpn { __siunitx_number_output_integer_aux_0:n } #1
1701 { \__siunitx_number_output_integer_first:nnNN #1 \q_nil }
1702 \cs_new:cpn { __siunitx_number_output_integer_aux_1:n } #1
1703 { \__siunitx_number_output_integer_first:nnNN { } { } #1 \q_nil }
1704 \cs_new:cpn { __siunitx_number_output_integer_aux_2:n } #1
1705 { \__siunitx_number_output_integer_first:nnNN { } #1 \q_nil }
1706 \cs_new:Npn \__siunitx_number_output_integer_first:nnNN #1#2#3#4
1707 {
1708   \exp_not:n {#1#2#3}
1709   \quark_if_nil:NF #4
1710   { \__siunitx_number_output_integer_loop:NNNN #4 }
1711 }
1712 \cs_new:Npn \__siunitx_number_output_integer_loop:NNNN #1#2#3#4
1713 {
1714   \str_if_eq:VnTF \l__siunitx_number_group_separator_tl { , }
1715   { \exp_not:N \mathord }
1716   { \use:n }
1717   { \exp_not:V \l__siunitx_number_group_separator_tl }
1718   \exp_not:n {#1#2#3}
1719   \quark_if_nil:NF #4
1720   { \__siunitx_number_output_integer_loop:NNNN #4 }
1721 }

```

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.

```

1722 \cs_new:Npn \__siunitx_number_output_decimal_aux:n #1
1723 {
1724   \__siunitx_number_output_decimal_loop:NNNN \c_empty_tl
1725   #1 \q_nil \q_nil \q_nil
1726 }
1727 \cs_new:Npn \__siunitx_number_output_decimal_loop:NNNN #1#2#3#4
1728 {
1729   \quark_if_nil:NF #2
1730   {
1731     \exp_not:V #1
1732     \exp_not:n {#2}
1733     \quark_if_nil:NTF #3
1734     { \use_none:n }
1735     {
1736       \exp_not:n {#3}
1737       \quark_if_nil:NTF #4
1738       { \use_none:nn }

```

```

1739         {
1740             \exp_not:n {#4}
1741             \__siunitx_number_output_decimal_loop:NNNN
1742             \l__siunitx_number_group_separator_tl
1743         }
1744     }
1745 }
1746 }

```

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.

```

1747 \cs_new:Npn \__siunitx_number_output_uncertainty:nnn #1#2#3
1748 {
1749     \tl_if_blank:nTF {#1}
1750     { \__siunitx_number_output_uncertainty_unaligned:n {#3} }
1751     {
1752         \use:c { __siunitx_number_output_uncert_ \tl_head:n {#1} :nnnw }
1753         {#2} {#3} #1
1754     }
1755 }
1756 \cs_new:Npn \__siunitx_number_output_uncertainty_unaligned:n #1
1757 { \exp_not:n { #1 #1 #1 #1 } }
1758 \cs_new:Npn \__siunitx_number_output_uncert_S:nnnw #1#2#3#4
1759 {
1760     \str_if_eq:VnTF \l__siunitx_number_uncert_mode_tl { separate }
1761     {
1762         \exp_not:n {#2}
1763         \__siunitx_number_output_sign:N \pm
1764         \exp_not:n {#2}
1765         \__siunitx_number_output_uncert_S_aux:nnn
1766         { \int_eval:n { \tl_count:n {#4} - \tl_count:n {#1} } }
1767         {#4} {#2}
1768     }
1769     {
1770         \exp_not:V \l__siunitx_number_uncert_separator_tl
1771         \exp_not:V \l__siunitx_number_output_uncert_open_tl
1772         \use:c { __siunitx_number_output_uncert_S_ \l__siunitx_number_uncert_mode_tl :nn } {#3}
1773         \exp_not:V \l__siunitx_number_output_uncert_close_tl
1774         \__siunitx_number_output_uncertainty_unaligned:n {#2}
1775     }
1776 }
1777 \cs_new:Npn \__siunitx_number_output_uncert_S_aux:nnn #1#2#3
1778 {
1779     \int_compare:nNnTF {#1} > 0
1780     {
1781         \__siunitx_number_output_uncert_S_aux:fnnw
1782         { \int_eval:n { #1 - 1 } }
1783         {#3}
1784         { }
1785         #2 \q_nil
1786     }

```

```

1787     {
1788       0
1789       \__siunitx_number_output_decimal:fn
1790       {
1791         \prg_replicate:nn { \int_abs:n {#1} } { 0 }
1792         #2
1793       }
1794       {#3}
1795     }
1796   }
1797 \cs_generate_variant:Nn \__siunitx_number_output_uncert_S_aux:nnn { f }
1798 \cs_new:Npn \__siunitx_number_output_uncert_S_aux:nnnw #1#2#3#4
1799 {
1800   \quark_if_nil:NF #4
1801   {
1802     \int_compare:nNnTF {#1} = 0
1803     { \__siunitx_number_output_uncert_S_aux:nnw {#3#4} {#2} }
1804     {
1805       \__siunitx_number_output_uncert_S_aux:fnnw
1806       { \int_eval:n { #1 - 1 } }
1807       {#2}
1808       {#3#4}
1809     }
1810   }
1811 }
1812 \cs_generate_variant:Nn \__siunitx_number_output_uncert_S_aux:nnnw { f }
1813 \cs_new:Npn \__siunitx_number_output_uncert_S_aux:nnw #1#2#3 \q_nil
1814 {
1815   \__siunitx_number_output_digits:nn { integer } {#1}
1816   \__siunitx_number_output_decimal:nn {#3} {#2}
1817 }

```

Handle the content of brackets: the only complex case is the mixed situation.

```

1818 \cs_new:Npn \__siunitx_number_output_uncert_S_compact:nn #1#2
1819 { \exp_not:n {#2} }
1820 \cs_new:cpn { \__siunitx_number_output_uncert_S_compact-marker:nn } #1#2
1821 {
1822   \bool_lazy_or:nnTF
1823   { \tl_if_blank_p:n {#1} }
1824   { ! \int_compare_p:nNn { \tl_count:n {#2} } > { \tl_count:n {#1} } }
1825   { \__siunitx_number_output_uncert_S_compact:nn }
1826   { \__siunitx_number_output_uncert_S_full:nn }
1827   {#1} {#2}
1828 }
1829 \cs_new:Npn \__siunitx_number_output_uncert_S_full:nn #1#2
1830 {
1831   \__siunitx_number_output_uncert_S_aux:fnn
1832   { \int_eval:n { \tl_count:n {#2} - \tl_count:n {#1} } }
1833   {#2} { }
1834 }

```

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.

```

1835 \cs_new:Npn \__siunitx_number_output_exponent:nnnn #1#2#3#4
1836 {
1837   \exp_not:n {#4}
1838   \bool_lazy_or:nnTF
1839     { \l__siunitx_number_zero_exponent_bool }
1840     { ! \str_if_eq_p:nn {#2} { 0 } }
1841   {
1842     \bool_lazy_and:nnTF
1843       { \str_if_eq_p:nn {#3} { 1. } }
1844       { ! \l__siunitx_number_unity_mantissa_bool }
1845       { \exp_not:n {#4} }
1846     {
1847       \bool_if:NTF \l__siunitx_number_tight_bool
1848         { \exp_not:N \mathord }
1849         { \use:n }
1850         { \exp_not:V \l__siunitx_number_exponent_product_tl }
1851       \exp_not:n {#4}
1852     }
1853     \exp_not:V \l__siunitx_number_exponent_base_tl
1854     ^
1855     {
1856       \tl_if_blank:nTF {#1}
1857       {
1858         \bool_if:NT \l__siunitx_number_implicit_plus_bool
1859         { \__siunitx_number_output_sign:N + }
1860       }
1861       { \__siunitx_number_output_sign:N #1 }
1862       \__siunitx_number_output_digits:nn { integer } {#2}
1863     }
1864   }
1865   { \exp_not:n {#4} }
1866 }

```

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

```

1867 \cs_new:Npn \__siunitx_number_output_end: { }

```

(End definition for \siunitx_number_output:N and others. These functions are documented on page 38.)

2.7 Miscellaneous tools

\l__siunitx_number_valid_tl The list of valid tokens.

```

1868 \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.

```

1869 \prg_new_protected_conditional:Npnn \siunitx_if_number:n #1
1870 { T , F , TF }
1871 {

```

```

1872 \group_begin:
1873 \bool_set_true:N \l__siunitx_number_validate_bool
1874 \bool_set_true:N \l_siunitx_number_parse_bool
1875 \siunitx_number_parse:nN {#1} \l__siunitx_number_parsed_tl
1876 \tl_if_empty:NTF \l__siunitx_number_parsed_tl
1877 {
1878 \group_end:
1879 \prg_return_false:
1880 }
1881 {
1882 \group_end:
1883 \prg_return_true:
1884 }
1885 }

```

(End definition for \siunitx_if_number:nTF. This function is documented on page 38.)

\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 1886 \prg_new_conditional:Npnn \siunitx_if_number_token:N #1
  \__siunitx_number_if_token_auxii:NN 1887 { p , T , F , TF }
  \__siunitx_number_if_token_auxiii:NN 1888 {
1889 \__siunitx_number_token_auxi:NN #1
1890 \l_siunitx_number_input_decimal_tl
1891 \l__siunitx_number_input_uncert_close_tl
1892 \l_siunitx_number_input_comparator_tl
1893 \l__siunitx_number_input_digit_tl
1894 \l_siunitx_number_input_exponent_tl
1895 \l__siunitx_number_input_ignore_tl
1896 \l__siunitx_number_input_uncert_open_tl
1897 \l_siunitx_number_input_sign_tl
1898 \l__siunitx_number_input_uncert_sign_tl
1899 \q_recursion_tail
1900 \q_recursion_stop
1901 }
1902 \cs_new:Npn \__siunitx_number_token_auxi:NN #1#2
1903 {
1904 \quark_if_recursion_tail_stop_do:Nn #2 { \prg_return_false: }
1905 \__siunitx_number_token_auxii:NN #1 #2
1906 \__siunitx_number_token_auxi:NN #1
1907 }
1908 \cs_new:Npn \__siunitx_number_token_auxii:NN #1#2
1909 {
1910 \exp_after:wN \__siunitx_number_token_auxiii:NN \exp_after:wN #1
1911 #2 \q_recursion_tail \q_recursion_stop
1912 }
1913 \cs_new:Npn \__siunitx_number_token_auxiii:NN #1#2
1914 {
1915 \quark_if_recursion_tail_stop:N #2
1916 \str_if_eq:nnT {#1} {#2}
1917 {
1918 \use_i_delimit_by_q_recursion_stop:nw
1919 {
1920 \use_i_delimit_by_q_recursion_stop:nw

```

```

1921         { \prg_return_true: }
1922     }
1923 }
1924 \__siunitx_number_token_auxiii:NN #1
1925 }

```

(End definition for \siunitx_if_number_token:NTF and others. This function is documented on page 38.)

2.8 Messages

```

1926 \msg_new:nnnn { siunitx } { invalid-number }
1927 { Invalid-number~'#1'. }
1928 {
1929     The~input~'#1'~could~not~be~parsed~as~a~number~following~the~
1930     format~defined~in~module~documentation.
1931 }

```

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.

```

1932 \keys_set:nn { siunitx }
1933 {
1934     bracket-ambiguous-numbers = true ,
1935     bracket-negative-numbers  = false ,
1936     drop-exponent             = false ,
1937     drop-uncertainty          = false ,
1938     drop-zero-decimal         = false ,
1939     evaluate-expression       = false ,
1940     exponent-base              = 10 ,
1941     exponent-mode              = input ,
1942     exponent-product           = \times ,
1943     expression                 = #1 ,
1944     fixed-exponent             = 0 ,
1945     group-digits               = all ,
1946     group-minimum-digits       = 4 ,
1947     group-separator            = \, , % (
1948     input-close-uncertainty    = ) ,
1949     input-comparators          = { <=>\approx\ge\geq\gg\le\leq\ll\sim } ,
1950     input-decimal-markers      = { ., } ,
1951     input-digits               = 0123456789 ,
1952     input-exponent-markers     = dDeE ,
1953     input-ignore                = \, , % )
1954     input-open-uncertainty     = ( , % (
1955     input-signs                 = +-\mp\pm ,
1956     input-uncertainty-signs    = \pm ,
1957     minimum-decimal-digits     = 0 ,
1958     minimum-integer-digits     = 0 ,
1959     negative-color              = , % (
1960     output-close-uncertainty    = ) ,
1961     output-decimal-marker      = . ,
1962     output-open-uncertainty     = ( , % )
1963     parse-numbers               = true ,
1964     print-implicit-plus        = false ,

```

```

1965     print-unity-mantissa      = true      ,
1966     print-zero-exponent      = false     ,
1967     retain-explicit-plus      = false     ,
1968     retain-zero-uncertainty    = false     ,
1969     round-half                 = up        ,
1970     round-minimum              = 0         ,
1971     round-mode                  = none      ,
1972     round-pad                   = true      ,
1973     round-precision             = 2        ,
1974     tight-spacing               = false     ,
1975     uncertainty-mode             = compact  ,
1976     uncertainty-separator       =
1977   }
1978 </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`, `\mbox`, `\textsubscript` and `\textsuperscript` for printing in text mode. The following packages are also required to provide the functionality detailed.

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

<code>\siunitx_print_number:n</code>
<code>\siunitx_print_number:(V x)</code>
<code>\siunitx_print_unit:n</code>
<code>\siunitx_print_unit:(V x)</code>

```
\siunitx_print_number:n {\langle material \rangle}  
\siunitx_print_unit:n {\langle material \rangle}
```

Prints the $\langle material \rangle$ according to the prevailing settings for the submodule as applicable to the $\langle type \rangle$ of content (**number** or **unit**). The $\langle material \rangle$ 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 {\langle material \rangle}  
\siunitx_print_math:n {\langle material \rangle}  
\siunitx_print_text:n {\langle material \rangle}
```

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

1.1 Key-value options

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

<code>color</code>

`color = $\langle color \rangle$`

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_...:n</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_prin_number:n</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_...:n</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-font-command	text-font-command = $\langle cmd \rangle$ Command applied to text during output, inserted after any reset of font set-up. This can therefore be used to apply non-standard font set up when printing in text mode. The standard setting is empty.
<hr/> <hr/> text-series-to-math	text-series-to-math = true false Switch to determine if the weight of the current text font should be applied (where possible) to printing in math mode. This is achieved by setting the <code>\mathversion</code> , and so will override <code>reset-math-version</code> . The mappings between text and math weight are set . The standard setting is <code>false</code> .
<hr/> <hr/> unit-color	unit-color = $\langle color \rangle$ Color to apply to units in output: the latter should be a named color defined for use with <code>\textcolor</code> . The standard setting is empty (no color).
<hr/> <hr/> unit-mode	unit-mode = match math text Selects which mode (math or text) units are printed in: a choice from the options <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_...:n</code> is called. The <code>math</code> and <code>text</code> options choose the relevant T _E X mode for printing. The standard setting is <code>math</code> .
<hr/> <hr/> series-version-mapping	series-version-mapping / $\langle weight \rangle$ = $\langle version \rangle$ 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> : <ul style="list-style-type: none"> • ul • el • l • sl • m • sb • b • eb • ub

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

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.

```
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_text_font_tl
\l_siunitx_print_math_weight_bool

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-font-command .tl_set:N =
61     \l_siunitx_print_text_font_tl ,
62   text-series-to-math .bool_set:N =
63     \l_siunitx_print_math_weight_bool ,
64   unit-color .tl_set:N =
65     \l_siunitx_print_unit_color_tl ,
66   unit-mode .choices:nn =
67     { match , math , text }
68     {
69       \tl_set_eq:NN
70         \l__siunitx_print_unit_mode_tl \l_keys_choice_tl
71     }
72 }

```

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

One set of “focussed” options.

```

\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

73 \keys_define:nn { siunitx / series-version-mapping }
74 {
75   ul . tl_set:N = \l_siunitx_print_version_ul_tl ,
76   el . tl_set:N = \l_siunitx_print_version_el_tl ,
77   l . tl_set:N = \l_siunitx_print_version_l_tl ,

```

```

78   sl . tl_set:N = \l__siunitx_print_version_sl_tl ,
79   m . tl_set:N = \l__siunitx_print_version_m_tl ,
80   sb . tl_set:N = \l__siunitx_print_version_sb_tl ,
81   b . tl_set:N = \l__siunitx_print_version_b_tl ,
82   eb . tl_set:N = \l__siunitx_print_version_eb_tl ,
83   ub . tl_set:N = \l__siunitx_print_version_ub_tl
84 }

```

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

`\siunitx_print_number:n`
`\siunitx_print_number:V`
`\siunitx_print_number:x`
`\siunitx_print_unit:n`
`\siunitx_print_unit:V`
`\siunitx_print_unit:x`
`__siunitx_print_aux:nn`

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

```

85 \cs_new_protected:Npn \siunitx_print_number:n #1
86 { \__siunitx_print_aux:nn { number } {#1} }
87 \cs_generate_variant:Nn \siunitx_print_number:n { V , x }
88 \cs_new_protected:Npn \siunitx_print_unit:n #1
89 { \__siunitx_print_aux:nn { unit } {#1} }
90 \cs_generate_variant:Nn \siunitx_print_unit:n { V , x }
91 \cs_new_protected:Npn \__siunitx_print_aux:nn #1#2
92 {
93   \tl_if_empty:cTF { l__siunitx_print_ #1 _color_tl }
94   { \use:n }
95   { \exp_args:Nv \textcolor { l__siunitx_print_ #1 _color_tl } }
96   {
97     \use:c
98     {
99       siunitx_print_
100       \tl_use:c { l__siunitx_print_ #1 _mode_tl } :n
101     }
102     {#2}
103   }
104 }

```

(End definition for `\siunitx_print_number:n`, `\siunitx_print_unit:n`, and `__siunitx_print_aux:nn`. These functions are documented on page 89.)

`\siunitx_print_match:n`

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

```

105 \cs_new_protected:Npn \siunitx_print_match:n #1
106 {
107   \mode_if_math:TF
108   { \siunitx_print_math:n {#1} }
109   { \siunitx_print_text:n {#1} }
110 }

```

(End definition for `\siunitx_print_match:n`. This function is documented on page 89.)

`__siunitx_print_replace_font:N`

A simple auxiliary for “zapping” the unit font.

```

111 \cs_new_protected:Npn \__siunitx_print_replace_font:N #1
112 {
113   \tl_if_empty:NF \l_siunitx_unit_font_tl
114   {
115     \tl_replace_all:NVn #1
116     \l_siunitx_unit_font_tl
117     { \use:n }
118   }
119 }

```

(End definition for `_siunitx_print_replace_font:N`.)

```
\c_siunitx_print_weight_uc_tl
\c_siunitx_print_weight_ecl_tl
\c_siunitx_print_weight_c_tl
\c_siunitx_print_weight_sc_tl
\c_siunitx_print_weight_sx_tl
\c_siunitx_print_weight_x_tl
\c_siunitx_print_weight_l_tl
\c_siunitx_print_weight_m_tl
\c_siunitx_print_weight_b_tl
```

Font widths where the m for weight is omitted.

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

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

Font widths with one letter.

```
122 \clist_map_inline:nn { l , m , b }
123   { \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
\_siunitx_print_math_version:nn
\_siunitx_print_math_version:Vn
\_siunitx_print_math_auxi:n
\_siunitx_print_math_auxii:n
\_siunitx_print_math_auxiii:n
\_siunitx_print_math_auxiv:n
\_siunitx_print_math_auxv:n
\_siunitx_print_math_aux:Nn
\_siunitx_print_math_aux:cn
\_siunitx_print_math_sub:n
\_siunitx_print_math_super:n
\_siunitx_print_math_script:n
\_siunitx_print_math_text:n

124 \cs_new_protected:Npn \siunitx_print_math:n #1
125   {
126     \bool_if:NTF \l__siunitx_print_math_weight_bool
127     {
128       \tl_set:Nx \l__siunitx_print_tmp_tl
129       { \exp_after:wN \_siunitx_print_extract_series:Nw \f@series ? \q_stop }
130       \tl_if_empty:NTF \l__siunitx_print_tmp_tl
131       { \_siunitx_print_math_auxi:n {#1} }
132       { \_siunitx_print_math_version:Vn \l__siunitx_print_tmp_tl {#1} }
133     }
134     { \_siunitx_print_math_auxi:n {#1} }
135   }
```

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.

```
136 \cs_new:Npn \_siunitx_print_extract_series:Nw #1#2 ? #3 \q_stop
137   {
138     \cs_if_exist:cTF { c__siunitx_print_weight_ #1#2 _tl }
139     { \_siunitx_print_convert_series:v { c__siunitx_print_weight_ #1#2 _tl } }
140     {
141       \cs_if_exist:cTF { c__siunitx_print_weight_ #1 _tl }
142       { \_siunitx_print_convert_series:v { c__siunitx_print_weight_ #1 _tl } }
143       { \_siunitx_print_convert_series:n {#1#2} }
144     }
145   }
146 \cs_new:Npn \_siunitx_print_convert_series:n #1
147   { \tl_use:c { l__siunitx_print_version_ #1 _tl } }
148 \cs_generate_variant:Nn \_siunitx_print_convert_series:n { v }
149 \cs_new_protected:Npn \_siunitx_print_math_auxi:n #1
150   {
151     \bool_if:NTF \l__siunitx_print_math_version_bool
152     { \_siunitx_print_math_version:nn { normal } {#1} }
153     { \_siunitx_print_math_auxii:n {#1} }
154   }
```

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

```

155 \cs_new_protected:Npn \__siunitx_print_math_version:nn #1#2
156 {
157   \str_if_eq:VnTF \math@version { #1 }
158   { \__siunitx_print_math_auxii:n {#2} }
159   {
160     \mode_if_math:TF
161     { \text }
162     { \use:n }
163     {
164       \mathversion {#1}
165       \__siunitx_print_math_auxii:n {#2}
166     }
167   }
168 }
169 \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.

```

170 \cs_new_protected:Npn \__siunitx_print_math_auxii:n #1
171 { \ensuremath { \__siunitx_print_math_auxiii:n {#1} } }
172 \cs_new_protected:Npn \__siunitx_print_math_auxiii:n #1
173 {
174   \bool_if:NTF \l__siunitx_print_math_family_bool
175   {
176     \str_case_e:nnF { \f@family }
177     {
178       { \rmdefault } { \__siunitx_print_math_auxv:n }
179       { \sfdefault } { \__siunitx_print_math_aux:Nn \mathsf }
180       { \ttdefault } { \__siunitx_print_math_aux:Nn \mathtt }
181     }
182     { \__siunitx_print_math_auxiv:n }
183   }
184   { \__siunitx_print_math_auxiv:n }
185   {#1}
186 }

```

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.

```

187 \cs_new_protected:Npn \__siunitx_print_math_auxiv:n #1
188 {
189   \bool_if:NTF \l__siunitx_print_math_font_bool
190   {
191     \int_case:nnF \fam
192     {

```

```

193         \c__siunitx_print_mathsf_int { \__siunitx_print_math_aux:Nn \mathsf }
194         \c__siunitx_print_mathtt_int { \__siunitx_print_math_aux:Nn \mathtt }
195     }
196     { \use:n }
197 }
198 { \__siunitx_print_math_auxv:n }
199 {#1}
200 }
201 \cs_new_protected:Npn \__siunitx_print_math_auxv:n #1
202 {
203     \bool_lazy_or:nnTF
204     { \int_compare_p:nNn \fam = { -1 } }
205     { \int_compare_p:nNn \fam = \c__siunitx_print_mathrm_int }
206     { \use:n }
207     { \mathrm }
208     {#1}
209 }

```

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

```

210 \cs_new_protected:Npx \__siunitx_print_math_aux:Nn #1#2
211 {
212     \group_begin:
213     \tl_set:Nn \exp_not:N \l__siunitx_print_tmp_tl {#2}
214     \__siunitx_print_replace_font:N \exp_not:N \l__siunitx_print_tmp_tl
215     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
216     { \char_generate:nn { '\_ } { 8 } }
217     { \exp_not:N \__siunitx_print_math_sub:n }
218     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
219     { ^ }
220     { \exp_not:N \__siunitx_print_math_super:n }
221     #1 { \exp_not:N \tl_use:N \exp_not:N \l__siunitx_print_tmp_tl }
222     \group_end:
223 }
224 \cs_generate_variant:Nn \__siunitx_print_math_aux:Nn { c }
225 \cs_new_protected:Npx \__siunitx_print_math_sub:n #1
226 {
227     \char_generate:nn { '\_ } { 8 }
228     { \exp_not:N \__siunitx_print_math_script:n {#1} }
229 }
230 \cs_new_protected:Npn \__siunitx_print_math_super:n #1
231 { ^ { \__siunitx_print_math_script:n {#1} } }
232 \cs_new_protected:Npn \__siunitx_print_math_script:n #1
233 {
234     \group_begin:
235     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
236     \__siunitx_print_replace_font:N \l__siunitx_print_tmp_tl
237     \tl_use:N \l__siunitx_print_tmp_tl
238     \group_end:
239 }

```

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

```

240 \AtBeginDocument
241 {

```

```

242 \ifpackageloaded { tex4ht }
243 {
244   \cs_set_protected:Npn \__siunitx_print_math_auxii:n #1
245   {
246     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
247     \exp_args:NNnx \tl_replace_all:Nnn \l__siunitx_print_tmp_tl
248       { ^ } { \token_to_str:N ^ }
249     \ensuremath { \exp_args:NV \__siunitx_print_math_auxiii:n \l__siunitx_print_tmp_tl }
250   }
251 }
252 { }
253 }

```

(End definition for `\siunitx_print_math:n` and others. This function is documented on page 89.)

```

\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:NnNn
  \__siunitx_print_text_scripts_two:nn
  \__siunitx_print_text_scripts_two:n
  \__siunitx_print_text_fraction:Nnn
254 \cs_new_protected:Npn \siunitx_print_text:n #1
255 {
256   \text
257   {
258     \bool_if:NT \l__siunitx_print_text_family_bool
259       { \fontfamily { \familydefault } }
260     \bool_if:NT \l__siunitx_print_text_series_bool
261       { \fontseries { \seriesdefault } }
262     \bool_if:NT \l__siunitx_print_text_shape_bool
263       { \fontshape { \shapedefault } }
264     \bool_lazy_any:nT
265       {
266         \l__siunitx_print_text_family_bool }
267         \l__siunitx_print_text_series_bool }
268         \l__siunitx_print_text_shape_bool }
269     { \selectfont }
270     \tl_use:N \l__siunitx_print_text_font_tl
271     \exp_args:NnV \tl_if_head_eq_meaning:nNTF {#1} \l_siunitx_unit_fraction_tl
272       { \__siunitx_print_text_fraction:Nnn #1 }
273       { \__siunitx_print_text_replace:n {#1} }
274   }
275 }
276 }

```

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

```

277 \cs_new_protected:Npn \__siunitx_print_text_replace:n #1
278 {
279   \group_begin:
280     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
281     \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
282     \tl_use:N \l__siunitx_print_tmp_tl
283   \group_end:
284 }
285 \cs_new_protected:Npx \__siunitx_print_text_replace:N #1
286 {
287   \__siunitx_print_replace_font:N #1
288   \exp_not:N \__siunitx_print_text_replace:NNn #1

```

```

289 \exp_not:N \mathord { }
290 \exp_not:N \pm
291 { \exp_not:N \textpm }
292 \exp_not:N \mp
293 { \exp_not:n { \ensuremath { \mp } } } }
294 -
295 { \exp_not:N \textminus }
296 \char_generate:nn { '\_ } { 8 }
297 { \exp_not:N \__siunitx_print_text_sub:n }
298 ~
299 { \exp_not:N \__siunitx_print_text_super:n }
300 \exp_not:N \q_recursion_tail
301 { ? }
302 \exp_not:N \q_recursion_stop
303 }
304 \cs_new_protected:Npn \__siunitx_print_text_replace:NNn #1#2#3
305 {
306   \quark_if_recursion_tail_stop:N #2
307   \tl_replace_all:Nnn #1 {#2} {#3}
308   \__siunitx_print_text_replace:NNn #1
309 }

```

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

```

310 \sys_if_engine_xetex:T
311 {
312   \AtBeginDocument
313   {
314     \@ifpackageloaded { bidi }
315     {
316       \cs_set_protected:Npn \__siunitx_print_text_replace:n #1
317       {
318         \group_begin:
319         \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
320         \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
321         \LRE { \tl_use:N \l__siunitx_print_tmp_tl }
322         \group_end:
323       }
324     }
325   }
326 }
327 }

```

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.

```

328 \cs_new_protected:Npn \__siunitx_print_text_sub:n #1
329 {
330   \__siunitx_print_text_scripts:NnN
331   \textsubscript {#1} \__siunitx_print_text_super:n
332 }
333 \cs_new_protected:Npn \__siunitx_print_text_super:n #1
334 {
335   \__siunitx_print_text_scripts:NnN
336   \textsuperscript {#1} \__siunitx_print_text_sub:n
337 }

```

```

338 \cs_new_protected:Npn \__siunitx_print_text_scripts:NnN #1#2#3
339 {
340   \cs_set_protected:Npn \__siunitx_print_text_scripts:
341   {
342     \if_meaning:w \l_peek_token #3
343     \exp_after:wN \__siunitx_print_text_scripts_two:NnNn
344     \else:
345     \exp_after:wN \__siunitx_print_text_scripts_one:Nn
346     \fi:
347     #1 {#2}
348   }
349   \peek_after:Nw \__siunitx_print_text_scripts:
350 }
351 \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.

```

352 \cs_new_protected:Npn \__siunitx_print_text_scripts_one:Nn #1#2
353 {
354   \group_begin:
355   \tl_set:Nn \l__siunitx_print_tmp_tl {#2}
356   \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
357   \exp_args:NNV \group_end:
358   #1 \l__siunitx_print_tmp_tl
359 }

```

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

```

360 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:NnNn #1#2#3#4
361 {
362   \cs_if_eq:NNTF #1 \textsubscript
363   { \__siunitx_print_text_scripts_two:nn {#4} {#2} }
364   { \__siunitx_print_text_scripts_two:nn {#2} {#4} }
365 }
366 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:nn #1#2
367 {
368   \group_begin:
369   \exp_not:N \m@th
370   \exp_not:N \ensuremath
371   {
372     ~ { \exp_not:N \__siunitx_print_text_scripts_two:n {#1} }
373     \char_generate:nn { '\_ } { 8 }
374     { \exp_not:N \__siunitx_print_text_scripts_two:n {#2} }
375   }
376   \group_end:
377 }
378 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:n #1
379 {
380   \mbox
381   {
382     \fontsize \sf@size \z@ \selectfont
383     \__siunitx_print_text_scripts_one:Nn \use:n {#1}
384   }
385 }

```


Fraction commands are always math mode, so we have to go back and forth: this is done after general font setting for performance reasons.

```

386 \cs_new_protected:Npn \__siunitx_print_text_fraction:Nnn #1#2#3
387 {
388   \ensuremath
389   {
390     #1
391     { \mbox { \__siunitx_print_text_replace:n {#2} } }
392     { \mbox { \__siunitx_print_text_replace:n {#3} } }
393   }
394 }

```

(End definition for `\siunitx_print_text:n` and others. This function is documented on page 89.)

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.

```

395 \keys_set:nn { siunitx }
396 {
397   color                = ,
398   mode                 = math ,
399   number-color         = ,
400   number-mode          = math ,
401   propagate-math-font = false ,
402   reset-math-version   = true ,
403   reset-text-shape     = true ,
404   reset-text-series    = true ,
405   reset-text-family    = true ,
406   text-family-to-math  = false ,
407   text-font-command    = ,
408   text-series-to-math  = false ,
409   unit-color           = ,
410   unit-mode            = math
411 }

```

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

```

412 \keys_set:nn { siunitx / series-version-mapping }
413 {
414   ul = light ,
415   el = light ,
416   l  = light ,
417   sl = light ,
418   m  = normal ,
419   sb = bold ,
420   b  = bold ,
421   eb = bold ,
422   ub = bold
423 }
424 \</package>

```

Part VII

siunitx-quantity — Quantities

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

`\siunitx_quantity:nn`

`\siunitx_quantity:nn {<number>} {<unit>}`

Parses the `<number>` and the `<unit>` as detailed for `\siunitx_number_parse:nn` and `\siunitx_unit_format:nn`, then prints the results using `\siunitx_print_unit:n`.

`\siunitx_quantity_print:nn`

`\siunitx_quantity_print:nn {<number>} {<unit>}`
`\siunitx_quantity_print:(nV|VV|xV)`

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

`allow-quantity-breaks`

`allow-quantity-breaks = true|false`

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

`prefix-mode`

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

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

`quantity-product`

`quantity-product = <tokens>`

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

`separate-uncertainty-units`

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

Specifies how units are applied when a separated uncertainty is present: a choice from `bracket`, `repeat` and `single`. The option `bracket` places brackets around the number, with the unit given after these. The option `repeat` means that the unit 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_unit:V \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_number:n { { } #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 102.)

```
\_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_number:n {#1}
142   \tl_if_blank:nF {#2}
143   {
144     \tl_use:N \l__siunitx_quantity_product_tl
145     \bool_if:NTF \l__siunitx_quantity_break_bool
146       { \penalty \binoppenalty }
147       { \nobreak }
148     \siunitx_print_unit:n {#2}
149   }
150 }
151 \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.

```
152 \keys_set:nn { siunitx }
153 {
154   allow-quantity-breaks      = false ,
155   prefix-mode                 = input ,
156   quantity-product           = \ ,
157   separate-uncertainty-units = bracket
158 }
```

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.

```
159 \bool_lazy_or:nnTF
160 { \sys_if_engine luatex_p: }
161 { \sys_if_engine xetex_p: }
162 {
```

```

163 \cs_new:Npn \__siunitx_quantity_non_latin:n #1
164 { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
165 }
166 {
167 \cs_new:Npn \__siunitx_quantity_non_latin:n #1
168 {
169 \exp_last_unbraced:Nf \__siunitx_quantity_non_latin:nnnn
170 { \char_to_utfviii_bytes:n {#1} }
171 }
172 \cs_new:Npn \__siunitx_quantity_non_latin:nnnn #1#2#3#4
173 {
174 \exp_after:wN \exp_after:wN \exp_after:wN
175 \exp_not:N \char_generate:nn {#1} { 13 }
176 \exp_after:wN \exp_after:wN \exp_after:wN
177 \exp_not:N \char_generate:nn {#2} { 13 }
178 }
179 }

```

(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.

```

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

```

(End definition for `\degree`. This function is documented on page [140](#).)

```

183 \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:Nn \arcminute
54     { \exp_not:N \ensuremath { { } ' } } }
55   }
56   \_siunitx_symbol_if_replace:NnT \arcsecond { "02BA }
57   {
58     \siunitx_declare_unit:Nn \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

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

<code>\siunitx_cell_begin:w</code>	<code>\siunitx_cell_begin:w <preamble> \ignorespaces</code>
<code>\siunitx_cell_end:</code>	<code><content></code> <code>\siunitx_cell_end:</code>

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

1.1 Key-value options

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

<code>table-align-comparator</code>	<code>table-align-comparator = true false</code>
-------------------------------------	--

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

<code>table-align-exponent</code>	<code>table-align-exponent = true false</code>
-----------------------------------	--

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

<code>table-align-text-after</code>	<code>table-align-text-after = true false</code>
-------------------------------------	--

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

<code>table-align-text-before</code>	<code>table-align-text-before = true false</code>
--------------------------------------	---

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

<code>table-align-uncertainty</code>	<code>table-align-uncertainty = true false</code>
--------------------------------------	---

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

<u>table-alignment</u>	<p><code>table-alignment = center left right</code></p> <p>Selects the alignment of all tabular content with the margins of the table cell (or other boundary). See also <code>table-number-alignment</code> and <code>table-text-alignment</code>. The standard setting is <code>center</code>.</p>
<u>table-alignment-mode</u>	<p><code>table-alignment-mode = format marker none</code></p> <p>Selects the method used to align numbers with the desired position in the cell (set by <code>table-alignment</code>). When set to <code>format</code>, a dedicated amount of space is calculated from the <code>table-format</code>. When <code>marker</code> is selected, alignment is carried out symmetrically around the decimal marker. Finally, <code>none</code> switches off all alignment: numbers are parsed and formatted but with no attempt at placement within the cell. The standard setting is <code>marker</code>.</p>
<u>table-auto-round</u>	<p><code>table-auto-round = true false</code></p> <p>Switch which determines whether numbers are rounded to fit within the <code>table-format</code> specification (if possible). The standard setting is <code>false</code>.</p>
<u>table-column-width</u>	<p><code>table-column-width = <width></code></p> <p>Sets the width of the table column used for numbers. This is only used when <code>table-fixed-width</code> is <code>true</code>.</p>
<u>table-fixed-width</u>	<p><code>table-fixed-width = true false</code></p> <p>Switch which determines whether a fixed-width column is used for numbers in tables. When <code>true</code>, the width is taken from <code>table-column-width</code>. The standard setting is <code>false</code>.</p>
<u>table-format</u>	<p><code>table-format = <format></code></p> <p>Describes the amount of space that should be reserved when <code>table-alignment-mode</code> is set to <code>format</code>. The <code><format></code> takes the same general form as input for a table cell, with the numerical parts describing how many digits to reserve space for. For example, <code>1.2e3</code> would allow space for one digit in the integer part, two in the decimal part and three in the exponent part. Signs can be allowed for using any valid input sign, so for example <code>+1.2 \pm 1.2</code> would allow for a sign, a number with one integer and two decimal digits and a uncertainty of the same size.</p>
<u>table-number-alignment</u>	<p><code>table-number-alignment = center left right</code></p> <p>Selects the alignment of numerical content with the margins of the table cell (or other boundary). See also <code>table-alignment</code> and <code>table-text-alignment</code>. The standard setting is <code>center</code>.</p>
<u>table-text-alignment</u>	<p><code>table-text-alignment = center left right</code></p> <p>Selects the alignment of non-numerical content with the margins of the table cell (or other boundary). See also <code>table-alignment</code> and <code>table-number-alignment</code>. The standard setting is <code>center</code>.</p>

2 siunitx-table implementation

Start the DocStrip guards.

```
1 <*package>
```

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

```
2 <@@=siunitx_table>
```

Scratch space.

```
\l__siunitx_table_tmp_box
```

```
\l__siunitx_table_tmp_dim
```

```
\l__siunitx_table_tmp_tl
```

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

```
4 \dim_new:N \l__siunitx_table_tmp_dim
```

```
5 \tl_new:N \l__siunitx_table_tmp_tl
```

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

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
```

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

```
\siunitx_cell_end:
```

```
7 \cs_new_protected:Npn \siunitx_cell_begin:w
```

```
8 {
```

```
9   \bool_set_false:N \l__siunitx_table_text_bool
```

```
10   \bool_if:NTF \l_siunitx_number_parse_bool
```

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

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

```
13 }
```

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

```
15 {
```

```
16   \bool_if:NF \l__siunitx_table_text_bool
```

```
17   {
```

```
18     \bool_if:NTF \l_siunitx_number_parse_bool
```

```
19       { \__siunitx_table_collect_end: }
```

```
20       { \__siunitx_table_direct_end: }
```

```
21   }
```

```
22 }
```

(End definition for \siunitx_cell_begin:w and \siunitx_cell_end:. These functions are documented on page [113](#).)

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`.) Whilst the group formed by a cell will normally tidy up `\cr`, we add an extra one as the collected material could be a tabular in itself. 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   \group_begin:
27   \tl_clear:N \l__siunitx_table_collect_tl
28   \if_false: { \fi:
29     \cs_set_protected:Npn \cr
30     {
31       \_siunitx_table_collect_loop:
32       \tex_cr:D
33     }
34     \if_false: } \fi:
35     \_siunitx_table_collect_begin:w
36   }
37 \cs_new_protected:Npn \_siunitx_table_collect_begin:w #1 \ignorespaces
38 { \_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.

```

39 \cs_new_protected:Npn \_siunitx_table_collect_loop:
40 {
41   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
42   { \_siunitx_table_collect_group:n }
43   { \_siunitx_table_collect_token:N }
44 }
45 \cs_new_protected:Npn \_siunitx_table_collect_group:n #1
46 {
47   \tl_put_right:Nn \l__siunitx_table_collect_tl { {#1} }
48   \_siunitx_table_collect_loop:
49 }
50 \cs_new_protected:Npn \_siunitx_table_collect_token:N #1
51 {
52   \_siunitx_table_collect_search:NnF #1
53   {
54     \unskip
55     \end
56   }
57   { \_siunitx_table_collect_loop: }
58   { \tabularnewline \end }

```



```

56         \relax          { \relax }
57         \tabularnewline { \tabularnewline }
58         \siunitx_cell_end: { \siunitx_cell_end: }
59     }
60     {
61         \tl_put_right:Nn \l__siunitx_table_collect_tl {#1}
62         \__siunitx_table_collect_loop:
63     }
64 }
65 \AtBeginDocument
66 {
67     \@ifpackageloaded { mdwtab }
68     {
69         \cs_set_protected:Npn \__siunitx_table_collect_token:N #1
70         {
71             \__siunitx_table_collect_search:NnF #1
72             {
73                 \@maybe@unskip { \__siunitx_table_collect_loop: }
74                 \tab@setcr { \__siunitx_table_collect_loop: }
75                 \unskip { \__siunitx_table_collect_loop: }
76                 \end { \tabularnewline \end }
77                 \relax { \relax }
78                 \tabularnewline { \tabularnewline }
79                 \siunitx_cell_end: { \siunitx_cell_end: }
80             }
81             {
82                 \tl_put_right:Nn \l__siunitx_table_collect_tl {#1}
83                 \__siunitx_table_collect_loop:
84             }
85         }
86     }
87     { }
88 }
89 \cs_new_protected:Npn \__siunitx_table_collect_search:NnF #1#2#3
90 {
91     \__siunitx_table_collect_search_aux:NNn #1
92     #2
93     #1 {#3}
94     \q_stop
95 }
96 \cs_new_protected:Npn \__siunitx_table_collect_search_aux:NNn #1#2#3
97 {
98     \token_if_eq_meaning:NNTF #1 #2
99     { \use_i_delimit_by_q_stop:nw {#3} }
100     { \__siunitx_table_collect_search_aux:NNn #1 }
101 }

```

(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` 102 `\tl_new:N \l__siunitx_table_before_tl`

`\l__siunitx_table_after_tl` 103 `\tl_new:N \l__siunitx_table_number_tl`

104 `\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, escape the group and check for expansion. We only do that if the entire content is not a brace group: there is more likely to be problematic content in the case of a header.

`_siunitx_table_collect_end:n`

`_siunitx_table_collect_end_aux:n`

`_siunitx_table_collect_end:w`

105 `\cs_new_protected:Npn _siunitx_table_collect_end:`

106 `{`

107 `\exp_args:NNV \group_end:`

108 `_siunitx_table_collect_end:n \l__siunitx_table_collect_tl`

109 `\exp_args:NV _siunitx_table_split:nNNN`

110 `\l__siunitx_table_collect_tl`

111 `\l__siunitx_table_before_tl`

112 `\l__siunitx_table_number_tl`

113 `\l__siunitx_table_after_tl`

114 `\tl_if_empty:NTF \l__siunitx_table_number_tl`

115 `{ _siunitx_table_print_text:V \l__siunitx_table_before_tl }`

116 `{`

117 `_siunitx_table_print:VVV`

118 `\l__siunitx_table_before_tl`

119 `\l__siunitx_table_number_tl`

120 `\l__siunitx_table_after_tl`

121 `}`

122 `}`

123 `\cs_new_protected:Npn _siunitx_table_collect_end:n #1`

124 `{`

125 `\str_if_eq:eeTF { \exp_not:n {#1} }`

126 `{ { _siunitx_table_collect_end_aux:n {#1} } }`

127 `{ \tl_set:Nn }`

128 `{ \protected@edef }`

129 `\l__siunitx_table_collect_tl {#1}`

130 `}`

131 `\cs_new:Npn _siunitx_table_collect_end_aux:n #1`

132 `{ \exp_after:wN _siunitx_table_collect_end:w #1 \q_stop }`

133 `\cs_new:Npn _siunitx_table_collect_end:w #1 \q_stop`

134 `{ \exp_not:n {#1} }`

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

`_siunitx_table_split:nNNN` Splitting into parts uses the fact that numbers cannot contain groups and that we can track where we are up to based on the content of the token lists.

`_siunitx_table_split_loop:NNN`

`_siunitx_table_split_group:NNNn`

`_siunitx_table_split_token:NNNN`

135 `\cs_new_protected:Npn _siunitx_table_split:nNNN #1#2#3#4`

136 `{`

137 `\tl_clear:N #2`

138 `\tl_clear:N #3`

139 `\tl_clear:N #4`

140 `_siunitx_table_split_loop:NNN #2#3#4 #1 \q_recursion_tail \q_recursion_stop`

141 `_siunitx_table_split_tidy:N #2`

142 `_siunitx_table_split_tidy:N #4`

```

143 }
144 \cs_new_protected:Npn \__siunitx_table_split_loop:NNN #1#2#3
145 {
146   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
147   { \__siunitx_table_split_group:NNNn #1#2#3 }
148   { \__siunitx_table_split_token:NNNN #1#2#3 }
149 }
150 \cs_new_protected:Npn \__siunitx_table_split_group:NNNn #1#2#3#4
151 {
152   \tl_if_empty:NTF #2
153   { \tl_put_right:Nn #1 { {#4} } }
154   { \tl_put_right:Nn #3 { {#4} } }
155   \__siunitx_table_split_loop:NNN #1#2#3
156 }
157 \cs_new_protected:Npn \__siunitx_table_split_token:NNNN #1#2#3#4
158 {
159   \quark_if_recursion_tail_stop:N #4
160   \tl_if_empty:NTF \l__siunitx_table_after_tl
161   {
162     \siunitx_if_number_token:NTF #4
163     { \tl_put_right:Nn #2 {#4} }
164     {
165       \tl_if_empty:NTF #2
166       { \tl_put_right:Nn #1 {#4} }
167       { \tl_put_right:Nn #3 {#4} }
168     }
169   }
170   { \tl_put_right:Nn #3 {#4} }
171   \__siunitx_table_split_loop:NNN #1#2#3
172 }

```

(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).

```

173 \cs_new_protected:Npn \__siunitx_table_split_tidy:N #1
174 {
175   \tl_if_empty:NF #1
176   { \__siunitx_table_split_tidy:NV #1 #1 }
177 }
178 \cs_new_protected:Npn \__siunitx_table_split_tidy:Nn #1#2
179 {
180   \str_if_eq:onT { \exp_after:wN { \use:n #2 } } {#2}
181   { \tl_set:No #1 { \use:n #2 } }
182 }
183 \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.

```

184 \cs_new_protected:Npn \__siunitx_table_skip:n #1
185 {
186   \skip_horizontal:n {#1}
187   \tex_kern:D \c_zero_skip
188 }

```

(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
189 \keys_define:nn { siunitx }
190 {
191   table-column-width .dim_set:N =
192     \l_siunitx_table_column_width_dim ,
193   table-fixed-width .bool_set:N =
194     \l_siunitx_table_fixed_width_bool
195 }

```

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

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

```

\_siunitx_table_align_left:n
\_siunitx_table_align_right:n
\_siunitx_table_align_auxi:nn
\_siunitx_table_align_auxii:nn
196 \cs_new_protected:Npn \_siunitx_table_align_center:n #1
197 { \_siunitx_table_align_auxi:nn {#1} { Opt~plus~0.5fill } }
198 \cs_new_protected:Npn \_siunitx_table_align_left:n #1
199 { \_siunitx_table_align_auxi:nn {#1} { Opt } }
200 \cs_new_protected:Npn \_siunitx_table_align_right:n #1
201 { \_siunitx_table_align_auxi:nn {#1} { Opt~plus~1fill } }
202 \cs_new_protected:Npn \_siunitx_table_align_auxi:nn #1#2
203 {
204   \bool_if:NTF \l_siunitx_table_fixed_width_bool
205     { \hbox_to_wd:nn \l_siunitx_table_column_width_dim }
206     { \use:n }
207     {
208       \_siunitx_table_skip:n {#2}
209       #1
210       \_siunitx_table_skip:n { Opt~plus~1fill - #2 }
211     }

```

```

212 }
213 \AtBeginDocument
214 {
215   \ifpackageloaded { colortbl }
216   {
217     \cs_new_eq:NN
218       \__siunitx_table_align_auxii:nn
219       \__siunitx_table_align_auxi:nn
220     \cs_set_protected:Npn \__siunitx_table_align_auxi:nn #1#2
221     {
222       \__siunitx_table_skip:n{ Opt~plus~-0.5fill }
223       \__siunitx_table_align_auxii:nn {#1} {#2}
224       \__siunitx_table_skip:n { Opt~plus~-0.5fill }
225     }
226   }
227   { }
228 }

```

(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.

```

229 \keys_define:nn { siunitx }
230 {
231   table-text-alignment .choices:nn =
232     { center , left , right }
233     { \tl_set:Nn \l__siunitx_table_align_text_tl {#1} } ,
234   }
235 \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:n
\_siunitx_table_print_text:V
236 \cs_new_protected:Npn \__siunitx_table_print_text:n #1
237 {
238   \bool_set_true:N \l__siunitx_table_text_bool
239   \use:c { \__siunitx_table_align_ \l__siunitx_table_align_text_tl :n } {#1}
240 }
241 \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
242 \box_new:N \l__siunitx_table_integer_box
243 \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.

```

244 \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.

```

245 \cs_new:Npn \_siunitx_table_cleanup_decimal:w
246   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
247   { #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.

```

248 \cs_new_protected:Npn \_siunitx_table_center_marker:
249   {
250     \dim_compare:nNnTF
251       { \box_wd:N \l__siunitx_table_integer_box }
252       > { \box_wd:N \l__siunitx_table_decimal_box }
253     {
254       \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
255         { \box_wd:N \l__siunitx_table_integer_box }
256       {
257         \hbox_unpack:N \l__siunitx_table_decimal_box
258         \__siunitx_table_fil:
259       }
260     }
261     {
262       \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
263         { \box_wd:N \l__siunitx_table_decimal_box }
264       {
265         \__siunitx_table_fil:
266         \hbox_unpack:N \l__siunitx_table_integer_box
267       }
268     }
269   }

(End definition for \_siunitx_table_center_marker:.)

```

`\l__siunitx_table_auto_round_bool` Options for tables with defined space.

`\l__siunitx_table_align_mode_tl`

`\l__siunitx_table_align_number_tl`

```

270 \keys_define:nn { siunitx }
271   {
272     table-alignment .meta:n =
273       { table-number-alignment = #1 , table-text-alignment = #1 },
274     table-alignment-mode .choices:nn =
275       { none , format , marker }
276       { \tl_set_eq:NN \l__siunitx_table_align_mode_tl \l_keys_choice_tl } ,
277     table-auto-round .bool_set:N =
278       \l__siunitx_table_auto_round_bool ,
279     table-format .code:n =
280       {
281         \__siunitx_table_split:nNNN {#1}
282         \l__siunitx_table_before_model_tl

```

```

283         \l__siunitx_table_model_tl
284         \l__siunitx_table_after_model_tl
285         \exp_args:NV \__siunitx_table_generate_model:n \l__siunitx_table_model_tl
286         \tl_set:Nn \l__siunitx_table_align_mode_tl { format }
287     } ,
288     table-number-alignment .choices:nn =
289     { center , left , right }
290     { \tl_set_eq:NN \l__siunitx_table_align_number_tl \l_keys_choice_tl }
291 }
292 \tl_new:N \l__siunitx_table_align_mode_tl
293 \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
294 \tl_new:N \l__siunitx_table_format_tl
295 \tl_new:N \l__siunitx_table_before_model_tl
296 \tl_new:N \l__siunitx_table_model_tl
297 \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. Notice that the uncertainty part needs to allow for the case where we cross the decimal.

```

298 \cs_new_protected:Npn \__siunitx_table_generate_model:n #1
299 {
300     \group_begin:
301     \bool_set_true:N \l_siunitx_number_parse_bool
302     \keys_set:nn { siunitx } { retain-explicit-plus = true }
303     \siunitx_number_parse:n {#1} \l__siunitx_table_format_tl
304     \exp_args:NNNV \group_end:
305     \tl_set:Nn \l__siunitx_table_format_tl \l__siunitx_table_format_tl
306     \tl_if_empty:NF \l__siunitx_table_format_tl
307     {
308         \exp_after:wN \__siunitx_table_generate_model:nnnnnnn
309         \l__siunitx_table_format_tl
310     }
311 }
312 \cs_new_protected:Npn \__siunitx_table_generate_model:nnnnnnn #1#2#3#4#5#6#7
313 {
314     \tl_set:Nx \l__siunitx_table_model_tl
315     {
316         \exp_not:n { {#1} {#2} }
317         { \prg_replicate:nn {#3} { 8 } }
318         { \prg_replicate:nn { 0 #4 } { 8 } }
319         {
320             \tl_if_blank:NF {#5}
321             {
322                 \use:c { __siunitx_table_generate_model_ \tl_head:n {#5} :nnw }
323                 {#4} #5

```

```

324         }
325     }
326     \exp_not:n { #{#6} }
327     {
328         \int_compare:nNnTF {#7} = 0
329         { 0 }
330         { \prg_replicate:nn {#7} { 8 } }
331     }
332 }
333 }
334 \cs_new:Npn \__siunitx_table_generate_model_S:nnw #1#2#3
335 {
336     { S }
337     {
338         \exp_args:Nff \__siunitx_table_generate_model_S:nnn
339         { \tl_count:n {#1} } { \tl_count:n {#3} }
340         {#3}
341     }
342 }
343 \cs_new:Npn \__siunitx_table_generate_model_S:nnn #1#2#3
344 {
345     \prg_replicate:nn
346     {
347         \int_compare:nNnTF {#2} > {#1}
348         {
349             \str_range:nnn {#3} { 1 } {#1}
350             +
351             \str_range:nnn {#3} { 1 + #1 } {#2}
352         }
353         {#3}
354     }
355     { 8 }
356 }

```

(End definition for __siunitx_table_generate_model:n and others.)

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.

<pre> __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:nnnnnn __siunitx_table_direct_format:w __siunitx_table_direct_format_switch: __siunitx_table_direct_format_end: __siunitx_table_direct_none: __siunitx_table_direct_none_end: </pre>	<p>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.</p> <pre> 357 \cs_new_protected:Npn __siunitx_table_direct_begin: 358 { __siunitx_table_direct_begin:w } 359 \cs_new_protected:Npn __siunitx_table_direct_begin:w #1 \ignorespaces 360 { 361 #1 362 \peek_catcode_ignore_spaces:NTF \c_group_begin_token 363 { __siunitx_table_print_text:n } 364 { 365 \m@th 366 \use:c { __siunitx_table_direct_ \l__siunitx_table_align_mode_tl : } 367 } </pre>
---	---


```

368 }
369 \cs_new_protected:Npn \__siunitx_table_direct_end:
370 { \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.

```

371 \cs_new_protected:Npn \__siunitx_table_direct_marker:
372 {
373   \hbox_set:Nn \l__siunitx_table_tmp_box
374     { \ensuremath { \mathord { \l_siunitx_number_output_decimal_tl } } }
375   \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
376     { \box_wd:N \l__siunitx_table_tmp_box }
377   { \__siunitx_table_fil: }
378   \hbox_set:Nw \l__siunitx_table_integer_box
379     \c_math_toggle_token
380     \tl_map_inline:Nn \l_siunitx_number_input_decimal_tl
381       {
382         \char_set_active_eq:NN ##1 \__siunitx_table_direct_marker_switch:
383         \char_set_mathcode:nn { '##1 } { "8000 }
384       }
385 }
386 \cs_new_protected:Npn \__siunitx_table_direct_marker_switch:
387 {
388   \c_math_toggle_token
389   \hbox_set_end:
390   \hbox_set:Nw \l__siunitx_table_decimal_box
391     \c_math_toggle_token
392     \l_siunitx_number_output_decimal_tl
393 }
394 \cs_new_protected:Npn \__siunitx_table_direct_marker_end:
395 {
396   \c_math_toggle_token
397   \hbox_set_end:
398   \__siunitx_table_center_marker:
399   \box_use_drop:N \l__siunitx_table_integer_box
400   \box_use_drop:N \l__siunitx_table_decimal_box
401 }

```

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

```

402 \cs_new_protected:Npn \__siunitx_table_direct_format:
403 {
404   \tl_set:Nx \l__siunitx_table_tmp_tl
405     { \siunitx_number_output:NN \l__siunitx_table_model_tl \q_nil }
406   \exp_after:wN \__siunitx_table_direct_format_aux:w
407     \l__siunitx_table_tmp_tl \q_stop
408 }
409 \cs_new_protected:Npn \__siunitx_table_direct_format_aux:w
410   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_stop
411 {
412   \hbox_set:Nn \l__siunitx_table_tmp_box
413     { \ensuremath { \__siunitx_table_cleanup_decimal:w #4 } }
414   \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box

```

```

415 { \box_wd:N \l__siunitx_table_tmp_box }
416 { \__siunitx_table_fil: }
417 \hbox_set:Nn \l__siunitx_table_tmp_box { \ensuremath { #1#2#3 } }
418 \hbox_set_to_wd:Nnw \l__siunitx_table_integer_box
419 { \box_wd:N \l__siunitx_table_tmp_box }
420 \c_math_toggle_token
421 \tl_map_inline:Nn \l_siunitx_number_input_decimal_tl
422 {
423   \char_set_active_eq:NN ##1 \__siunitx_table_direct_format_switch:
424   \char_set_mathcode:nn { '##1 } { "8000 }
425 }
426 \__siunitx_table_fil:
427 }
428 \cs_new_protected:Npn \__siunitx_table_direct_format_switch:
429 {
430   \c_math_toggle_token
431   \hbox_set_end:
432   \hbox_set_to_wd:Nnw \l__siunitx_table_decimal_box
433   { \box_wd:N \l__siunitx_table_decimal_box }
434   \c_math_toggle_token
435   \mathord { \l_siunitx_number_output_decimal_tl }
436 }
437 \cs_new_protected:Npn \__siunitx_table_direct_format_end:
438 {
439   \c_math_toggle_token
440   \__siunitx_table_fil:
441   \hbox_set_end:
442   \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
443   {
444     \box_use_drop:N \l__siunitx_table_integer_box
445     \box_use_drop:N \l__siunitx_table_decimal_box
446   }
447 }

```

No parsing and no alignment is easy.

```

448 \cs_new_protected:Npn \__siunitx_table_direct_none: { \c_math_toggle_token }
449 \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
450 \box_new:N \l__siunitx_table_before_box
451 \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.

```

452 \dim_new:N \l__siunitx_table_carry_dim

```

(End definition for `\l__siunitx_table_carry_dim`.)

```

\l__siunitx_table_align_comparator_bool
\l__siunitx_table_align_exponent_bool
\l__siunitx_table_align_after_bool
\l__siunitx_table_align_before_bool
\l__siunitx_table_align_uncertainty_bool

```

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

```

453 \keys_define:nn { siunitx }
454 {
455   table-align-comparator .bool_set:N =
456     \l__siunitx_table_align_comparator_bool ,
457   table-align-exponent .bool_set:N =
458     \l__siunitx_table_align_exponent_bool ,
459   table-align-text-after .bool_set:N =
460     \l__siunitx_table_align_after_bool ,
461   table-align-text-before .bool_set:N =
462     \l__siunitx_table_align_before_bool ,
463   table-align-uncertainty .bool_set:N =
464     \l__siunitx_table_align_uncertainty_bool
465 }

```

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

```

\__siunitx_table_print:nnn
\__siunitx_table_print: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

```

```

466 \cs_new_protected:Npn \__siunitx_table_print:nnn #1#2#3
467 { \use:c { __siunitx_table_print_ \l__siunitx_table_align_mode_tl :nnn } {#1} {#2} {#3} }
468 \cs_generate_variant:Nn \__siunitx_table_print:nnn { VVV }

```

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.

```

469 \cs_new_protected:Npn \__siunitx_table_print_marker:nnn #1#2#3
470 {
471   \hbox_set:Nn \l__siunitx_table_before_box {#1}
472   \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { Opt }
473   {
474     \box_clear:N \l__siunitx_table_before_box
475     #1
476   }
477   \hbox_set:Nn \l__siunitx_table_after_box {#3}
478   \dim_compare:nNnTF
479     { \box_wd:N \l__siunitx_table_after_box }
480     > { \box_wd:N \l__siunitx_table_before_box }
481     {
482       \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
483         { \box_wd:N \l__siunitx_table_after_box }
484       {
485         \__siunitx_table_fil:
486         \hbox_unpack:N \l__siunitx_table_before_box
487       }
488     }
489     {
490       \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
491         { \box_wd:N \l__siunitx_table_before_box }
492       {
493         \hbox_unpack:N \l__siunitx_table_after_box

```

```

494         \_siunitx_table_fil:
495     }
496 }
497 \box_use_drop:N \l__siunitx_table_before_box
498 \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
499 \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
500 \tl_set:Nx \l__siunitx_table_tmp_tl
501 { \siunitx_number_output:NN \l__siunitx_table_tmp_tl \q_nil }
502 \exp_after:wN \_siunitx_table_print_marker:w
503 \l__siunitx_table_tmp_tl \q_stop
504 \box_use_drop:N \l__siunitx_table_after_box
505 }
506 \cs_new_protected:Npn \_siunitx_table_print_marker:w
507 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_stop
508 {
509     \hbox_set:Nn \l__siunitx_table_integer_box
510     { \siunitx_print_number:n { #1#2#3 } }
511     \hbox_set:Nn \l__siunitx_table_decimal_box
512     {
513         \siunitx_print_number:x
514         { \_siunitx_table_print_marker_aux:w #4 }
515     }
516     \_siunitx_table_center_marker:
517     \box_use_drop:N \l__siunitx_table_integer_box
518     \box_use_drop:N \l__siunitx_table_decimal_box
519 }
520 \cs_new:Npn \_siunitx_table_print_marker_aux:w
521 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
522 {
523     \exp_not:n {#1#2#3#4#5}
524     \tl_if_blank:nT {#1#2#3#4#5} { { } }
525     \exp_not:n {#6#7}
526 }

```

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.

```

527 \cs_new_protected:Npn \_siunitx_table_print_format:nnn #1#2#3
528 {
529     \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_before_model_tl }
530     \hbox_set:Nn \l__siunitx_table_before_box {#1}
531     \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { 0pt }
532     {
533         \box_clear:N \l__siunitx_table_before_box
534         #1
535     }
536     \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
537     { \box_wd:N \l__siunitx_table_tmp_box }
538     {
539         \_siunitx_table_fil:
540         \hbox_unpack:N \l__siunitx_table_before_box
541     }
542     \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl

```

```

543 \group_begin:
544   \bool_if:NT \l__siunitx_table_auto_round_bool
545   {
546     \exp_args:Nx \keys_set:nn { siunitx }
547     {
548       round-mode      = places ,
549       round-pad       = true   ,
550       round-precision =
551         \exp_after:wN \__siunitx_table_print_format:nnnnnn
552         \l__siunitx_table_format_tl
553     }
554   }
555   \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
556   \exp_args:NNNV \group_end:
557   \tl_set:Nn \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
558   \tl_set:Nx \l__siunitx_table_tmp_tl
559   {
560     \siunitx_number_output:NN \l__siunitx_table_model_tl \q_nil
561     \exp_not:N \q_mark
562     \siunitx_number_output:NN \l__siunitx_table_tmp_tl \q_nil
563   }
564   \exp_after:wN \__siunitx_table_print_format_auxi:w
565   \l__siunitx_table_tmp_tl \q_stop
566   \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_after_model_tl }
567   \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
568   { \box_wd:N \l__siunitx_table_tmp_box + \l__siunitx_table_carry_dim }
569   {
570     \bool_if:NT \l__siunitx_table_align_after_bool
571     { \skip_horizontal:n { \l__siunitx_table_carry_dim } }
572     #3
573     \__siunitx_table_fil:
574   }
575   \use:c { __siunitx_table_align \l__siunitx_table_align_number_tl :n }
576   {
577     \box_use_drop:N \l__siunitx_table_before_box
578     \box_use_drop:N \l__siunitx_table_integer_box
579     \box_use_drop:N \l__siunitx_table_decimal_box
580     \box_use_drop:N \l__siunitx_table_after_box
581   }
582 }
583 \cs_new:Npn \__siunitx_table_print_format:nnnnnn #1#2#3#4#5#6#7
584 { 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).

```

585 \cs_new_protected:Npn \__siunitx_table_print_format_auxi:w
586   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
587   {
588     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1}
589     \bool_if:NTF \l__siunitx_table_align_before_bool
590     {
591       \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
592       { \box_wd:N \l__siunitx_table_tmp_box }

```

```

593     {
594       \__siunitx_table_fil:
595       \tl_if_blank:nF {#3}
596       { \siunitx_print_number:n {#3} }
597     }
598   }
599   {
600     \__siunitx_table_print_format_box:Nn \l__siunitx_table_integer_box {#3}
601     \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
602     {
603       \box_wd:N \l__siunitx_table_before_box
604       + \box_wd:N \l__siunitx_table_tmp_box
605       - \box_wd:N \l__siunitx_table_integer_box
606     }
607     {
608       \__siunitx_table_fil:
609       \hbox_unpack:N \l__siunitx_table_before_box
610     }
611   }
612   \__siunitx_table_print_format_auxii:w #2 \q_mark #4 \q_stop
613 }

```

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`”.)

```

614 \cs_new_protected:Npn \__siunitx_table_print_format_auxii:w
615   #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
616   {
617     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
618     \bool_lazy_and:nnTF
619     { \l__siunitx_table_align_comparator_bool }
620     { \dim_compare_p:nNn { \box_wd:N \l__siunitx_table_integer_box } > { Opt } }
621     {
622       \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
623       {
624         \box_wd:N \l__siunitx_table_integer_box
625         + \box_wd:N \l__siunitx_table_tmp_box
626       }
627       {
628         \hbox_unpack:N \l__siunitx_table_integer_box
629         \__siunitx_table_fil:
630         \siunitx_print_number:n {#4#5}
631       }
632     }
633     {
634       \bool_if:NTF \l__siunitx_table_align_before_bool
635       {
636         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
637         {
638           \box_wd:N \l__siunitx_table_integer_box
639           + \box_wd:N \l__siunitx_table_tmp_box
640         }

```

```

641         {
642             \_siunitx_table_fil:
643             \hbox_unpack:N \l__siunitx_table_integer_box
644             \siunitx_print_number:n {#4#5}
645         }
646     }
647     {
648         \dim_set:Nn \l__siunitx_table_tmp_dim
649         { \box_wd:N \l__siunitx_table_integer_box }
650         \hbox_set:Nn \l__siunitx_table_integer_box
651         {
652             \hbox_unpack:N \l__siunitx_table_integer_box
653             \siunitx_print_number:n {#4#5}
654         }
655         \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
656         {
657             \box_wd:N \l__siunitx_table_before_box
658             + \box_wd:N \l__siunitx_table_tmp_box
659             + \l__siunitx_table_tmp_dim
660             - \box_wd:N \l__siunitx_table_integer_box
661         }
662         {
663             \_siunitx_table_fil:
664             \hbox_unpack:N \l__siunitx_table_before_box
665         }
666     }
667 }
668 \_siunitx_table_print_format_auxiii:w #3 \q_mark #6 \q_stop
669 }

```

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.

```

670 \cs_new_protected:Npn \_siunitx_table_print_format_auxiii:w
671 #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
672 {
673     \_siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
674     \_siunitx_table_print_format_box:Nn \l__siunitx_table_decimal_box {#4#5}
675     \dim_set:Nn \l__siunitx_table_carry_dim
676     {
677         \box_wd:N \l__siunitx_table_tmp_box
678         - \box_wd:N \l__siunitx_table_decimal_box
679     }
680     \_siunitx_table_print_format_auxiv:w #3 \q_mark #6 \q_stop
681 }

```

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.

```

682 \cs_new_protected:Npn \_siunitx_table_print_format_auxiv:w
683 #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
684 {
685     \tl_if_blank:nTF {#1}
686     { \_siunitx_table_print_format_auxv:w }
687     { \_siunitx_table_print_format_auxvi:w }

```

```

688      #1#2 \q_mark #3#4 \q_stop
689    }
690 \cs_new_protected:Npn \__siunitx_table_print_format_auxv:w
691   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
692   #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
693   { \__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.

```

694 \cs_new_protected:Npn \__siunitx_table_print_format_auxvi:w
695   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
696   #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
697   {
698     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2#3 }
699     \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
700     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #5#6#7 }
701     \__siunitx_table_print_format_after:N \l__siunitx_table_align_uncertainty_bool
702     \__siunitx_table_print_format_auxvii:w #4 \q_mark #8 \q_stop
703   }

```

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.

```

704 \cs_new_protected:Npn \__siunitx_table_print_format_auxvii:w
705   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
706   {
707     \tl_if_blank:nF {#2}
708     {
709       \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2 }
710       \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
711       \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #3#4 }
712       \__siunitx_table_print_format_after:N \l__siunitx_table_align_exponent_bool
713     }
714   }

```

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

```

715 \cs_new_protected:Npn \__siunitx_table_print_format_box:Nn #1#2
716   {
717     \hbox_set:Nn #1
718     {
719       \tl_if_blank:nF {#2}
720       { \siunitx_print_number:n {#2} }
721     }
722   }

```

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

```

723 \cs_new_protected:Npn \__siunitx_table_print_format_after:N #1
724   {
725     \bool_if:NTF #1
726     {
727       \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
728       {
729         \box_wd:N \l__siunitx_table_decimal_box
730         + \l__siunitx_table_carry_dim

```



```

731         + \box_wd:N \l__siunitx_table_tmp_box
732     }
733     {
734         \hbox_unpack:N \l__siunitx_table_decimal_box
735         \__siunitx_table_fil:
736         \hbox_unpack:N \l__siunitx_table_tmp_box
737     }
738     \dim_set:Nn \l__siunitx_table_carry_dim
739     {
740         \l__siunitx_table_tmp_dim
741         - \box_wd:N \l__siunitx_table_tmp_box
742     }
743 }
744 {
745     \hbox_set:Nn \l__siunitx_table_decimal_box
746     {
747         \hbox_unpack:N \l__siunitx_table_decimal_box
748         \hbox_unpack:N \l__siunitx_table_tmp_box
749     }
750     \dim_add:Nn \l__siunitx_table_carry_dim
751     {
752         \l__siunitx_table_tmp_dim
753         - \box_wd:N \l__siunitx_table_tmp_box
754     }
755 }
756 }

```

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

```

757 \cs_new_protected:Npn \__siunitx_table_print_none:nnn #1#2#3
758 {
759     \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
760     {
761         #1
762         \siunitx_number_format:nN {#2} \l__siunitx_table_tmp_tl
763         \siunitx_print_number:V \l__siunitx_table_tmp_tl
764         #3
765     }
766 }

```

(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.

```

767 \keys_set:nn { siunitx }
768 {
769     table-align-comparator = true ,
770     table-align-exponent  = true ,
771     table-align-text-after = true ,
772     table-align-text-before = true ,
773     table-align-uncertainty = true ,
774     table-alignment       = center ,

```

```

775     table-auto-round      = false ,
776     table-column-width    = Opt   ,
777     table-fixed-width     = false ,
778     table-format          = 2.2   ,
779     table-number-alignment = center ,
780     table-text-alignment  = center ,

```

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

```

781     table-alignment-mode   = marker
782   }
783 \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`, `_` and `\,` are used by the standard module settings. For the display of colored (highlighted) and cancelled units, the commands `\textcolor` and `\cancel` are assumed to be available.

1 Formatting units

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

`\siunitx_unit_format:nN {<units>} <tl var>`

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

For example,

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

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

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

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

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

`\l_siunitx_unit_symbolic_seq`

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

`\l_siunitx_unit_seq`

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

3 Per-unit options

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

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

4 Units in (PDF) strings

`\siunitx_unit_pdfstring_context:` `\group_begin:`
 `\siunitx_unit_pdfstring_context:`
 $\langle Expansion\ context \rangle$ $\langle units \rangle$
 `\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 used within a surrounding group as shown in the example.

5 Pre-defined symbolic unit components

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

Broadly, the pre-defined units are those defined by the BIPM in the documentation for the *International System of Units* (SI) [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>	

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

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

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

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

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

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

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

`\percent`

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

`\square`
`\cubic`

`\square` $\langle prefix \rangle \langle unit \rangle$
`\cubic` $\langle prefix \rangle \langle unit \rangle$

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

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

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

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

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 $\backslash power$, and will work correctly if the $\backslash power$ itself is negative. In literal mode $\backslash per$ will print a slash (“/”).

<hr/>	$\backslash cancel \langle prefix \rangle \langle unit \rangle \langle power \rangle$
-------	---

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

<hr/>	$\backslash highlight \{ \langle color \rangle \} \langle prefix \rangle \langle unit \rangle \langle power \rangle$
-------	--

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

<hr/>	$\langle prefix \rangle \langle unit \rangle \langle power \rangle \backslash of \{ \langle qualifier \rangle \}$
-------	---

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

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

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

5.1 Key-value options

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

<hr/>	$\text{bracket-unit-denominator} = \text{true false}$
-------	---

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

<u>extract-mass-in-kilograms</u>	<p><code>extract-mass-in-kilograms = true false</code></p> <p>Determines whether prefix extraction treats kilograms as a base unit; when set false, grams are used. The standard setting is true.</p>
<u>forbid-literal-units</u>	<p><code>forbid-literal-units = true false</code></p> <p>Switch which determines if literal units are allowed when parsing is active; does not apply when parse-units is false.</p>
<u>fraction-command</u>	<p><code>fraction-command = \langlecommand\rangle</code></p> <p>Command used to create fractional output when per-mode is set to fraction. The standard setting is <code>\frac</code>.</p>
<u>inter-unit-product</u>	<p><code>inter-unit-product = \langleseparator\rangle</code></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>
<u>parse-units</u>	<p><code>parse-units = true false</code></p> <p>Determines whether parsing of unit symbols is attempted or literal mode is used directly. The standard setting is true.</p>
<u>per-mode</u>	<p><code>per-mode =</code> <code>fraction power power-positive-first repeated-symbol symbol symbol-or-fraction</code></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>
<u>per-symbol</u>	<p><code>per-symbol = \langlesymbol\rangle</code></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>
<u>qualifier-mode</u>	<p><code>qualifier-mode = bracket combine phrase subscript</code></p> <p>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.</p>
<u>qualifier-phrase</u>	<p><code>qualifier-phrase = \langlephrase\rangle</code></p> <p>Defines the \langle<i>phrase</i>\rangle used when qualifier-mode is set to phrase.</p>

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

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

<code>unit-font-command</code>	<code>unit-font-command = <command></code>
--------------------------------	--

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

6 siunitx-unit implementation

Start the DocStrip guards.

```
1 <*package>
```

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

```
2 <@@=siunitx_unit>
```

6.1 Initial set up

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

Variants not provided by expl3.

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

<code>\l__siunitx_unit_tmp_fp</code>	Scratch space.
--------------------------------------	----------------

<code>\l__siunitx_unit_tmp_int</code>	<pre>4 \fp_new:N \l__siunitx_unit_tmp_fp</pre>
<code>\l__siunitx_unit_tmp_tl</code>	<pre>5 \int_new:N \l__siunitx_unit_tmp_int</pre>
	<pre>6 \tl_new:N \l__siunitx_unit_tmp_tl</pre>

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

<code>\c__siunitx_unit_math_subscript_tl</code>	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`.)

<code>\l__siunitx_unit_parsing_bool</code>	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`.)

<code>\l__siunitx_unit_test_bool</code>	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 138.)

`\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 138.)

`__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:Nn` 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:Nn #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:Nn`. This function is documented on page [137](#).)

`\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.

```

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 137.)

`\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 137.)

`\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 137.)

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 138.)

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 141.)

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

```

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

```

(End definition for `\cancel` and `\highlight`. These functions are documented on page 141.)

\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 141.)

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

```

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

```

```

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

```

(End definition for `\raiseto` and `\tothe`. These functions are documented on page 141.)

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.

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

```

\l_siunitx_unit_font_tl
\l__siunitx_unit_product_tl
\l__siunitx_unit_mass_kilogram_bool

```

```

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

```

(End definition for `\l_siunitx_unit_font_tl`, `\l__siunitx_unit_product_tl`, and `\l__siunitx_unit_mass_kilogram_bool`. This variable is documented on page 137.)

```
\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:nnN
\siunitx_unit_format_multiply:nnN
\siunitx_unit_format_multiply_extract_prefixes:nnN
\siunitx_unit_format_multiply_combine_exponent:nnnN
\__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 not 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 135.)

6.6 Formatting literal units

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

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

```

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

```

(End definition for `__siunitx_unit_literal_power:nn`.)

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

```

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

```

(End definition for `__siunitx_unit_literal_special:nN`.)

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

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

```

\__siunitx_unit_format_literal:n
\__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
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 \l__siunitx_unit_format_literal_superscript:
224 \char_set_active_eq:NN _
225 \exp_not:N \l__siunitx_unit_format_literal_subscript:
226 \char_set_active_eq:NN \exp_not:N ~
227 \exp_not:N \l__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 \l__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 \l__siunitx_unit_format_literal_subscript: { \c__siunitx_unit_math_subscript_tl }
246 \cs_new:Npn \l__siunitx_unit_format_literal_superscript: { ^ }
247 \cs_new:Npn \l__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 \l__siunitx_unit_format_literal_auxi:w #1 .
249 {
250 \quark_if_recursion_tail_stop:n {#1}
251 \l__siunitx_unit_format_literal_auxii:n {#1}
252 \tl_set_eq:NN \l__siunitx_unit_separator_tl \l__siunitx_unit_product_tl
253 \l__siunitx_unit_format_literal_auxi:w
254 }
255 \cs_set_protected:Npn \l__siunitx_unit_format_literal_auxii:n #1
256 {
257 \l__siunitx_unit_format_literal_auxiii:w
258 #1 / \q_recursion_tail / \q_recursion_stop
259 }
260 \cs_new_protected:Npn \l__siunitx_unit_format_literal_auxiii:w #1 /
261 {
262 \quark_if_recursion_tail_stop:n {#1}
263 \l__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:V
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 138.)

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-1** 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`
`\l_siunitx_unit_per_bool`
`\l_siunitx_unit_position_int` Parsing units requires a small number of variables are available: a `prop` for the parsed units themselves, a `bool` to indicate if `\per` is active and an `int` to track how many units have be parsed.

```

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:nxxx { 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_fraction_tl
\l_siunitx_unit_denominator_bracket_bool
\l_siunitx_unit_forbid_literal_bool
\l_siunitx_unit_parse_bool
\l_siunitx_unit_per_symbol_tl
\l_siunitx_unit_qualifier_mode_tl
\l_siunitx_unit_qualifier_phrase_tl

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_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_fraction_tl and others. This variable is documented on page 138.)

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

```

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

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
`_siunitx_unit_format_parsed_aux:n` 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 { \l__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     { \l__siunitx_unit_format_parsed_aux:n {##1} }
590   \l__siunitx_unit_format_output:
591 }
592 \l__siunitx_unit_format_finalise:
593 }
594 \cs_new_protected:Npn \l__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 \l__siunitx_unit_format_parsed: and \l__siunitx_unit_format_parsed_aux:n.)

\l__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 \l__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.

```

\__siunitx_unit_format_power_negative_aux:w
\__siunitx_unit_format_power_superscript:
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:
\_siunitx_unit_format_prefix_exp:
\_siunitx_unit_format_prefix_gram:
\_siunitx_unit_format_prefix_symbol:

```

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:
\__siunitx_unit_format_qualifier_bracket:
\__siunitx_unit_format_qualifier_combine:
\__siunitx_unit_format_qualifier_phrase:
\__siunitx_unit_format_qualifier_subscript:

```

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.

```

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:n
\__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: 916 \cs_new_protected:Npn __siunitx_unit_format_finalise:

_siunitx_unit_format_finalise_fractional:

_siunitx_unit_format_finalise_power:

```

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_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

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
\mole
\kelvin
\candela
\second
\ampere
1009 \siunitx_declare_unit:Nn \kilogram { \kilo \gram }
1010 \siunitx_declare_unit:Nn \metre { m }
1011 \siunitx_declare_unit:Nn \meter { \metre }
1012 \siunitx_declare_unit:Nn \mole { mol }
1013 \siunitx_declare_unit:Nn \second { s }
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 139.)

\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 139.)

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

```

\zepto
\atto
\femto
\pico
\nano
\micro
\milli
\centi
\deci
1018 \siunitx_declare_prefix:Nnn \yocto { -24 } { y }
1019 \siunitx_declare_prefix:Nnn \zepto { -21 } { z }
1020 \siunitx_declare_prefix:Nnn \atto { -18 } { a }
1021 \siunitx_declare_prefix:Nnn \femto { -15 } { f }
1022 \siunitx_declare_prefix:Nnn \pico { -12 } { p }
1023 \siunitx_declare_prefix:Nnn \nano { -9 } { n }
1024 \siunitx_declare_prefix:Nnn \micro { -6 } { \_siunitx_unit_non_latin:n { "03BC } }
1025 \siunitx_declare_prefix:Nnn \milli { -3 } { m }
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 139.)

\deca Now the large ones.

```

\deka
\hecto
\kilo
\mega
\giga
\tera
\peta
\exa
\zetta
\yotta
1028 \siunitx_declare_prefix:Nnn \deca { 1 } { da }
1029 \siunitx_declare_prefix:Nnn \deka { 1 } { da }
1030 \siunitx_declare_prefix:Nnn \hecto { 2 } { h }
1031 \siunitx_declare_prefix:Nnn \kilo { 3 } { k }
1032 \siunitx_declare_prefix:Nnn \mega { 6 } { M }
1033 \siunitx_declare_prefix:Nnn \giga { 9 } { G }
1034 \siunitx_declare_prefix:Nnn \tera { 12 } { T }
1035 \siunitx_declare_prefix:Nnn \peta { 15 } { P }

```

```

1036 \siunitx_declare_prefix:Nnn \exa { 18 } { E }
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 139.)

```

\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 }
\lumen 1047 \siunitx_declare_unit:Nn \katal { kat }
\lux 1048 \siunitx_declare_unit:Nn \lumen { lm }
1049 \siunitx_declare_unit:Nn \lux { lx }

```

(End definition for \becquerel and others. These functions are documented on page 140.)

```

\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 140.)

```

\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 1066 \siunitx_declare_unit:Nn \electronvolt { eV }
\litre 1067 \siunitx_declare_unit:Nn \hectare { ha }
\liter 1068 \siunitx_declare_unit:Nn \hour { h }
\minute 1069 \siunitx_declare_unit:Nn \litre { L }
\neper 1070 \siunitx_declare_unit:Nn \liter { \litre }
\tonne 1071 \siunitx_declare_unit:Nn \neper { 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 140.)

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

\arcsecond 1074 `\siunitx_declare_unit:Nx \arcminute { _siunitx_unit_non_latin:n { "02B9 } }`

\degree 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 140.)

\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 140.)

\square Basic powers.

\squared 1078 `\siunitx_declare_power:NNn \square \squared { 2 }`

\cubic 1079 `\siunitx_declare_power:NNn \cubic \cubed { 3 }`

\cubed *(End definition for \square and others. These functions are documented on page 140.)*

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

```

References

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

`\dB` Abbreviation for decibel.

`\kWh` Abbreviation for kilowatt-hours.

1 siunitx-abbreviation implementation

Start the DocStrip guards.

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

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 }
\nA 3 \siunitx_declare_unit:Nn \pA { \pico \ampere }
\uA 4 \siunitx_declare_unit:Nn \nA { \nano \ampere }
\mA 5 \siunitx_declare_unit:Nn \uA { \micro \ampere }
\kA 6 \siunitx_declare_unit:Nn \mA { \milli \ampere }
7 \siunitx_declare_unit:Nn \kA { \kilo \ampere }
```

(End definition for \A and others. These functions are documented on page 175.)

```
\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 175.)

\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 }
\kmol 19 \siunitx_declare_unit:Nn \mmol { \milli \mole }
20 \siunitx_declare_unit:Nn \kmol { \kilo \mole }

(End definition for \mol and others. These functions are documented on page 176.)

\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 176.)

\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 176.)

\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 175.)

\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
\eV
\meV
\keV
\MeV
\GeV
\TeV
\kWh

```

50 \siunitx_declare_unit:Nn \mJ { \milli \joule }
51 \siunitx_declare_unit:Nn \kJ { \kilo \joule }
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 176.)

\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 175.)

\K Temperatures.

```

68 \siunitx_declare_unit:Nn \K { \kelvin }

```

(End definition for \K. This function is documented on page 175.)

\dB

```

69 \siunitx_declare_unit:Nn \dB { \deci \bel }

```

(End definition for \dB. This function is documented on page 177.)

\F Capacitance.

```

\ff 70 \siunitx_declare_unit:Nn \F { \farad }
\pF 71 \siunitx_declare_unit:Nn \ff { \femto \farad }
\nF 72 \siunitx_declare_unit:Nn \pF { \pico \farad }
\uF 73 \siunitx_declare_unit:Nn \nF { \nano \farad }
74 \siunitx_declare_unit:Nn \uF { \micro \farad }

```

(End definition for \F and others. These functions are documented on page 177.)

\H Capacitance.

```

\mH 75 \siunitx_declare_unit:Nn \H { \henry }
\uH 76 \siunitx_declare_unit:Nn \mH { \milli \henry }
77 \siunitx_declare_unit:Nn \uH { \micro \henry }

```

(End definition for \H, \mH, and \uH. These functions are documented on page ??.)

\N Forces.

```

\mN 78 \siunitx_declare_unit:Nn \N { \newton }
\kN 79 \siunitx_declare_unit:Nn \mN { \milli \newton }
\MN 80 \siunitx_declare_unit:Nn \kN { \kilo \newton }
81 \siunitx_declare_unit:Nn \MN { \mega \newton }

```

(End definition for `\N` and others. These functions are documented on page 176.)

`\Pa` Pressures.

```
\kPa 82 \siunitx_declare_unit:Nn \Pa { \pascal }
\MPa 83 \siunitx_declare_unit:Nn \kPa { \kilo \pascal }
\GPa 84 \siunitx_declare_unit:Nn \MPa { \mega \pascal }
      85 \siunitx_declare_unit:Nn \GPa { \giga \pascal }
```

(End definition for `\Pa` and others. These functions are documented on page 177.)

`\mohm` Resistances.

```
\kohm 86 \siunitx_declare_unit:Nn \mohm { \milli \ohm }
\Mohm 87 \siunitx_declare_unit:Nn \kohm { \kilo \ohm }
      88 \siunitx_declare_unit:Nn \Mohm { \mega \ohm }
```

(End definition for `\mohm`, `\kohm`, and `\Mohm`. These functions are documented on page 177.)

`\s` Finally, times.

```
\as 89 \siunitx_declare_unit:Nn \s { \second }
\fs 90 \siunitx_declare_unit:Nn \as { \atto \second }
\ps 91 \siunitx_declare_unit:Nn \fs { \femto \second }
\ns 92 \siunitx_declare_unit:Nn \ps { \pico \second }
\us 93 \siunitx_declare_unit:Nn \ns { \nano \second }
\ms 94 \siunitx_declare_unit:Nn \us { \micro \second }
      95 \siunitx_declare_unit:Nn \ms { \milli \second }
```

(End definition for `\s` and others. These functions are documented on page 175.)

```
96 \end{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 181.)

```
\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 181.)

```
12 </package>
```

Part XIII

siunitx-command – Units as document command

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

1 Creating units as document commands

`\siunitx_command_create:`

`\siunitx_command_create:`

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

1.1 Key-value options

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

These options are all preamble-only.

`free-standing-units`

`free-standing-units = true|false`

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

`overwrite-commands`

`overwrite-commands = true|false`

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

`space-before-unit`

`space-before-unit = true|false`

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

`unit-optional-argument`

`unit-optional-argument = true|false`

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

`use-xspace`

`use-xspace = true|false`

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

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_unit:V
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 [182](#).)

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 \</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-angle-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-angle-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-angle-seconds }
55     \l_keys_value_tl
56   } ,
57   add-arc-second-zero .default:n = true ,
58   arc-separator .code:n =
59   {
60     \__siunitx_option_deprecated:nnV
61     { arc-separator }
62     { angle-separator }
63     \l_keys_value_tl
64   }
65 }
```

1.3 Combination functions options

```
66 \keys_define:nn { siunitx }
67 {
68   list-units / brackets .code:n =
69   {
70     \__siunitx_option_deprecated:nn
71     { list-units~~brackets }
72     { list-units~~bracket }
73   } ,
```

```

74 range-units / brackets .code:n =
75 {
76   \_siunitx_option_deprecated:nn
77   { range-units~==brackets }
78   { range-units~==bracket }
79 } ,
80 product-units / brackets .code:n =
81 {
82   \_siunitx_option_deprecated:nn
83   { product-units~==brackets }
84   { product-units~==bracket }
85 }
86 }

```

1.4 Command options

```

87 \keys_define:nn { siunitx }
88 {
89   overwrite-functions .code:n =
90   {
91     \_siunitx_option_deprecated:nnV
92     { overwrite-functions }
93     { overwrite-commands }
94     \l_keys_value_tl
95   } ,
96   overwrite-functions .default:n = true
97 }

```

1.5 Print options

```

98 \keys_define:nn { siunitx }
99 {
100   detect-all .code:n =
101   {
102     \_siunitx_option_deprecated:nn
103     { detect-all }
104     {
105       mode~==match , ~
106       propagate-math-font~==true , ~
107       reset-math-version~==false , ~
108       reset-text-family~==false , ~
109       reset-text-series~==false , ~
110       text-family-to-math~==true , ~
111       text-series-to-math~==true
112     }
113   } ,
114   detect-family .code:n =
115   {
116     \_siunitx_option_deprecated:nn
117     { detect-family }
118     {
119       reset-text-family~==false , ~
120       text-family-to-math~==true
121     }
122   } ,
123   detect-mode .code:n =

```



```

124 {
125   \_siunitx_option_deprecated:nn
126   { detect-mode }
127   { mode~==match }
128 } ,
129 detect-none .code:n =
130 {
131   \_siunitx_option_deprecated:nn
132   { detect-none }
133   {
134     mode~==math , ~
135     propagate-math-font~==false , ~
136     reset-math-version~==true , ~
137     reset-text-family~==true , ~
138     reset-text-series~==true , ~
139     text-family-to-math~==false , ~
140     text-series-to-math~==false
141   }
142 } ,
143 detect-shape .code:n =
144 {
145   \_siunitx_option_deprecated:nn
146   { detect-shape }
147   { reset-text-shape~==false }
148 } ,
149 detect-weight .code:n =
150 {
151   \_siunitx_option_deprecated:nn
152   { detect-weight }
153   {
154     reset-text-series~==false , ~
155     text-series-to-math~==true
156   }
157 }
158 }
159 \clist_map_inline:nn
160 {
161   detect-display-math ,
162   detect-inline-family ,
163   detect-inline-weight
164 }
165 {
166   \keys_define:nn { siunitx } { #1 .code:n = \_siunitx_option_removed:n {#1} }
167 }

```

The old font insertion options.

```

168 \clist_map_inline:nn
169 {
170   math-rm ,
171   math-sf ,
172   math-tt ,
173   number-math-rm ,
174   number-math-sf ,
175   number-math-tt ,
176   number-text-rm ,

```

```

177     number-text-sf ,
178     number-text-tt ,
179     text-rm      ,
180     text-sf      ,
181     text-tt      ,
182     unit-math-rm ,
183     unit-math-sf ,
184     unit-math-tt ,
185     unit-text-rm ,
186     unit-text-sf ,
187     unit-text-tt
188   }
189   {
190     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
191   }

```

1.6 Symbol options

```

192 \clist_map_inline:nn
193 {
194   math-angstrom ,
195   math-arcminute ,
196   math-arcsecond ,
197   math-celsius ,
198   math-degree ,
199   math-micro ,
200   math-ohm ,
201   text-angstrom ,
202   text-arcminute ,
203   text-arcsecond ,
204   text-celsius ,
205   text-degree ,
206   text-micro ,
207   text-ohm ,
208 }
209 {
210   \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
211 }

```

1.7 Number options

```

212 \keys_define:nn { siunitx }
213 {
214   separate-uncertainty .choice: ,
215   separate-uncertainty / false .code:n =
216   {
217     \__siunitx_option_deprecated:nn
218     { separate-uncertainty }
219     { uncertainty-mode~==compact }
220   } ,
221   separate-uncertainty / true .code:n =
222   {
223     \__siunitx_option_deprecated:nn
224     { separate-uncertainty }
225     { uncertainty-mode~==separate }
226   } ,

```

```

227     separate-uncertainty .default:n = true
228 }

    A small number of removed options.
229 \clist_map_inline:nn
230 {
231     input-protect-tokens ,
232     input-quotient ,
233     quotient-mode
234 }
235 {
236     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
237 }

    Options for number processing: largely removals.
238 \keys_define:nn { siunitx }
239 {
240     add-decimal-zero .choice: ,
241     add-decimal-zero / false .code:n =
242     {
243         \__siunitx_option_deprecated:nn
244         { add-decimal-zero }
245         { minimum-decimal-digits~0 }
246     } ,
247     add-decimal-zero / true .code:n =
248     {
249         \__siunitx_option_deprecated:nn
250         { add-decimal-zero }
251         { minimum-decimal-digits~1 }
252     } ,
253     add-decimal-zero .default:n = true ,
254     add-integer-zero .code:n =
255     { \__siunitx_option_removed:V \l_keys_key_tl } ,
256     close-bracket .code:n =
257     { \__siunitx_option_removed:V \l_keys_key_tl } ,
258     bracket-numbers .choice: ,
259     bracket-numbers / false .code:n =
260     {
261         \__siunitx_option_deprecated:nn
262         { bracket-numbers }
263         { bracket-ambiguous-numbers~false }
264     } ,
265     bracket-numbers / true .code:n =
266     {
267         \__siunitx_option_deprecated:nn
268         { bracket-numbers }
269         { bracket-ambiguous-numbers~true }
270     } ,
271     bracket-numbers .default:n = true ,
272     explicit-sign .code:n =
273     {
274         \str_if_eq:nnTF {#1} { + }
275         {
276             \__siunitx_option_deprecated:nn
277             { explicit-sign }

```

```

278         { print-implicit-plus==~true }
279     }
280     { \_siunitx_option_removed:V \l_keys_key_tl }
281 },
282 omit-uncertainty .code:n =
283 {
284     \_siunitx_option_deprecated:nnV
285     { omit-uncertainty }
286     { drop-uncertainty }
287     \l_keys_value_tl
288 },
289 omit-uncertainty .default:n = true ,
290 open-bracket .code:n =
291 { \_siunitx_option_removed:V \l_keys_key_tl } ,
292 retain-unity-mantissa .code:n =
293 {
294     \_siunitx_option_deprecated:nnV
295     { retain-unity-mantissa }
296     { print-unity-mantissa }
297     \l_keys_value_tl
298 },
299 retain-unity-mantissa .default:n = true ,
300 retain-zero-exponent .code:n =
301 {
302     \_siunitx_option_deprecated:nnV
303     { retain-zero-exponent }
304     { print-zero-exponent }
305     \l_keys_value_tl
306 },
307 retain-zero-exponent .default:n = true ,
308 round-integer-to-decimal .code:n =
309 { \_siunitx_option_removed:V \l_keys_key_tl } ,
310 scientific-notation .choice: ,
311 scientific-notation / engineering .code:n =
312 {
313     \_siunitx_option_deprecated:nn
314     { scientific-notation==~engineering }
315     { exponent-mode==~engineering }
316 },
317 scientific-notation / fixed .code:n =
318 {
319     \_siunitx_option_deprecated:nn
320     { scientific-notation==~fixed }
321     { exponent-mode==~fixed }
322 },
323 scientific-notation / false .code:n =
324 {
325     \_siunitx_option_deprecated:nn
326     { scientific-notation==~false }
327     { exponent-mode==~input }
328 },
329 scientific-notation / true .code:n =
330 {
331     \_siunitx_option_deprecated:nn

```

```

332         { scientific-notation~~true }
333         { exponent-mode~~scientific }
334     } ,
335     scientific-notation .default:n = true ,
336     zero-decimal-to-integer .code:n =
337     {
338         \__siunitx_option_deprecated:nnV
339         { zero-decimal-to-integer }
340         { drop-zero-decimal }
341         \l_keys_value_tl
342     } ,
343     zero-decimal-to-integer .default:n = true
344 }

```

1.7.1 Table options

All straight-forward emulation.

```

345 \keys_define:nn { siunitx }
346 {
347     table-align-text-post .code:n =
348     {
349         \__siunitx_option_deprecated:nnV
350         { table-align-text-post }
351         { table-align-text-after }
352         \l_keys_value_tl
353     } ,
354     table-align-text-post .default:n = true ,
355     table-align-text-pre .code:n =
356     {
357         \__siunitx_option_deprecated:nnV
358         { table-align-text-pre }
359         { table-align-text-before }
360         \l_keys_value_tl
361     } ,
362     table-align-text-pre .default:n = true ,
363     table-number-alignment / center-decimal-marker .code:n =
364     {
365         \msg_info:nnnn { siunitx } { option-deprecated }
366         { table-number-alignment~~center-decimal-marker }
367         { table-alignment-mode~~marker }
368         \keys_set:nn
369         { siunitx }
370         { table-alignment-mode = marker }
371     } ,
372     table-omit-exponent .code:n =
373     {
374         \__siunitx_option_deprecated:nnV
375         { table-omit-exponent }
376         { drop-exponent }
377         \l_keys_value_tl
378     } ,
379     table-omit-exponent .default:n = true ,
380     table-parse-only .code:n =
381     {
382         \msg_info:nnnn { siunitx } { option-deprecated }

```

```

383         { table-parse-only }
384         { table-alignment-mode~~none }
385     \str_if_eq:VnTF \l_keys_value_tl { false }
386     {
387         \keys_set:nn
388         { siunitx }
389         { table-alignment-mode = marker }
390     }
391     {
392         \keys_set:nn
393         { siunitx }
394         { table-alignment-mode = none }
395     }
396 },
397 table-space-text-post .code:n =
398 {
399     \msg_info:nnnn { siunitx } { option-deprecated }
400     { table-space-text-post }
401     { table-format }
402     \tl_set:Nn \l__siunitx_table_after_model_tl {#1}
403 },
404 table-space-text-pre .code:n =
405 {
406     \msg_info:nnnn { siunitx } { option-deprecated }
407     { table-space-text-post }
408     { table-format }
409     \tl_set:Nn \l__siunitx_table_before_model_tl {#1}
410 }
411 }

\__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

412 \cs_new_protected:Npn \__siunitx_option_table_format:n #1
413 {
414     \msg_info:nnnn { siunitx } { option-deprecated }
415     { table- #1 }
416     { table-format }
417     \tl_set:Nx \l__siunitx_table_format_tl
418     {
419         \cs:w __siunitx_option_table_ #1 :nnnnnnnn
420         \exp_after:wN \exp_after:wN \exp_after:wN \cs_end:
421         \exp_after:wN \l__siunitx_table_format_tl
422         \exp_after:wN { \l_keys_value_tl }
423     }
424     \exp_after:wN \__siunitx_table_generate_model:nnnnnnn
425     \l__siunitx_table_format_tl
426 }
427 \cs_new:Npn \__siunitx_option_table_comparator:nnnnnnnn #1#2#3#4#5#6#7#8
428 { \exp_not:n { {#8} {#2} {#3} {#4} {#5} {#6} {#7} } }
429 \cs_new:cpn { __siunitx_option_table_figures-decimal:nnnnnnnn }
430 #1#2#3#4#5#6#7#8
431 { \exp_not:n { {#1} {#2} {#3} {#8} {#5} {#6} {#7} } }
432 \cs_new:cpn { __siunitx_option_table_figures-exponent:nnnnnnnn }
433 #1#2#3#4#5#6#7#8
434 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#8} } }

```

```

435 \cs_new:cpn { __siunitx_option_table_figures-integer:nnnnnnnn }
436   #1#2#3#4#5#6#7#8
437   { \exp_not:n { {#1} {#2} {#8} {#4} {#5} {#6} {#7} } }
438 \cs_new:cpn { __siunitx_option_table_figures-uncertainty:nnnnnnnn }
439   #1#2#3#4#5#6#7#8
440   { \exp_not:n { {#1} {#2} {#3} {#4} { { S } {#8} } {#6} {#7} } }
441 \cs_new:cpn { __siunitx_option_table_sign-exponent:nnnnnnnn }
442   #1#2#3#4#5#6#7#8
443   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#8} {#7} } }
444 \cs_new:cpn { __siunitx_option_table_sign-mantissa:nnnnnnnn }
445   #1#2#3#4#5#6#7#8
446   { \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.

```

447 \keys_define:nn { siunitx }
448 {
449   table-comparator .code:n =
450     { \_siunitx_option_table_format:n { comparator } } ,
451   table-figures-decimal .code:n =
452     { \_siunitx_option_table_format:n { figures-decimal } } ,
453   table-figures-exponent .code:n =
454     { \_siunitx_option_table_format:n { figures-exponent } } ,
455   table-figures-integer .code:n =
456     { \_siunitx_option_table_format:n { figures-integer } } ,
457   table-figures-uncertainty .code:n =
458     { \_siunitx_option_table_format:n { figures-uncertainty } } ,
459   table-sign-exponent .code:n =
460     { \_siunitx_option_table_format:n { sign-exponent } } ,
461   table-sign-mantissa .code:n =
462     { \_siunitx_option_table_format:n { sign-mantissa } }
463 }

```

1.8 Unit options

```

464 \keys_define:nn { siunitx }
465 {
466   fraction-function .code:n =
467     {
468       \_siunitx_option_deprecated:nnV
469       { fraction-function }
470       { fraction-command }
471       \l_keys_value_tl
472     } ,
473   literal-superscript-as-power .code:n =
474     { \_siunitx_option_removed:V \l_keys_key_tl } ,
475   per-mode / reciprocal .code:n =
476     {
477       \_siunitx_option_deprecated:nn
478       { per-mode-==reciprocal }
479       { per-mode-==power }
480     } ,
481   per-mode / reciprocal-positive-first .code:n =
482     {

```

```

483     \_siunitx_option_deprecated:nn
484     { per-mode~==reciprocal-positive-first }
485     { per-mode~==power-positive-first }
486   } ,
487   power-font .code:n =
488   { \_siunitx_option_removed:V \l_keys_key_tl } ,
489   qualifier-mode / brackets .code:n =
490   {
491     \_siunitx_option_deprecated:nn
492     { qualifier-mode~==brackets }
493     { qualifier-mode~==bracket }
494   } ,
495   qualifier-mode / space .code:n =
496   {
497     \msg_info:nnnn { siunitx } { option-deprecated }
498     { qualifier-mode~==space }
499     { qualifier-mode~==phrase"~plus~"qualifier-phrase=\ }
500     \keys_set:nn
501     { siunitx }
502     { qualifier-mode = phrase, qualifier-phrase = \ }
503   } ,
504   qualifier-mode / text .code:n =
505   {
506     \_siunitx_option_deprecated:nn
507     { qualifier-mode~==text }
508     { qualifier-mode~==combine }
509   }
510 }

```

1.9 Quantity units

```

511 \keys_define:nn { siunitx }
512 {
513   allow-number-unit-breaks .code:n =
514   {
515     \_siunitx_option_deprecated:nnV
516     { allow-number-unit-breaks }
517     { allow-quantity-breaks }
518     \l_keys_value_tl
519   } ,
520   allow-number-unit-breaks .default:n = true ,
521   exponent-to-prefix .choice: ,
522   exponent-to-prefix / false .code:n =
523   {
524     \_siunitx_option_deprecated:nn
525     { exponent-to-prefix~==false }
526     { prefix-mode~==input }
527   } ,
528   exponent-to-prefix / true .code:n =
529   {
530     \_siunitx_option_deprecated:nn
531     { exponent-to-prefix~==true }
532     { prefix-mode~==combine-exponent }
533   } ,
534   exponent-to-prefix .default:n = true ,

```



```

535 multi-part-units .choice: ,
536 multi-part-units / brackets . code:n =
537 {
538   \_siunitx_option_deprecated:nn
539   { multi-part-units~--brackets }
540   { separate-uncertainty-units~--bracket }
541 } ,
542 multi-part-units / repeat . code:n =
543 {
544   \_siunitx_option_deprecated:nn
545   { multi-part-units~--repeat }
546   { separate-uncertainty-units~--repeat }
547 } ,
548 multi-part-units / single . code:n =
549 {
550   \_siunitx_option_deprecated:nn
551   { multi-part-units~--single }
552   { separate-uncertainty-units~--single }
553 } ,
554 number-unit-product .code:n =
555 {
556   \_siunitx_option_deprecated:nnV
557   { number-unit-product }
558   { quantity-product }
559   \l_keys_value_tl
560 } ,
561 prefixes-as-symbols .choice: ,
562 prefixes-as-symbols / false . code:n =
563 {
564   \_siunitx_option_deprecated:nn
565   { prefixes-as-symbols~--false }
566   { prefix-mode~--extract-exponent }
567 } ,
568 prefixes-as-symbols / true . code:n =
569 {
570   \_siunitx_option_deprecated:nn
571   { prefixes-as-symbols~--true }
572   { prefix-mode~--input }
573 } ,
574 prefixes-as-symbols .default:n = true
575 }
576 </options>

```

1.10 Preamble commands

```

577 <*interfaces>

```

`\DeclareBinaryPrefix` We simply drop #3.

```

578 \NewDocumentCommand \DeclareBinaryPrefix { +m m m }
579 {
580   \siunitx_declare_prefix:Nn #1 {#2}
581 }

```

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

`\DeclareSIPrePower` Simply use a throw-away command for the part we do not need: this can be followed by
`\DeclareSIPostPower` some clean-up.

```

582 \NewDocumentCommand \DeclareSIPrePower { +m m }
583 {
584   \siunitx_declare_power:NNn #1 \_siunitx_tmp:w {#2}
585   \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
586 }
587 \NewDocumentCommand \DeclareSIPostPower { +m m }
588 {
589   \siunitx_declare_power:NNn \_siunitx_tmp:w #1 {#2}
590   \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
591 }

```

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

1.11 Document commands

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

```

592 \NewDocumentCommand \si { 0 { } m }
593 {
594   \mode_leave_vertical:
595   \group_begin:
596     \keys_set:nn { siunitx } {#1}
597     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
598     \siunitx_print_unit:V \l__siunitx_tmp_tl
599   \group_end:
600 }

```

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

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

```

601 \NewDocumentCommand \SI { 0 { } m o m }
602 {
603   \mode_leave_vertical:
604   \group_begin:
605     \keys_set:nn { siunitx } {#1}
606     \IfNoValueF {#3}
607     {
608       \siunitx_unit_format:nN {#3} \l__siunitx_tmp_tl
609       \siunitx_print_unit:V \l__siunitx_tmp_tl
610       \nobreak
611     }
612     \siunitx_quantity:nn {#2} {#4}
613   \group_end:
614 }

```

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

`\SIlist` Straight copies.
`\SIrange`

```

615 \NewDocumentCommand \SIlist
616 { 0 { } > { \SplitList { ; } } m > { \TrimSpaces } m }
617 {
618   \mode_leave_vertical:

```

```

619 \group_begin:
620 \siunitx_unit_options_apply:n {#3}
621 \keys_set:nn { siunitx } {#1}
622 \siunitx_quantity_list:nn {#2} {#3}
623 \group_end:
624 }
625 \NewDocumentCommand \SIrange { 0 { } m m > { \TrimSpaces } m }
626 {
627 \mode_leave_vertical:
628 \group_begin:
629 \siunitx_unit_options_apply:n {#4}
630 \keys_set:nn { siunitx } {#1}
631 \siunitx_quantity_range:nnn {#2} {#3} {#4}
632 \group_end:
633 }

```

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

1.12 Symbol commands

```

634 <@@=siunitx_emulation>

```

```

\__siunitx_emulation_non_latin:n
\__siunitx_emulation_non_latin:nnnn

```

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

```

635 \bool_lazy_or:nnTF
636 { \sys_if_engine luatex_p: }
637 { \sys_if_engine xetex_p: }
638 {
639 \cs_new:Npn \__siunitx_emulation_non_latin:n #1
640 { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
641 }
642 {
643 \cs_new:Npn \__siunitx_emulation_non_latin:n #1
644 {
645 \exp_last_unbraced:Nf \__siunitx_emulation_non_latin:nnnn
646 { \char_to_utfviii_bytes:n {#1} }
647 }
648 \cs_new:Npn \__siunitx_emulation_non_latin:nnnn #1#2#3#4
649 {
650 \exp_after:wN \exp_after:wN \exp_after:wN
651 \exp_not:N \char_generate:nn {#1} { 13 }
652 \exp_after:wN \exp_after:wN \exp_after:wN
653 \exp_not:N \char_generate:nn {#2} { 13 }
654 }
655 }

```

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

```

\SIUnitSymbolAngstrom
\SIUnitSymbolArcminute
\SIUnitSymbolArcsecond
\SIUnitSymbolCelsius
\SIUnitSymbolDegree
\SIUnitSymbolMicro
\SIUnitSymbolOhm

```

The same setup as elsewhere but localised to the emulation module

```

656 \AtBeginDocument
657 {
658 \cs_new_protected:Npn \SIUnitSymbolArcminute
659 { \ensuremath { { } ^{\circ} } }
660 \cs_new_protected:Npn \SIUnitSymbolArcsecond
661 { \ensuremath { { } ^{\prime} } }
662 \@ifpackageloaded { fontspec }

```

```

663 {
664   \cs_new_protected:Npx \SIUnitSymbolAngstrom
665     { \__siunitx_emulation_non_latin:n { "00C5 } }
666   \cs_new_protected:Npx \SIUnitSymbolDegree
667     { \__siunitx_emulation_non_latin:n { "00B0 } }
668   \cs_new_protected:Npx \SIUnitSymbolCelsius
669     { \__siunitx_emulation_non_latin:n { "00B0 } C }
670 }
671 {
672   \cs_new_protected:Npx \SIUnitSymbolAngstrom
673     {
674       \siunitx_print_text:n
675       { \__siunitx_emulation_non_latin:n { "00C5 } }
676     }
677   \cs_new_protected:Npx \SIUnitSymbolCelsius
678     {
679       \siunitx_print_text:n
680       { \__siunitx_emulation_non_latin:n { "00B0 } } C
681     }
682   \cs_new_protected:Npx \SIUnitSymbolDegree
683     {
684       \siunitx_print_text:n
685       { \__siunitx_emulation_non_latin:n { "00B0 } }
686     }
687 }
688 \cs_new_protected:Npx \SIUnitSymbolMicro
689 {
690   \siunitx_print_text:n
691   {
692     \bool_lazy_or:nnTF
693     { \sys_if_engine luatex_p: }
694     { \sys_if_engine xetex_p: }
695     { \__siunitx_emulation_non_latin:n { "00B5 } }
696     { \exp_not:N \textmu }
697   }
698 }
699 \cs_new_protected:Npx \SIUnitSymbolOhm
700 {
701   \exp_not:N \ifmmode
702     \cs_if_exist:NTF \upOmega
703     { \exp_not:N \upOmega }
704     { \exp_not:N \Omega }
705   \exp_not:N \else
706     \siunitx_print_text:n
707     {
708       \bool_lazy_or:nnTF
709       { \sys_if_engine luatex_p: }
710       { \sys_if_engine xetex_p: }
711       { \__siunitx_emulation_non_latin:n { "03A9 } }
712       { \exp_not:N \textohm }
713     }
714   \exp_not:N \fi
715 }
716 }

```

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

717 `</interfaces>`

718 `</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	<i>102</i> , <i>135</i> , 9, 15, 21, 27, 156, 1127, 1947, 1953
$\backslash-$	101
$\backslash\backslash$	6, 84, 1089
$\backslash_$	8, 216, 227, 296, 373
$\backslash\sim$	212
\backslashsqcup	48, 50, 54, 56, 60, 62, 80, 114, 499, 499, 501, 502, 505, 507, 510, 516, 518, 522, 524, 527, 533, 535, 542, 544, 1096
A	
$\backslash A$	<i>175</i> , <u>2</u>
allow-quantity-breaks	<i>102</i>
\backslash ampere	<i>139</i> , 2, 3, 4, 5, 6, 7, <u>1009</u>
\backslash ang	6, <u>100</u> , 277, 313
angle-mode	<i>12</i>
angle-symbol-degree	<i>12</i>
angle-symbol-minute	<i>12</i>
angle-symbol-over-decimal	<i>12</i>
angle-symbol-second	<i>12</i>
\backslash approx	1949
arc-separator	<i>12</i>
\backslash arcminute ...	<i>140</i> , 51, 53, 286, 353, <u>1074</u>
\backslash arcsecond ...	<i>140</i> , 56, 58, 288, 358, <u>1074</u>
\backslash as	<i>175</i> , <u>89</u>
\backslash astronomicalunit	<i>140</i> , <u>1061</u>
\backslash AtBeginDocument	5, 17, 37, 49, 59, 65, 129, 150, 213, 240, 244, 312, 318, 656
\backslash atto	<i>139</i> , 90, <u>1018</u>
B	
\backslash becquerel	<i>140</i> , <u>1039</u>
\backslash begin	203
\backslash bel	<i>140</i> , 69, <u>1061</u>
\backslash bfseries	<i>89</i>
\backslash binoppenalty	146
\backslash bit	<i>181</i> , <u>10</u>
\backslash boldmath	<i>90</i>
bool commands:	
\backslash bool_if:NTF	10, 16, 17, 18, 32, 34, 39, 47, 50, 61, 62, 64, 71, 75, 80, 88, 91, 94, 106, 110, 119, 121, 126, 144, 145, 151, 158, 162, 174, 183, 186, 189, 192, 204, 233, 233, 235, 244, 256, 258, 260, 262,
354, 380, 381, 407, 415, 422, 431, 437, 439, 452, 475, 476, 544, 570, 582, 589, 634, 676, 699, 725, 750, 759, 837, 857, 869, 886, 906, 920, 929, 932, 995, 1005, 1282, 1446, 1610, 1618, 1634, 1682, 1847, 1858	
\backslash bool_lazy_all:nTF	1580, 1657
\backslash bool_lazy_all_p:n	1093, 1111
\backslash bool_lazy_and:nnTF	109, 137, 217, 247, 262, 431, 515, 576, 618, 675, 699, 871, 879, 1375, 1842
\backslash bool_lazy_any:nTF	264
\backslash bool_lazy_or:nnTF	10, 105, 120, 159, 203, 237, 317, 371, 443, 543, 635, 692, 708, 988, 1090, 1108, 1406, 1822, 1838
\backslash bool_lazy_or_p:nn	1586
\backslash bool_new:N	3, 6, 9, 10, 10, 11, 17, 18, 19, 20, 21, 22, 43, 44, 76, 88, 332, 405, 545, 550, 551, 552, 553, 554, 555, 558, 662, 1486, 1546, 1547
\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, 209, 216, 236, 240, 323, 338, 339, 416, 497, 498, 504, 505, 506, 507, 511, 512, 513, 518, 521, 525, 583, 585, 651, 659, 698, 746, 781, 853, 1507, 1511, 1516, 1517
\backslash bool_set_true:N ...	11, 14, 17, 24, 29, 30, 33, 34, 51, 55, 61, 62, 68, 75, 102, 141, 163, 189, 208, 214, 234, 238, 301, 337, 414, 433, 499, 500, 514, 519, 520, 526, 527, 528, 532, 533, 534, 535, 586, 649, 843, 874, 913, 1501, 1502, 1506, 1512, 1873, 1874
\backslash c_false_bool	47, 121, 387, 449, 453, 458, 466, 490, 496, 534, 549, 591, 614
\backslash c_true_bool	44, 118, 372, 387, 447, 463, 490, 496, 547, 618
box commands:	
\backslash box_clear:N	474, 533
\backslash box_new:N	3, 6, 170, 171, 242, 243, 450, 451
\backslash box_use:N	249
\backslash box_use_drop:N	244, 246, 399, 400, 444, 445,

497, 504, 517, 518, 577, 578, 579, 580
`\box_wd:N` 228, 229, 251, 252,
 255, 263, 263, 376, 415, 419, 433,
 472, 479, 480, 483, 491, 531, 537,
 568, 592, 603, 604, 605, 620, 624,
 625, 638, 639, 649, 657, 658, 660,
 677, 678, 699, 710, 729, 731, 741, 753
`bracket-ambiguous-numbers` 39
`bracket-negative-numbers` 39
`bracket-unit-denominator` 141
`\byte` 181, 10

C

`\cancel` 135, 141, 147, 107
`\candela` 139, 1009
`\cdot` 8, 33
`\centi` 139, 65, 1018
char commands:
`\char_generate:nn` 8,
 15, 26, 28, 164, 175, 177, 216, 227,
 296, 373, 640, 651, 653, 993, 1004, 1006
`\char_set_active_eq:NN`
 222, 224, 226, 382, 423
`\char_set_catcode_active:N` 101
`\char_set_catcode_active:n` 212
`\char_set_mathcode:nn` 383, 424
`\char_to_utfviii_bytes:n`
 21, 170, 646, 999
`\char_value_catcode:n` 15, 164, 640, 993
clist commands:
`\clist_map_break:` 119
`\clist_map_function:nN` 34, 39
`\clist_map_inline:nn`
 19, 28, 107, 120,
 122, 123, 159, 168, 192, 229, 476, 588
`\cm` 175, 60
`color` 89
`\complexnum` 181
`\complexqty` 181
`compound-exponents` 21
`compound-final-separator` 21
`compound-pair-separator` 21
`compound-separator` 21
`compound-separator-mode` 21
`compound-units` 21
`\coulomb` 140, 1039
`\cr` 116, 29
cs commands:
`\cs:w` 117, 419, 714, 1019
`\cs_end:` 117, 420, 717, 1022
`\cs_generate_variant:Nn` . . . 2, 3, 3,
 4, 5, 5, 6, 21, 27, 61, 72, 87, 87, 90,
 114, 148, 151, 169, 181, 183, 193,
 199, 207, 224, 241, 281, 358, 368,

456, 468, 788, 843, 871, 903, 1032,
 1194, 1197, 1208, 1679, 1797, 1812
`\cs_if_eq:NNTF` 362
`\cs_if_exist:NTF`
 99, 138, 141, 223, 250, 333, 702
`\cs_if_free:NNTF` 7, 55
`\cs_new:Npn` 14, 18, 23, 67,
 131, 133, 136, 146, 163, 167, 168,
 172, 209, 210, 229, 240, 245, 246,
 247, 334, 343, 344, 346, 361, 363,
 369, 384, 386, 389, 427, 429, 432,
 435, 438, 441, 444, 520, 583, 639,
 643, 648, 671, 674, 692, 711, 725,
 731, 738, 740, 746, 766, 778, 789,
 795, 801, 807, 819, 826, 844, 866,
 872, 873, 880, 900, 905, 919, 929,
 942, 950, 963, 977, 983, 988, 992,
 996, 1001, 1001, 1013, 1025, 1027,
 1033, 1042, 1058, 1072, 1083, 1104,
 1122, 1152, 1179, 1186, 1193, 1195,
 1198, 1209, 1219, 1226, 1233, 1234,
 1235, 1242, 1244, 1250, 1255, 1269,
 1277, 1286, 1301, 1309, 1327, 1343,
 1344, 1362, 1371, 1373, 1395, 1404,
 1421, 1430, 1452, 1462, 1464, 1469,
 1474, 1476, 1549, 1551, 1553, 1555,
 1557, 1562, 1567, 1578, 1593, 1600,
 1606, 1626, 1632, 1638, 1647, 1655,
 1666, 1680, 1691, 1700, 1702, 1704,
 1706, 1712, 1722, 1727, 1747, 1756,
 1758, 1777, 1798, 1813, 1818, 1820,
 1829, 1835, 1867, 1902, 1908, 1913
`\cs_new:Npx` 245, 391, 400
`\cs_new_eq:NN` 217, 244, 764
`\cs_new_protected:Npn` 7,
 11, 14, 14, 16, 19, 22, 23, 24, 25, 27,
 28, 37, 37, 39, 40, 40, 45, 45, 49, 50,
 55, 59, 60, 61, 64, 68, 70, 70, 73, 77,
 78, 84, 85, 88, 89, 89, 89, 91, 91, 91,
 93, 94, 96, 98, 101, 103, 105, 105,
 108, 111, 115, 120, 123, 124, 127,
 131, 132, 133, 133, 133, 135, 139,
 139, 140, 141, 144, 146, 149, 150,
 151, 153, 155, 156, 156, 157, 160,
 160, 166, 168, 170, 172, 172, 172,
 173, 176, 178, 182, 184, 186, 187,
 187, 194, 196, 198, 200, 201, 202,
 204, 208, 208, 209, 210, 216, 221,
 221, 226, 228, 230, 230, 232, 236,
 243, 248, 248, 248, 254, 254, 259,
 260, 260, 261, 269, 270, 272, 275,
 276, 277, 282, 282, 293, 294, 298,
 304, 307, 312, 315, 317, 318, 321,
 328, 329, 333, 334, 337, 338, 347,

347, 350, 350, 351, 352, 352, 357,
 359, 360, 361, 367, 368, 369, 371,
 374, 376, 378, 381, 386, 386, 388,
 390, 394, 398, 402, 404, 409, 410,
 412, 423, 428, 429, 429, 430, 437,
 437, 441, 443, 443, 448, 449, 451,
 456, 461, 466, 468, 469, 470, 474,
 477, 480, 484, 499, 505, 506, 510,
 526, 527, 537, 553, 565, 566, 575,
 585, 594, 594, 596, 602, 603, 614,
 625, 634, 650, 658, 660, 666, 670,
 682, 684, 685, 690, 694, 694, 696,
 697, 704, 710, 713, 715, 718, 723,
 748, 754, 757, 776, 786, 788, 797,
 808, 809, 819, 831, 845, 850, 862,
 867, 893, 904, 916, 925, 937, 953,
 958, 962, 971, 993, 1003, 1015, 1444
 \cs_new_protected:Npx 102, 171, 179,
 195, 200, 210, 213, 225, 285, 366,
 664, 666, 668, 672, 677, 682, 688, 699
 \cs_set:Npn 30, 34
 \cs_set_eq:NN . 156, 180, 311, 325, 326
 \cs_set_protected:Npn
 29, 56, 69, 85, 220,
 229, 244, 248, 255, 277, 316, 340, 434
 \cs_set_protected:Npx 69, 198
 \cs_to_str:N 273, 327, 1077
 \cs_undefine:N ... 47, 48, 67, 225, 252
 \cubed 141, 1078
 \cubic 140, 1078

D

\dalton 140, 1061
 \day 140, 1061
 \dB 177, 69
 \deca 139, 1028
 \deci 139, 66, 69, 1018, 1063
 \decibel 140, 1061
 \DeclareBinaryPrefix 578
 \DeclareCurrentRelease 5, 8
 \DeclareExpandableDocumentCommand . 272
 \DeclareRelease 4, 6, 7
 \DeclareSIPostPower 582
 \DeclareSIPower 62
 \DeclareSIPrefix 62
 \DeclareSIPrePower 582
 \DeclareSIQualifier 62
 \DeclareSIUnit 62
 \DeclareTranslation 40, 41, 42, 43
 \def 333
 \degree 107, 140, 61, 66, 180, 285, 349, 1074
 \degreeCelsius 140, 79, 86, 1039
 \deka 139, 1028

dim commands:

\dim_add:Nn 750
 \dim_compare:nNnTF
 228, 250, 472, 478, 531
 \dim_compare_p:nNn 620
 \dim_new:N 4, 4, 452
 \dim_set:Nn 263, 648, 675, 699, 710, 738
 \dm 175, 60
 \document 8
 drop-exponent 39
 drop-uncertainty 39
 drop-zero-decimal 39

E

\edef 334
 \electronvolt
 140, 52, 53, 54, 55, 56, 57, 1061
 \else 48, 50, 54, 56, 60, 62, 102, 143, 145,
 499, 501, 505, 507, 510, 516, 518,
 522, 524, 527, 533, 535, 542, 544, 705
 else commands:
 \else: 344
 \end 55, 76, 200
 \endinput 21
 \ensuremath 38, 89, 24,
 24, 53, 54, 59, 158, 171, 249, 257,
 293, 370, 374, 388, 413, 417, 659, 661
 \ERROR 56
 \eV 176, 42
 evaluate-expression 39
 \exa 139, 1028
 \exbi 181, 2
 exp commands:

\exp_after:wN . 25, 27, 68, 80, 115,
 124, 129, 132, 138, 140, 144, 148,
 153, 168, 170, 174, 176, 180, 205,
 231, 235, 245, 308, 343, 345, 358,
 371, 372, 406, 420, 421, 422, 424,
 482, 502, 507, 551, 564, 595, 650,
 652, 669, 687, 691, 703, 715, 716,
 722, 961, 998, 1003, 1005, 1008,
 1021, 1264, 1389, 1449, 1560, 1910
 \exp_args:Nc 129, 272
 \exp_args:Ne 48, 63, 1331, 1472
 \exp_args:Nf
 402, 727, 742, 1067, 1078, 1324
 \exp_args:Nff 338
 \exp_args:NNc 232
 \exp_args:Nnno 1414
 \exp_args:NNNV
 . 25, 130, 239, 304, 556, 661, 783, 941
 \exp_args:NNnx 247
 \exp_args:NNV 16, 107, 357
 \exp_args:NnV 272

<code>\exp_args:Nv</code>	1837, 1845, 1850, 1851, 1853, 1865
... 73, 109, 191, 249, 285, 1222, 1467	
<code>\exp_args:Nv</code>	234
<code>\exp_args:Nx</code>	<code>\ExplFileDate</code> 1, 12
154, 278, 546	exponent-base 39
<code>\exp_args:Nxx</code>	exponent-mode 39
96	exponent-product 39
<code>\exp_last_unbraced:Nf</code>	expression 39
20, 169, 645, 998, 1160, 1423	extract-mass-in-kilograms 142
<code>\exp_not:N</code>	
26, 28, 36, 54, 59, 76, 77, 80,	
82, 84, 88, 98, 100, 101, 102, 105,	
109, 111, 124, 156, 157, 173, 175,	
175, 177, 181, 183, 197, 198, 200,	
202, 202, 204, 213, 214, 215, 217,	
217, 218, 218, 220, 220, 221, 223,	
225, 226, 227, 228, 269, 270, 272,	
276, 278, 280, 281, 282, 285, 288,	
288, 288, 289, 289, 290, 290, 291,	
292, 292, 293, 294, 295, 295, 297,	
297, 298, 299, 300, 300, 302, 302,	
303, 304, 306, 307, 308, 312, 313,	
314, 336, 337, 338, 369, 370, 372,	
374, 393, 396, 403, 404, 561, 651,	
653, 696, 701, 703, 704, 705, 712,	
714, 1004, 1006, 1641, 1672, 1715, 1848	
<code>\exp_not:n</code>	
110,	
111, 112, 113, 114, 114, 125, 134,	
161, 169, 201, 215, 216, 221, 228,	
242, 267, 269, 270, 274, 279, 280,	
287, 288, 289, 292, 293, 296, 297,	
302, 303, 304, 307, 312, 314, 316,	
326, 326, 327, 333, 340, 341, 346,	
354, 356, 357, 365, 366, 394, 396,	
397, 398, 402, 428, 431, 434, 436,	
437, 440, 440, 441, 443, 446, 448,	
449, 450, 474, 489, 523, 525, 589,	
610, 678, 679, 679, 680, 682, 682,	
703, 712, 725, 733, 733, 734, 736,	
739, 748, 756, 774, 798, 802, 803,	
804, 805, 814, 815, 816, 826, 827,	
829, 834, 835, 839, 840, 872, 882,	
897, 899, 900, 902, 909, 910, 911,	
913, 914, 915, 923, 933, 937, 938,	
939, 947, 948, 949, 950, 954, 957,	
958, 959, 965, 966, 967, 968, 971,	
975, 982, 983, 984, 990, 1002, 1014,	
1026, 1292, 1295, 1299, 1333, 1341,	
1343, 1379, 1384, 1385, 1386, 1393,	
1402, 1409, 1413, 1416, 1418, 1454,	
1456, 1459, 1460, 1471, 1479, 1480,	
1595, 1597, 1603, 1604, 1624, 1630,	
1635, 1636, 1641, 1650, 1652, 1668,	
1674, 1676, 1686, 1689, 1708, 1717,	
1718, 1731, 1732, 1736, 1740, 1757,	
1762, 1764, 1770, 1771, 1773, 1819,	
	F
	<code>\F</code> 177, 70
	<code>\fam</code> 92, 27, 191, 204, 205
	<code>\familydefault</code> 89, 259
	<code>\farad</code> 140, 70, 71, 72, 73, 74, 1039
	<code>\femto</code> 139, 15, 35, 71, 91, 1018
	<code>\fF</code> 177, 70
	<code>\fg</code> 175, 35, 333
	<code>\fi</code> 48, 50, 54, 56, 60, 62, 111, 143, 145,
	499, 501, 505, 507, 510, 516, 518,
	522, 524, 527, 533, 535, 542, 544, 714
	fi commands:
	<code>\fi:</code> 28, 34, 346
	file commands:
	<code>\file_if_exist:nTF</code> 37
	fill-angle-degrees 12
	fill-angle-minutes 12
	fill-angle-seconds 13
	fixed-exponent 40
	<code>\fmol</code> 176, 14
	<code>\fmtversion</code> 42
	<code>\fontfamily</code> 89, 259
	<code>\fontseries</code> 89, 261
	<code>\fontshape</code> 89, 263
	<code>\fontsize</code> 382
	forbid-literal-units 142
	fp commands:
	<code>\fp_add:Nn</code> 771
	<code>\fp_compare:nNnTF</code>
	458, 572, 574, 622, 642
	<code>\fp_eval:n</code> 48,
	64, 73, 74, 76, 122, 457, 621, 1472
	<code>\fp_new:N</code> 4, 4, 557, 559, 560
	<code>\fp_set:Nn</code>
	135, 136, 143, 149, 150, 157,
	165, 172, 173, 202, 229, 606, 640, 668
	<code>\fp_use:N</code> 203, 646
	<code>\fp_zero:N</code> . 135, 142, 156, 164, 182, 570
	<code>\c_one_fp</code> 136, 143, 150, 574, 642
	<code>\l_tmpa_fp</code> 136
	<code>\c_zero_fp</code> 572, 622
	<code>\frac</code> 135, 1126
	fraction-command 142
	free-standing-units 182
	<code>\fs</code> 175, 89

G	
<code>\g</code>	175, 35
<code>\ge</code>	98, 1949
<code>\geq</code>	1949
<code>\GetTranslation</code>	49, 55, 61
<code>\GeV</code>	176, 42
<code>\gg</code>	97, 1949
<code>\GHz</code>	175, 8
<code>\gibi</code>	181, 2
<code>\giga</code>	139, 12, 47, 56, 85, 1028
<code>\GPa</code>	177, 82
<code>\gram</code>	139, 153, 35, 36, 37, 38, 39, 40, 41, 413, 1009, 1017
<code>\gray</code>	140, 1039
group commands:	
<code>\group_begin:</code>	138, 13, 16, 21, 26, 33, 42, 74, 74, 86, 93, 94, 100, 103, 111, 117, 120, 131, 131, 140, 149, 159, 167, 176, 178, 184, 192, 201, 211, 212, 215, 232, 234, 279, 300, 318, 354, 368, 376, 383, 425, 432, 472, 479, 543, 595, 604, 619, 628, 658, 780, 939, 1872
<code>\c_group_begin_token</code> ...	41, 146, 362
<code>\group_end:</code> ...	138, 16, 25, 30, 41, 45, 59, 82, 87, 89, 93, 98, 106, 107, 107, 115, 125, 130, 135, 138, 139, 143, 148, 153, 162, 171, 179, 187, 196, 205, 206, 222, 238, 239, 244, 258, 283, 304, 322, 357, 376, 379, 386, 428, 435, 475, 482, 556, 599, 613, 623, 632, 661, 783, 941, 1878, 1882
group-digits	40
group-minimum-digits	40
group-separator	40
<code>\GW</code>	176, 42
H	
<code>\H</code>	75, 334, 337, 338
hbox commands:	
<code>\hbox_overlap_right:n</code>	243, 245
<code>\hbox_set:Nn</code>	22, 214, 219, 373, 412, 417, 471, 477, 509, 511, 529, 530, 566, 650, 717, 745
<code>\hbox_set:Nw</code>	378, 390
<code>\hbox_set_end:</code> ...	389, 397, 431, 441
<code>\hbox_set_to_wd:Nnn</code>	240, 254, 262, 264, 375, 414, 482, 490, 536, 567, 591, 601, 622, 636, 655, 727
<code>\hbox_set_to_wd:Nnw</code>	418, 432
<code>\hbox_to_wd:nn</code>	205
<code>\hbox_unpack:N</code>	257, 266, 486, 493, 540, 609, 628, 643, 652, 664, 734, 736, 747, 748
<code>\hbox_unpack_drop:N</code>	268
<code>\hectare</code>	140, 1061
<code>\hecto</code>	139, 27, 31, 1028
<code>\henry</code>	140, 75, 76, 77, 1039
<code>\hertz</code>	140, 8, 9, 10, 11, 12, 13, 1039
<code>\highlight</code>	141, 147, 107
<code>\hL</code>	176, 27
<code>\hl</code>	176, 184, 27, 47
hook commands:	
<code>\hook_gput_code:nnn</code>	9
<code>\hour</code>	140, 58, 1061
<code>\Hz</code>	175, 8
I	
if commands:	
<code>\if_false:</code>	28, 34
<code>\if_meaning:w</code>	342
<code>\IfFormatAtLeastTF</code>	8, 16, 42, 59
<code>\ifmode</code>	48, 50, 54, 56, 60, 62, 98, 143, 145, 499, 501, 505, 507, 510, 516, 518, 522, 524, 527, 533, 535, 542, 544, 701
<code>\IfNoValueF</code>	606
<code>\IfNoValueTF</code>	76, 76
<code>\ignorespaces</code>	113, 18, 37, 204, 359
input-close-uncertainty	40
input-comparators	40
input-decimal-markers	40
input-digits	40
input-exponent-markers	40
input-open-uncertainty	40
input-signs	40
input-uncertainty-signs	40
int commands:	
<code>\int_abs:n</code> ...	190, 835, 838, 878, 1791
<code>\int_case:nnTF</code> ...	191, 284, 320, 371, 888
<code>\int_compare:nNnTF</code>	175, 197, 328, 347, 598, 606, 749, 780, 783, 797, 828, 875, 886, 979, 1065, 1076, 1137, 1140, 1171, 1211, 1229, 1262, 1273, 1290, 1305, 1313, 1334, 1346, 1684, 1779, 1802
<code>\int_compare_p:nNn</code>	111, 204, 205, 1091, 1109, 1408, 1824
<code>\int_const:Nn</code>	27
<code>\int_do_while:nNnn</code>	580
<code>\int_eval:n</code> ...	182, 380, 393, 407, 439, 479, 728, 743, 779, 812, 822, 1068, 1079, 1147, 1175, 1316, 1325, 1356, 1366, 1696, 1766, 1782, 1806, 1832
<code>\int_if_odd_p:n</code>	1096, 1114
<code>\int_incr:N</code>	412, 587
<code>\int_mod:nn</code>	888, 895, 1696
<code>\int_new:N</code>	5, 11, 333, 564

<code>\int_set:Nn</code>	308, 363	<code>\kWh</code>	177, <u>42</u>
<code>\int_set_eq:NN</code>	567		
<code>\int_step_inline:nn</code>	343, 636, 652		
<code>\int_use:N</code>	365, 370, 393, 397, 416, 420, 425, 597, 736, 762, 767		
<code>\int_zero:N</code>	340, 571		
<code>inter-unit-product</code>	142		
	J		L
<code>\J</code>	176, <u>42</u>	<code>\L</code>	176, 27
<code>\joule</code>	140, 48, 49, 50, 51, <u>1039</u>	<code>\l</code>	176, 27
		<code>\le</code>	96, 1949
		<code>\leq</code>	1949
		<code>list-exponents</code>	21
		<code>list-final-separator</code>	21
		<code>list-pair-separator</code>	21
		<code>list-separator</code>	21
		<code>list-units</code>	21
		<code>\liter</code>	140, 31, 32, 33, 34, <u>1061</u>
		<code>\litre</code>	140, 27, 28, 29, 30, <u>1061</u>
		<code>\ll</code>	95, 1949
		<code>locale</code>	35
		<code>\LRE</code>	321
		<code>\lumen</code>	140, <u>1039</u>
		<code>\lux</code>	140, <u>1039</u>
	K		M
<code>\K</code>	175, <u>68</u>	<code>\m</code>	175, <u>60</u>
<code>\kA</code>	175, 2	<code>\mA</code>	175, 2
<code>\katal</code>	140, <u>1039</u>	<code>\mathchoice</code>	135, 169, 946
<code>\kelvin</code>	139, 68, <u>1009</u>	<code>\mathord</code>	289, 374, 435, 1603, 1630, 1635, 1672, 1715, 1848
<code>\keV</code>	176, <u>42</u>	<code>\mathrm</code>	89, 135, 36, 77, 207, 1135
<code>keys commands:</code>		<code>\mathsf</code>	96, 179, 193
<code>\l_keys_choice_tl</code>		<code>\mathtt</code>	96, 180, 194
.	18, 46, 70, 276, 290, 366, 372, 411, 421, 464, 468, 541, 640, 656, 1542	<code>\mathversion</code>	89–91, 164
<code>\keys_define:nn</code>	3, 4, 6, 10, 12, 23, 28, 29, 30, 32, 34, 46, 66, 73, 87, 98, 122, 166, 189, 190, 210, 212, 229, 236, 238, 270, 327, 345, 362, 407, 447, 453, 460, 464, 484, 511, 630, 1488	<code>\mbox</code>	89, 135, 380, 391, 392
<code>\l_keys_key_tl</code>		<code>\mdseries</code>	90
.	255, 257, 280, 291, 309, 474, 488	<code>\mebi</code>	181, 2
<code>\keys_set:nn</code>	2, 14, 19, 37, 44, 51, 78, 94, 94, 96, 97, 104, 112, 122, 133, 140, 141, 144, 151, 152, 160, 164, 169, 177, 185, 194, 202, 209, 264, 282, 302, 368, 387, 390, 392, 395, 412, 445, 486, 494, 500, 546, 596, 605, 621, 630, 767, 1122, 1932	<code>\mega</code>	139, 11, 46, 55, 81, 84, 88, <u>1028</u>
<code>\l_keys_value_tl</code>	39, 47, 55, 63, 94, 287, 297, 305, 341, 352, 360, 377, 385, 422, 471, 518, 559	<code>\MessageBreak</code>	18
<code>\kg</code>	175, <u>35</u>	<code>\meter</code>	139, 171, <u>1009</u>
<code>\kHz</code>	175, 8	<code>\metre</code>	139, 140, 153, 171, 60, 61, 62, 63, 64, 65, 66, 67, <u>1009</u>
<code>\kibi</code>	181, 2	<code>\MeV</code>	176, <u>42</u>
<code>\kilo</code>	139, 153, 7, 10, 20, 26, 41, 45, 51, 54, 58, 67, 80, 83, 87, 660, 782, 1009, <u>1028</u>	<code>\meV</code>	176, <u>42</u>
<code>\kilogram</code>	139, <u>1009</u>	<code>\mg</code>	175, <u>35</u>
<code>\kJ</code>	176, <u>42</u>	<code>\mH</code>	75
<code>\km</code>	175, <u>60</u>	<code>\MHz</code>	175, 8
<code>\kmol</code>	176, <u>14</u>	<code>\mHz</code>	175, 8
<code>\kN</code>	176, <u>78</u>	<code>\micro</code>	139, 5, 18, 24, 30, 34, 38, 43, 49, 63, 74, 77, 94, 114, 116, <u>1018</u>
<code>\kohm</code>	177, <u>86</u>	<code>\milli</code>	139, 6, 9, 19, 25, 29, 33, 39, 44, 50, 53, 64, 76, 79, 86, 95, <u>1018</u>
<code>\kPa</code>	177, <u>82</u>	<code>minimum-decimal-digits</code>	40
<code>\kV</code>	176, <u>21</u>	<code>minimum-integer-digits</code>	40
<code>\kW</code>	176, <u>42</u>	<code>\minute</code>	140, <u>1061</u>
		<code>\mJ</code>	176, <u>42</u>
		<code>\mL</code>	176, 27
		<code>\ml</code>	176, <u>27</u>

<code>\mm</code>	175, 60	<code>number-mode</code>	90
<code>\mmol</code>	176, 14	<code>\numlist</code>	127 , 280, 314
<code>\MN</code>	176, 78	<code>\numproduct</code>	127 , 294
<code>\mN</code>	176, 78	<code>\numrange</code>	173, 296, 315
<code>mode</code>	90	<code>\nV</code>	176, 21
mode commands:			
<code>\mode_if_math:TF</code>	107, 160, 207	O	
<code>\mode_leave_vertical:</code>	73, 93, 102, 110, 119, 130, 139, 148, 158, 166, 175, 183, 191, 200, 594, 603, 618, 627	<code>\of</code>	141, 113
<code>\Mohm</code>	177, 86	<code>\ohm</code>	140, 86, 87, 88, 94, 96, 157, 1050
<code>\mohm</code>	177, 86	<code>\Omega</code>	101, 704
<code>\mol</code>	176, 14	<code>output-close-uncertainty</code>	40
<code>\mole</code> ..	139, 14, 15, 16, 17, 18, 19, 20, 1009	<code>output-decimal-marker</code>	40
<code>\mp</code>	93, 292, 293, 1955	<code>output-open-uncertainty</code>	41
<code>\MPa</code>	177, 82	<code>\over</code>	143
<code>\ms</code>	175, 89	<code>overwrite-commands</code>	182
msg commands:			
<code>\msg_error:nn</code>	435	P	
<code>\msg_error:nnn</code>	31, 160, 193, 481, 614, 629	<code>\Pa</code>	177, 82
<code>\msg_error:nnnn</code>	353, 386, 400	<code>\pA</code>	175, 2
<code>\msg_info:nnnn</code>	13, 18, 365, 382, 399, 406, 414, 497	<code>\PackageError</code>	15
<code>\msg_new:nnn</code>	4, 9, 35, 82, 242	<code>parse-numbers</code>	41
<code>\msg_new:nnnn</code>	25, 1080, 1086, 1092, 1098, 1104, 1110, 1116, 1926	<code>parse-units</code>	142
<code>\msg_warning:nn</code>	87	<code>\pascal</code>	140, 82, 83, 84, 85, 1050
<code>\msg_warning:nnn</code>	24, 31, 226, 253	<code>\pdfstringdefDisableCommands</code>	154, 322, 330
<code>\mV</code>	176, 21	<code>\pebi</code>	181, 2
<code>\MW</code>	176, 42	peek commands:	
<code>\mW</code>	176, 42	<code>\peek_after:Nw</code>	349
N		<code>\peek_catcode_ignore_spaces:NTF</code> ..	41, 146, 362
<code>\N</code>	176, 78	<code>\l_peek_token</code>	342
<code>\nA</code>	175, 2	<code>\penalty</code>	146
<code>\nano</code> ...	139, 4, 17, 23, 37, 62, 73, 93, 1018	<code>\per</code>	141–143, 147, 154, 156, 157, 159, 104 , 426, 434, 440, 1093, 1096
<code>negative-color</code>	40	<code>per-mode</code>	142
<code>\neper</code>	140, 1061	<code>per-symbol</code>	142
<code>\newcolumnntype</code>	228, 255	<code>\percent</code>	140, 1077
<code>\NewDocumentCommand</code>	62, 66, 70, 74, 91, 100, 108, 117, 127, 137, 145, 155, 164, 173, 181, 189, 198, 208, 578, 582, 587, 592, 601, 615, 625	<code>\peta</code>	139, 1028
<code>\newton</code>	140, 78, 79, 80, 81, 1050	<code>\pF</code>	177, 70
<code>\nF</code>	177, 70	<code>\pg</code>	175, 35
<code>\ng</code>	175, 35	<code>\pico</code> ...	139, 3, 16, 22, 36, 61, 72, 92, 1018
<code>\nm</code>	175, 60	<code>\pm</code> .	144, 175, 60 , 94, 290, 1763, 1955, 1956
<code>\nmol</code>	176, 14	<code>\pmol</code>	176, 14
<code>\nobreak</code>	147, 194, 277, 610	<code>\power</code>	141
<code>\ns</code>	175, 89	<code>prefix-mode</code>	102
<code>\num</code>	133, 100 , 278, 313	prg commands:	
<code>number-angle-product</code>	13	<code>\prg_new_conditional:Npnn</code>	1260, 1886
<code>number-color</code>	90	<code>\prg_new_protected_conditional:Npnn</code> ..	11, 31, 1869
		<code>\prg_replicate:nn</code>	190, 317, 318, 330, 345, 611, 680, 703, 835, 860, 981, 1283, 1379, 1457, 1791
		<code>\prg_return_false:</code>	19, 46, 1267, 1275, 1879, 1904

<code>\prg_return_true:</code>	18, 42, 1272, 1883, 1921
<code>print-implicit-plus</code>	41
<code>print-unity-mantissa</code>	41
<code>print-zero-exponent</code>	41
<code>\ProcessKeysOptions</code>	56
<code>product-exponents</code>	21
<code>product-mode</code>	21
<code>product-phrase</code>	21
<code>product-symbol</code>	21
prop commands:	
<code>\prop_clear:N</code>	336
<code>\prop_count:N</code>	636
<code>\prop_get:NnN</code>	613
<code>\prop_get:NnNTF</code>	451, 598, 604, 608, 610, 625, 638, 665, 756, 768
<code>\prop_if_empty:NTF</code>	188
<code>\prop_if_in:NnTF</code>	350, 394, 446, 480, 654, 656, 763
<code>\prop_if_in_p:Nn</code>	374
<code>\prop_new:N</code>	62, 63, 331
<code>\prop_put:Nnn</code>	58, 59, 357, 464, 469, 627, 645, 663
<code>\prop_remove:N</code>	643
<code>\prop_remove:Nn</code>	448, 460, 623
<code>propagate-math-font</code>	90
<code>\providecommand</code>	4, 5, 42
<code>\ProvideDocumentCommand</code>	71
<code>\ProvidesExplPackage</code>	23
<code>\ps</code>	175, 89
<code>\pV</code>	176, 21
Q	
<code>\qty</code>	198, 80, 276, 313
<code>\qtylist</code>	127, 287, 314
<code>\qtyproduct</code>	127, 295
<code>\qtyrange</code>	127, 298, 315
<code>qualifier-mode</code>	142
<code>qualifier-phrase</code>	142
<code>quantity-product</code>	102
quark commands:	
<code>\q_mark</code> ...	276, 285, 379, 380, 387, 388, 561, 586, 612, 615, 668, 671, 680, 683, 688, 691, 693, 695, 702, 705
<code>\q_nil</code>	52, 67, 95, 96, 99, 123, 128, 129, 130, 134, 205, 246, 273, 288, 405, 410, 501, 507, 521, 560, 562, 586, 615, 671, 683, 691, 692, 695, 696, 705, 1701, 1703, 1705, 1725, 1785, 1813
<code>\quark_if_nil:NTF</code>	1709, 1719, 1729, 1733, 1737, 1800
<code>\quark_if_nil:NTF</code>	296
<code>\quark_if_recursion_tail_stop:N</code> .	87, 159, 306, 339, 627, 1915
<code>\quark_if_recursion_tail_stop:n</code> .	250, 262, 402
<code>\quark_if_recursion_tail_stop_-</code> <code>do:Nn</code>	231, 331, 378, 400, 484, 528, 539, 1035, 1044, 1060, 1074, 1085, 1106, 1124, 1154, 1228, 1257, 1271, 1303, 1311, 1904
<code>\q_recursion_stop</code>	83, 140, 225, 237, 241, 258, 302, 326, 348, 374, 382, 397, 485, 500, 529, 540, 604, 623, 1030, 1038, 1049, 1053, 1063, 1088, 1134, 1180, 1187, 1224, 1253, 1265, 1294, 1900, 1911
<code>\q_recursion_tail</code>	82, 140, 225, 237, 241, 258, 300, 326, 374, 397, 604, 623, 1030, 1038, 1049, 1053, 1063, 1088, 1134, 1180, 1187, 1224, 1253, 1265, 1294, 1899, 1911
<code>\q_stop</code>	68, 94, 96, 97, 99, 125, 129, 129, 131, 132, 133, 134, 136, 187, 189, 205, 246, 248, 263, 270, 273, 280, 281, 288, 289, 302, 307, 345, 346, 359, 363, 381, 387, 388, 389, 407, 410, 503, 507, 508, 510, 565, 586, 612, 615, 668, 671, 680, 683, 688, 688, 692, 692, 693, 696, 702, 704, 705, 711, 716, 718, 754, 766, 771, 781, 789, 793, 799, 801, 817, 819, 913, 926, 929, 937, 1475, 1476
R	
<code>\radian</code>	140, 1050
<code>\raiseto</code>	141, 116
<code>range-exponents</code>	21
<code>range-phrase</code>	22
<code>range-units</code>	22
<code>\relax</code>	56, 77
<code>\renewcommand</code>	232
<code>\RequirePackage</code>	3, 4, 8, 10, 39, 55, 61, 63, 220
<code>reset-math-version</code>	90
<code>reset-text-family</code>	90
<code>reset-text-series</code>	90
<code>reset-text-shape</code>	90
<code>retain-explicit-plus</code>	41
<code>retain-zero-uncertainty</code>	41
<code>\rmdefault</code>	178
<code>\rmfamily</code>	90
<code>round-half</code>	41
<code>round-minimum</code>	41
<code>round-mode</code>	41
<code>round-pad</code>	41
<code>round-precision</code>	41

S

\s	175, 89
\scriptspace	202, 225, 250
\second	139, 140, 89, 90, 91, 92, 93, 94, 95, 1009
\selectfont	89, 270, 382
separate-uncertainty	41
separate-uncertainty-units	102
seq commands:	
\seq_clear:N	93
\seq_const_from_clist:Nn	311
\seq_count:N	279
\seq_item:Nn	290, 297, 302
\seq_map_function:NN	91
\seq_map_indexed_function:NN	312
\seq_map_inline:Nn	53, 66, 179, 324, 324
\seq_new:N	5, 21, 22
\seq_put_right:Nn	29, 75, 116, 136, 154, 158
\seq_remove_all:Nn	585, 590
\seq_use:Nnnn	28
series-version-mapping	91
\seriesdefault	89, 261
\sfdefault	179
\shapedefault	89, 263
\SI	301, 316, 601
\si	300, 316, 592
\siemens	140, 1050
\sievert	140, 1050
\SIlist	302, 316, 615
\sim	1949
\SIrange	309, 316, 615
\sisetup	208
\SIUnitSymbolAngstrom	656
\SIUnitSymbolArcminute	656
\SIUnitSymbolArcsecond	656
\SIUnitSymbolCelsius	656
\SIUnitSymbolDegree	656
\SIUnitSymbolMicro	656
\SIUnitSymbolOhm	656
siunitx commands:	
\siunitx_angle:n	12, 45, 213
\siunitx_angle:nnn	12, 45, 216, 217
\l_siunitx_bracket_ambiguous_	
bool	39
\siunitx_cell_begin:w	113, 7, 203, 265
\siunitx_cell_end:	113, 7, 58, 79, 205, 267
\siunitx_command_create:	182, 37
\siunitx_complex_number:n	186
\siunitx_complex_quantity:nn	195
\siunitx_compound_number:n	20, 84, 378, 427, 474
\siunitx_compound_quantity:nn	20, 230, 385, 434, 481
\siunitx_declare_power:NNn	137, 40, 64, 584, 589, 1078, 1079
\siunitx_declare_power:NnN	137
\siunitx_declare_prefix:Nn	137, 2, 3, 4, 5, 6, 7, 8, 9, 49, 580
\siunitx_declare_prefix:Nnn	137, 49, 68, 116, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038
\siunitx_declare_qualifier:Nn	137, 64, 72
\siunitx_declare_unit:Nn	137, 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, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 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
\siunitx_declare_unit:Nnn	137, 138, 58, 66, 70, 78, 180
\siunitx_format_number:nN	12
\siunitx_if_number:nTF	38, 1869
\siunitx_if_number_p:n	38
\siunitx_if_number_token:NTF	38, 162, 1886
\siunitx_if_number_token_p:N	1886
\l_siunitx_list_separator_final_	
tl	20, 285, 292, 307, 360, 394
\l_siunitx_list_separator_pair_	
tl	20, 283, 290, 305, 360, 396
\l_siunitx_list_separator_tl	20, 284, 291, 306, 360, 398
\siunitx_number_adjust_exponent:Nn	38, 84, 212, 1462
\siunitx_number_adjust_exponent:nn	38, 1462
\l_siunitx_number_bracket_	
ambiguous_bool	63, 240, 1486, 1491, 1583

<code>\l_siunitx_number_comparator_tl</code> .	39	<code>\siunitx_quantity_list:nn</code>	20, 134, 374, 622
<code>\l_siunitx_number_exponent_tl</code> ...	39	<code>\l_siunitx_quantity_prefix_mode_-</code>	
<code>\siunitx_number_format:nN</code>	38, 14, 113, 153, 762	<code>tl</code>	9, 65, 164, 168, 242
<code>\l_siunitx_number_input_comparator_-</code>		<code>\siunitx_quantity_print:nn</code>	102, 55,
<code>tl</code>	29, 250, 1892		104, 108, 121, 123, 127, 139, 151, 351
<code>\l_siunitx_number_input_decimal_-</code>		<code>\siunitx_quantity_product:nn</code> ...	20, 152, 423
<code>tl</code>	39,	<code>\siunitx_quantity_range:nnn</code>	20, 170, 470, 631
	28, 40, 380, 390, 405, 421, 561, 1890	<code>\l_siunitx_range_phrase_tl</code>	21, 297, 299, 310, 458, 489
<code>\l_siunitx_number_input_exponent_-</code>		<code>\l_siunitx_unit_font_tl</code>	137, 113, 116,
<code>tl</code>	29, 261, 269, 270, 1894		122, 303, 313, 803, 815, 826, 839, 910
<code>\l_siunitx_number_input_sign_tl</code> .		<code>\siunitx_unit_format:nN</code>	102,
	29, 309, 417, 512, 1897		135, 136, 148, 38, 46, 54, 79, 81,
<code>\siunitx_number_list:nn</code> .	20, 142, 374		92, 123, 132, 219, 223, 279, 597, 608
<code>\siunitx_number_normalize_-</code>		<code>\siunitx_unit_format_combine_-</code>	
<code>symbols:N</code>	38, 77, 126	<code>exponent:nnN</code>	136, 75, 132, 175
<code>\siunitx_number_output:N</code>	38, 22, 115, 135, 743, 1549	<code>\siunitx_unit_format_extract_-</code>	
<code>\siunitx_number_output:n</code> ...	38, 1549	<code>prefixes:nnN</code>	136, 80, 132, 198
<code>\siunitx_number_output:NN</code> ..	38, 52,	<code>\siunitx_unit_format_multiply:nnN</code>	136, 132, 224
	67, 123, 129, 405, 501, 560, 562, 1549	<code>\siunitx_unit_format_multiply_-</code>	
<code>\siunitx_number_output:nN</code> ..	38, 1549	<code>combine_exponent:nnnN</code> ..	136, 132, 183
<code>\l_siunitx_number_output_-</code>		<code>\siunitx_unit_format_multiply_-</code>	
<code>decimal_tl</code>	39,	<code>extract_prefixes:nnNN</code> ..	136, 132, 204
	374, 392, 435, 1487, 1529, 1671, 1674	<code>\l_siunitx_unit_fraction_tl</code>	138, 272, 484, 957
<code>\siunitx_number_parse:nN</code>	37, 38, 102, 113, 19, 50, 64,	<code>\siunitx_unit_options_apply:n</code> ...	137, 138, 43, 79, 89, 95,
	105, 108, 152, 303, 498, 542, 668, 1875		121, 132, 150, 168, 193, 341, 620, 629
<code>\l_siunitx_number_parse_bool</code> ...	38, 39, 9,	<code>\siunitx_unit_pdfstring_context:</code>	138, 321, 332
	10, 10, 17, 18, 50, 61, 94, 110, 301, 1874	<code>\siunitx_unit_power_set:NnN</code> ...	147
<code>\siunitx_number_process:N</code>	37	<code>\l_siunitx_unit_seq</code> ..	138, 22, 75, 91
<code>\siunitx_number_process:NN</code>	37, 38, 20, 72, 87, 91,	<code>\l_siunitx_unit_symbolic_seq</code> ...	138, 21, 29, 53, 66, 179, 324, 585, 590
	107, 190, 213, 218, 223, 499, 555, 685	<code>siunitx internal commands:</code>	
<code>\siunitx_number_product:n</code> ..	20, 161, 423	<code>__siunitx_angle:n</code>	277, 344
<code>\siunitx_number_range:nn</code> ..	20, 178, 470	<code>__siunitx_angle:nnn</code>	105, 210
<code>\l_siunitx_number_sign_tl</code>	39	<code>__siunitx_angle:w</code>	344
<code>\siunitx_prin_number:n</code>	90	<code>__siunitx_angle_arc_convert:n</code> ..	45
<code>\siunitx_print_...:n</code>	28, 89-91	<code>__siunitx_angle_arc_print:nnn</code> ..	53, 134, 172
<code>\siunitx_print_match:n</code>	89, 105	<code>__siunitx_angle_arc_print_-</code>	
<code>\siunitx_print_math:n</code> ..	89, 108, 124	<code>auxi:nnn</code>	172
<code>\siunitx_print_number:n</code>	89,	<code>__siunitx_angle_arc_print_-</code>	
	85, 88, 114, 125, 141, 210, 217,	<code>auxii:nw</code>	187, 204
	251, 251, 252, 253, 255, 274, 356,	<code>__siunitx_angle_arc_print_-</code>	
	510, 513, 596, 630, 644, 653, 720, 763	<code>auxii:w</code>	172
<code>\siunitx_print_text:n</code>	89, 68, 88, 103, 109,		
	118, 156, 254, 674, 679, 684, 690, 706		
<code>\siunitx_print_unit:n</code> ..	89, 102,		
	47, 83, 85, 124, 148, 224, 280, 598, 609		
<code>\siunitx_quantity_print:nn</code>	139		
<code>\siunitx_quantity:nn</code>	102, 40, 86, 97, 612		

_siunitx_angle_arc_print_- auxiii:n	172
_siunitx_angle_arc_print_- auxiv:NN	172
_siunitx_angle_arc_print_- auxv:w	172
_siunitx_angle_arc_print_- auxvi:n	172
_siunitx_angle_arc_sign:nn ...	91
_siunitx_angle_arc_sign:nnn	66, 91
\l_siunitx_angle_astronomy_bool	6, 186
\l_siunitx_angle_degrees_tl ...	50, 51, 52, 54, 87, 135
_siunitx_angle_extract_- sign:nnnnnnnn	91
\l_siunitx_angle_fill_degrees_- bool	6
\l_siunitx_angle_fill_minutes_- bool	6
\l_siunitx_angle_fill_seconds_- bool	6
\l_siunitx_angle_force_arc_bool	6, 47
\l_siunitx_angle_force_decimal_- bool	6, 61
\l_siunitx_angle_marker_box ...	170, 214, 228, 232, 238, 240, 244, 249
\l_siunitx_angle_minutes_tl	87, 136
\l_siunitx_angle_product_tl	6, 278
\l_siunitx_angle_seconds_tl	87, 137
\l_siunitx_angle_separator_tl	6, 195
_siunitx_angle_sign:nnnnnnn ...	91
\l_siunitx_angle_sign_tl	90, 101, 105, 114, 163
\l_siunitx_angle_symbol_degree_- tl	6, 175
\l_siunitx_angle_symbol_minute_- tl	6, 177
\l_siunitx_angle_symbol_second_- tl	6, 179
\l_siunitx_angle_tmp_bool	3, 208, 209, 256
\l_siunitx_angle_tmp_dim	3, 241, 263, 265
\l_siunitx_angle_tmp_tl	3, 152, 153, 159, 223, 224, 279, 280
\l_siunitx_angle_unit_box	170, 219, 229, 233, 237, 246
_siunitx_bookmark_cmd:Nn	270
_siunitx_bookmark_cmd:Nnn	270, 276, 277, 278, 279, 280, 287, 294, 295, 296, 298, 300, 301, 302, 309
\c_siunitx_bookmark_seq ...	311, 324
\l_siunitx_column_type_tl ..	46, 260
_siunitx_command_create:	37
_siunitx_command_create:N	37
\l_siunitx_command_create_bool .	4, 39
\l_siunitx_command_optarg_bool .	4, 71, 75
\l_siunitx_command_overwrite_- bool	4, 64
\l_siunitx_command_prespace_- bool	4, 80
\l_siunitx_command_tmp_tl .	3, 82, 84
\l_siunitx_command_xspace_bool .	4, 62, 88
\l_siunitx_compound_bracket_- close_tl	13, 253, 269
\l_siunitx_compound_bracket_- open_tl	13, 251, 267
\l_siunitx_compound_count_int ..	11, 308, 331, 336
\l_siunitx_compound_end_tl	9, 310, 341
\l_siunitx_compound_exp_- bracket_bool	17, 234, 263
\l_siunitx_compound_exp_- combine_bool .	17, 110, 189, 214, 236
\l_siunitx_compound_exp_tl	8, 137, 243, 249, 264, 270, 274
_siunitx_compound_extract_- exp:nN	160
_siunitx_compound_extract_- exp:nnnnnnnN	160
_siunitx_compound_extract_- exponents:	112, 120
_siunitx_compound_extract_- exponents:N	120
_siunitx_compound_extract_- exponents_auxi:w	120
_siunitx_compound_extract_- exponents_auxii:nw	120
_siunitx_compound_extract_- exponents_auxiii:nnnnnnn	120
\l_siunitx_compound_first_tl ...	7, 107, 115, 123, 139, 190, 192, 213, 218, 223
_siunitx_compound_format:n ...	84
_siunitx_compound_format:nn ...	87, 91, 241
_siunitx_compound_format:nnn ..	84
_siunitx_compound_format_- combine-exponent:n	160
_siunitx_compound_format_- combine-exponent:nn	160

__siunitx_compound_format_-	__siunitx_declare_column:Nnn ..	221
combine-exponent_aux:n	__siunitx_emulation_non_latin:n	
160	635 ,
__siunitx_compound_format_-	665 , 667 , 669 , 675 , 680 , 685 , 695 , 711	
extract-exponent:n	__siunitx_emulation_non_-	
160	latin:nnnn	635
__siunitx_compound_format_-	__siunitx_list_aux:	374
extract-exponent:nn	__siunitx_list_count:n	361
160	__siunitx_list_count:w	361
__siunitx_compound_format_-	\l_siunitx_list_exp_tl	360 , 392
extract-exponent_aux:n	\l_siunitx_list_units_tl ..	360 , 400
160	__siunitx_list_use:nnnn	
__siunitx_compound_format_-	282 , 289 , 304 , 361
input:n	__siunitx_list_use_aux:nnnn ..	361
221	__siunitx_list_use_auxi:nw	
__siunitx_compound_format_-	374 , 375 , 384
units:nn	__siunitx_list_use_auxi:w	361
108 , 160	__siunitx_list_use_auxii:nnw ..	361
__siunitx_compound_parsed:n	__siunitx_list_use_auxiii:nnw ..	361
118 , 151	__siunitx_load_check:	25
__siunitx_compound_print:N	__siunitx_load_check:n ...	28 , 36 , 40
.....	__siunitx_number_adjust_exp:nn	1462
88 , 245 , 252 , 255 , 260	__siunitx_number_adjust_-	
__siunitx_compound_print:nnN ..	exp:nnnnnnnn	1462
260	__siunitx_number_adjust_exp:nw	
__siunitx_compound_print:nnnN	1462
260	\l_siunitx_number_arg_tl ...	49 ,
__siunitx_compound_print_aux:n	52 , 69 , 119 , 125 , 126 , 127 , 246 , 253 ,	
260	256 , 272 , 287 , 299 , 373 , 508 , 514 , 522	
__siunitx_compound_print_aux:nn	\l_siunitx_number_bracket_-	
260	close_tl	1482 , 1597 , 1652
__siunitx_compound_print_-	\l_siunitx_number_bracket_-	
quantity:n	negative_bool	1488 , 1618
245 , 256 , 350	\l_siunitx_number_bracket_open_-	
__siunitx_compound_print_-	tl	1482 , 1595 , 1650
separator:n	\l_siunitx_number_comparator_tl	
299 , 329 , 334 , 347 , 352 , 358	70 , 252 , 255 , 356
\l_siunitx_compound_separator_-	__siunitx_number_digits:NN	694 , 958
final_tl	__siunitx_number_digits:Nn ...	958
17 , 334	__siunitx_number_digits:nn ...	958
\l_siunitx_compound_separator_-	__siunitx_number_digits:nnnnnnn	958
pair_tl	__siunitx_number_digits_S:n ..	958
17 , 299	__siunitx_number_digits_-	
\l_siunitx_compound_separator_-	uncert:nn	974 , 983
text_bool	__siunitx_number_digits_uncert_-	
17 , 354	S:n	988
\l_siunitx_compound_separator_-	__siunitx_number_drop_exponent:NN	
tl	692 , 993
17 , 329 , 347	__siunitx_number_drop_exponent:nnnnnnn	
\l_siunitx_compound_start_tl	993
.....	\l_siunitx_number_drop_exponent_-	
9 , 309 , 326	bool	630 , 995
\l_siunitx_compound_tmp_fp	__siunitx_number_drop_uncertainty:NN	
.....	690 , 1003
4 , 192 , 193 , 210 , 212		
\l_siunitx_compound_tmp_seq ...		
.....		
4 , 93 , 116 ,		
136 , 154 , 158 , 279 , 290 , 297 , 302 , 313		
\l_siunitx_compound_tmp_tl		
..		
4 , 105 , 107 , 114 , 116 , 122 , 125 ,		
153 , 154 , 190 , 211 , 212 , 213 , 218 , 223		
\l_siunitx_compound_unit_-		
bracket_bool		
17 , 233 , 238 , 248		
\l_siunitx_compound_unit_power_-		
bool		
21 , 56 , 62 , 68 , 74 , 80 , 162		
\l_siunitx_compound_unit_-		
repeat_bool		
17 , 235 , 239 , 244		
\l_siunitx_compound_unit_tl ...		
.....		
12 , 193 , 210 , 219 , 224 , 351		
__siunitx_compound_unparsed:n ..		
.....		
101 , 151		

__siunitx_number_drop_uncertainty:nnnnnnn__siunitx_number_exponent_-	
.....	1003
\l_siunitx_number_drop_uncertainty_-	
bool	630 , 1005
\l_siunitx_number_drop_zero_-	
decimal_bool	630 , 1446
\l_siunitx_number_explicit_-	
plus_bool	29 , 318 , 517
__siunitx_number_exponent:NN ...	
.....	706 , 710
\l_siunitx_number_exponent_-	
base_tl	1488 , 1853
__siunitx_number_exponent_-	
engineering:nnnnnnn	710
__siunitx_number_exponent_-	
engineering:nnNw	710
__siunitx_number_exponent_-	
engineering_0:nnnn	710
__siunitx_number_exponent_-	
engineering_1:nnnn	710
__siunitx_number_exponent_-	
engineering_2:nnnn	710
__siunitx_number_exponent_-	
engineering_aux:nnnnnnn	710
__siunitx_number_exponent_-	
engineering_uncert:nn	710
__siunitx_number_exponent_-	
engineering_uncert_S:nnn	710
__siunitx_number_exponent_-	
finalise:n	710 , 1243
__siunitx_number_exponent_-	
fixed:nnnnnnn	710
__siunitx_number_exponent_-	
fixed:nnnnnnnn	710
\l_siunitx_number_exponent_-	
fixed_int	630 , 728 , 736
__siunitx_number_exponent_-	
input:nnnnnnn	710
\l_siunitx_number_exponent_-	
mode_tl	630 , 715 , 719 , 1203
\l_siunitx_number_exponent_-	
product_tl	1488 , 1850
__siunitx_number_exponent_-	
scientific:nnnnnnn	710
__siunitx_number_exponent_-	
scientific:nnnnnnnn	710
__siunitx_number_exponent_-	
scientific:nnnw	710
__siunitx_number_exponent_-	
shift:nnn	710 , 1237
__siunitx_number_exponent_-	
shift_down:nnn	710
__siunitx_number_exponent_-	
shift_down:nnnw	710
__siunitx_number_exponent_-	
shift_down:nw	710
__siunitx_number_exponent_-	
shift_uncert:nw	710
__siunitx_number_exponent_-	
shift_uncert_S:nnnn	710
__siunitx_number_exponent_-	
shift_up:nnn	710
__siunitx_number_exponent_-	
shift_up:nnw	710
__siunitx_number_exponent_-	
shift_up_aux:nnn	710
\l_siunitx_number_exponent_tl ..	
.....	71 , 263 , 297 , 312 , 320 , 321 , 332 , 342 , 359
__siunitx_number_exponent_-	
uncert:n	710
__siunitx_number_expression:n ..	
.....	29 , 122
\l_siunitx_number_expression_-	
bool	29 , 121
\l_siunitx_number_flex_tl	46 , 72 , 135 , 144 , 148 , 170 , 178 , 464 , 466 , 502
\l_siunitx_number_group_-	
decimal_bool	1488
\l_siunitx_number_group_-	
integer_bool	1488
\l_siunitx_number_group_-	
minimum_int	1488 , 1685
\l_siunitx_number_group_-	
separator_tl	1488 , 1714 , 1717 , 1742
__siunitx_number_if_token_-	
auxi:NN	1886
__siunitx_number_if_token_-	
auxii:NN	1886
__siunitx_number_if_token_-	
auxiii:NN	1886
\l_siunitx_number_implicit_-	
plus_bool	1488 , 1610 , 1858
\l_siunitx_number_input_digit_-	
tl .	29 , 340 , 384 , 402 , 486 , 541 , 1893
\l_siunitx_number_input_ignore_-	
tl	29 , 1895
\l_siunitx_number_input_tl	
.....	74 , 125 , 161
\l_siunitx_number_input_uncert_-	
close_tl ...	29 , 530 , 555 , 568 , 1891
\l_siunitx_number_input_uncert_-	
open_tl	29 , 412 , 1896
\l_siunitx_number_input_uncert_-	
sign_tl	29 , 149 , 1898
\l_siunitx_number_min_decimal_-	
int	630 , 972 , 991

\l_siunitx_number_min_integer_- int	630 , 967	_siunitx_number_output_sign_- brackets:w	1549
\l_siunitx_number_negative_- color:tl	1488 , 1616 , 1641	_siunitx_number_output_sign_- color:w	1549
_siunitx_number_normalize_- aux:nN	77	\l_siunitx_number_output_- uncert_close:tl	1488 , 1773
_siunitx_number_normalize_- aux:NnN	80 , 85 , 89	\l_siunitx_number_output_- uncert_open:tl	1488 , 1771
_siunitx_number_normalize_- minus:N	79 , 102	_siunitx_number_output_uncert_- S:nnnw	1549
_siunitx_number_normalize_- sign:N	77	_siunitx_number_output_uncert_- S:nnw	1549
\c_siunitx_number_normalize_tl .	77	_siunitx_number_output_uncert_- S_aux:nnn	1549
_siunitx_number_output:Nn ..	1549	_siunitx_number_output_uncert_- S_aux:nnnw .	1781 , 1798 , 1805 , 1812
_siunitx_number_output:nn ..	1549	_siunitx_number_output_uncert_- S_aux:nnw	1803 , 1813
_siunitx_number_output:nnnnnnn	1549	_siunitx_number_output_uncert_- S_compact-marker:nn	1549
_siunitx_number_output_- bracket:nn	1549	_siunitx_number_output_uncert_- S_compact:nn	1549
_siunitx_number_output_- bracket:w	1549	_siunitx_number_output_uncert_- S_full:nn	1549
_siunitx_number_output_- comparator:nn	1549	_siunitx_number_output_- uncertainty:nnn	1549
_siunitx_number_output_- decimal:nn	1549	_siunitx_number_output_- uncertainty_unaligned:n ...	1549
_siunitx_number_output_- decimal_aux:n	1549	\l_siunitx_number_outputted_tl	8 , 21 , 24 , 26
_siunitx_number_output_- decimal_loop:NNNN	1549	_siunitx_number_parse:nN	108
_siunitx_number_output_- digits:nn	1549	_siunitx_number_parse_check:	129 , 133
_siunitx_number_output_end: .	1549	_siunitx_number_parse_combine_- uncert:	151 , 166
_siunitx_number_output_- exponent:nnnn	1549	_siunitx_number_parse_combine_- uncert_auxi:nnnnnnnn	166
_siunitx_number_output_- integer:nnn	1549	_siunitx_number_parse_combine_- uncert_auxii:nnnnn	166
_siunitx_number_output_- integer_aux:n	1549	_siunitx_number_parse_combine_- uncert_auxiii:nnnnnn	166
_siunitx_number_output_- integer_aux_0:n	1549	_siunitx_number_parse_combine_- uncert_auxiv:nnnn	166
_siunitx_number_output_- integer_aux_1:n	1549	_siunitx_number_parse_combine_- uncert_auxv:w	166
_siunitx_number_output_- integer_aux_2:n	1549	_siunitx_number_parse_combine_- uncert_auxvi:w	166
_siunitx_number_output_- integer_first:nnNN	1549	_siunitx_number_parse_comparator:	128 , 243
_siunitx_number_output_- integer_loop:NNNN	1549	_siunitx_number_parse_comparator_- aux:Nw	243
_siunitx_number_output_sign:N	1549	_siunitx_number_parse_exponent:	259 , 524
_siunitx_number_output_sign:nN	1549		
_siunitx_number_output_- sign:nnn	1549		

_siunitx_number_parse_exponent_- auxi:w	259	_siunitx_number_parse_uncert:NNNN	526
_siunitx_number_parse_exponent_- auxii:nn	259	_siunitx_number_parse_uncert_- after:N	526
_siunitx_number_parse_exponent_- auxiii:Nw	259	_siunitx_number_parse_uncert_- auxi:NN	526
_siunitx_number_parse_exponent_- auxiv:nn	259	_siunitx_number_parse_uncert_- auxii:N	526
_siunitx_number_parse_exponent_- check:N	259	_siunitx_number_parse_uncert_- auxii:NN	526
_siunitx_number_parse_exponent_- cleanup:N	259	_siunitx_number_parse_uncert_- auxiii:N	557, 570, 575
_siunitx_number_parse_exponent_- cleanup:wN	345, 347	_siunitx_number_parse_uncert_- marker:N	526
_siunitx_number_parse_exponent_- zero_test:N	259	_siunitx_number_parse_uncert_- marker:nnnN	595, 596
_siunitx_number_parse_finalise:	164, 350	_siunitx_number_parse_uncert_- marker:nNw	601, 603
_siunitx_number_parse_finalise:nw	350	\l_siunitx_number_parsed_tl	47, 19, 20, 22, 73, 118, 131, 140, 152, 154, 156, 170, 177, 212, 214, 264, 301, 304, 313, 323, 349, 352, 354, 357, 372, 503, 518, 519, 521, 523, 1875, 1876
_siunitx_number_parse_loop:	265, 305, 368	\l_siunitx_number_partial_tl	75, 370, 432, 433, 436, 446, 470, 471, 474, 478, 488, 546, 577, 588, 589, 599, 610, 611
_siunitx_number_parse_loop_- after_decimal:NNN	368	_siunitx_number_process:nnnnnnnnNN	685
_siunitx_number_parse_loop_- break:wN	368, 529, 531, 540, 563, 573, 600, 622, 628	_siunitx_number_round:NN	707, 1015
_siunitx_number_parse_loop_- first:N	368	_siunitx_number_round:nnn	1027, 1315, 1354, 1364, 1425, 1434, 1439
_siunitx_number_parse_loop_- first:NNN	371, 376, 465	_siunitx_number_round_auxi:nnnN	1027
_siunitx_number_parse_loop_- main:NNNN	368	_siunitx_number_round_auxii:nnnN	1027
_siunitx_number_parse_loop_- main_decimal:NN	368	_siunitx_number_round_auxiii:nnnN	1027
_siunitx_number_parse_loop_- main_digit:NNNN	368	_siunitx_number_round_auxiv:nnN	1027
_siunitx_number_parse_loop_- main_end:NN	368	_siunitx_number_round_auxiv:nnnN	1048, 1062, 1072, 1078
_siunitx_number_parse_loop_- main_sign:NNN	368	_siunitx_number_round_auxv:nnN	1027
_siunitx_number_parse_loop_- main_store:NNN	368, 590	_siunitx_number_round_auxvi:nN	1027
_siunitx_number_parse_loop_- main_uncert:NNN	368	_siunitx_number_round_auxvi:nnN	1087
_siunitx_number_parse_loop_- root_swap:NwNN	368	_siunitx_number_round_auxvii:nnN	1081, 1104
_siunitx_number_parse_sign:	257, 505	_siunitx_number_round_auxviii:nnN	1027
_siunitx_number_parse_sign_- aux:Nw	505		
_siunitx_number_parse_uncert:NN	459, 526		

_siunitx_number_round_auxviii:nnN	_siunitx_number_round_places_-
..... 1027	finalise:n 1327
_siunitx_number_round_engineering:nn	_siunitx_number_round_places_-
..... 1027	finalise:nnnnn 1327
_siunitx_number_round_engineering:nnN	_siunitx_number_round_places_-
..... 1027	finalise:nnnnnnn 1327
_siunitx_number_round_engineering:NNNNn	_siunitx_number_round_places_-
..... 1027	integer:nn 1327
_siunitx_number_round_figures:nnnnnnn	\l_siunitx_number_round_-
..... 1286	precision_int 630,
_siunitx_number_round_figures_-	1223, 1290, 1313, 1316, 1321, 1334,
count:nnN 1286	1347, 1350, 1357, 1367, 1408, 1426
_siunitx_number_round_figures_-	_siunitx_number_round_scientific:nn
count:nnnN 1286 1235
_siunitx_number_round_final:nn	_siunitx_number_round_scientificc:nn
..... 1027 1027
_siunitx_number_round_final_-	_siunitx_number_round_truncate:n
decimal:nnw 1027 1027
_siunitx_number_round_final_-	_siunitx_number_round_truncate:nnN
integer:nnw 1027 1027
_siunitx_number_round_final_-	_siunitx_number_round_truncate_-
output:nn 1027	direct:n 1027, 1441
_siunitx_number_round_final_-	_siunitx_number_round_uncertainty:nnn
shift:nn 1027 1404
_siunitx_number_round_final_-	_siunitx_number_round_uncertainty:nnnnn
shift:Nw 1027 1404
_siunitx_number_round_fixed:nn	_siunitx_number_round_uncertainty:nnnnnnn
..... 1027 1404
\l_siunitx_number_round_half_-	_siunitx_number_set_round_-
even_bool 630, 1095, 1113	min:n 653, 666
_siunitx_number_round_if_-	_siunitx_number_set_round_-
half:N 1260	min:nnnnnnn 666
_siunitx_number_round_if_-	\l_siunitx_number_tight_bool ...
half:n 1260 1488, 1634, 1847
_siunitx_number_round_if_half_-	\l_siunitx_number_tmp_tl
p:n 1097, 1115, 1260 7, 147, 150,
_siunitx_number_round_input:nn	268, 273, 279, 280, 288, 289, 668, 669
..... 1027	_siunitx_number_token_auxi:NN .
\l_siunitx_number_round_min_tl 1889, 1902, 1906
..... 665, 673, 1382, 1390	_siunitx_number_token_auxii:NN
\l_siunitx_number_round_mode_tl 1905, 1908
..... 630, 1020, 1200, 1246	_siunitx_number_token_auxiii:NN
_siunitx_number_round_none:nnnnnnn 1910, 1913, 1924
..... 1015	\l_siunitx_number_uncert_mode_-
_siunitx_number_round_pad:nnn .	tl 1488, 1582, 1760, 1772
..... 1277, 1320, 1349	\l_siunitx_number_uncert_-
\l_siunitx_number_round_pad_-	separator_tl 1488, 1770
bool 630, 1282	\l_siunitx_number_unity_-
_siunitx_number_round_places:nnnnnnn	mantissa_bool ... 1488, 1662, 1844
..... 1327	\l_siunitx_number_valid_tl .. 1868
_siunitx_number_round_places_-	\l_siunitx_number_validate_bool
decimal:nn 1327 76, 158, 1873
_siunitx_number_round_places_-	_siunitx_number_zero_decimal:NN
end:nn 1242, 1327 693, 1444

<code>__siunitx_number_zero_decimal:nnnnnnn</code>	<code>__siunitx_print_math_text:n</code>	124
..... 1444	<code>__siunitx_print_math_version:nn</code>	124
<code>\l_siunitx_number_zero_exponent_-</code>	<code>\l_siunitx_print_math_version_-</code>	
bool 1488, 1587, 1839	bool 32, 151	
<code>\l_siunitx_number_zero_uncert_-</code>	<code>\l_siunitx_print_math_weight_-</code>	
bool 29, 233, 582	bool 32, 126	
<code>__siunitx_option_deprecated:nn</code>	<code>\c_siunitx_print_mathrm_int</code>	8, 205
..... 11, 70,	<code>\c_siunitx_print_mathsf_int</code>	8, 193
76, 82, 102, 116, 125, 131, 145, 151,	<code>\c_siunitx_print_mathtt_int</code>	8, 194
217, 223, 243, 249, 261, 267, 276,	<code>\l_siunitx_print_number_color_-</code>	
313, 319, 325, 331, 477, 483, 491,	tl 32	
506, 524, 530, 538, 544, 550, 564, 570	<code>\l_siunitx_print_number_mode_tl</code>	32
<code>__siunitx_option_deprecated:nnn</code>	<code>__siunitx_print_replace_font:N</code>	
.. 11, 36, 44, 52, 60, 91, 284, 294, 111, 214, 236, 287	
302, 338, 349, 357, 374, 468, 515, 556	<code>__siunitx_print_store_fam:n</code>	8
<code>__siunitx_option_removed:n</code>	<code>\l_siunitx_print_text_family_-</code>	
..... 22, 30, 166, 190, 210,	bool 53, 258, 266	
236, 255, 257, 280, 291, 309, 474, 488	<code>\l_siunitx_print_text_font_tl</code>	
<code>__siunitx_option_table_comparator:nnnnnnn</code> 32, 271	
..... 412	<code>__siunitx_print_text_fraction:Nnn</code>	
<code>__siunitx_option_table_comparator:nnnnnnnn</code> 254	
..... 427	<code>__siunitx_print_text_replace:N</code>	254
<code>__siunitx_option_table_figures-decimal:nnnnnnnn</code>	<code>__siunitx_print_text_replace:n</code>	254
..... 412	<code>__siunitx_print_text_replace:Nnn</code>	
<code>__siunitx_option_table_figures-exponent:nnnnnnnn</code> 254	
..... 412	<code>__siunitx_print_text_scripts:</code>	254
<code>__siunitx_option_table_figures-integer:nnnnnnnn</code>	<code>__siunitx_print_text_scripts:Nnn</code>	
..... 412 254	
<code>__siunitx_option_table_figures-uncertainty:nnnnnnnn</code>	<code>__siunitx_print_text_scripts_-</code>	
..... 412	one:Nn 345, 352, 383	
<code>__siunitx_option_table_format:n</code>	<code>__siunitx_print_text_scripts_-</code>	
412, 450, 452, 454, 456, 458, 460, 462	one:NnN 254	
<code>__siunitx_option_table_sign-exponent:nnnnnnnn</code>	<code>__siunitx_print_text_scripts_-</code>	
..... 412	two:n 254	
<code>__siunitx_option_table_sign-mantissa:nnnnnnnn</code>	<code>__siunitx_print_text_scripts_-</code>	
..... 412	two:nn 254	
<code>__siunitx_print_aux:nn</code>	<code>__siunitx_print_text_scripts_-</code>	
..... 85	two:NnN 254	
<code>__siunitx_print_convert_-</code>	<code>\l_siunitx_print_text_series_-</code>	
series:n 124	bool 55, 260, 267	
<code>__siunitx_print_extract_-</code>	<code>\l_siunitx_print_text_shape_-</code>	
series:Nw 124	bool 57, 262, 268	
<code>__siunitx_print_math_aux:Nn</code>	<code>__siunitx_print_text_sub:n</code>	254
..... 124	<code>__siunitx_print_text_super:n</code>	254
<code>__siunitx_print_math_auxi:n</code>	<code>\l_siunitx_print_tmp_box</code>	6, 22
..... 124	<code>\l_siunitx_print_tmp_tl</code>	6, 128,
<code>__siunitx_print_math_auxii:n</code>	130, 132, 213, 214, 215, 218, 221,	
..... 124	235, 236, 237, 246, 247, 249, 280,	
<code>__siunitx_print_math_auxiii:n</code>	281, 282, 319, 320, 321, 355, 356, 358	
..... 124	<code>\l_siunitx_print_unit_color_tl</code>	32
<code>__siunitx_print_math_auxiv:n</code>	<code>\l_siunitx_print_unit_mode_tl</code>	32
..... 124	<code>\l_siunitx_print_version_b_tl</code>	73
<code>__siunitx_print_math_auxv:n</code>	<code>\l_siunitx_print_version_eb_tl</code>	73
..... 124	<code>\l_siunitx_print_version_el_tl</code>	73
<code>\l_siunitx_print_math_family_-</code>		
bool 32, 174		
<code>\l_siunitx_print_math_font_bool</code>		
..... 32, 189		
<code>__siunitx_print_math_script:n</code>		
..... 124		
<code>__siunitx_print_math_sub:n</code>		
..... 124		
<code>__siunitx_print_math_super:n</code>		
..... 124		

\l__siunitx_print_version_l_tl ..	73
\l__siunitx_print_version_m_tl ..	73
\l__siunitx_print_version_sb_tl ..	73
\l__siunitx_print_version_sl_tl ..	73
\l__siunitx_print_version_ub_tl ..	73
\l__siunitx_print_version_ul_tl ..	73
\c__siunitx_print_weight_b_tl ..	122
\c__siunitx_print_weight_c_tl ..	120
\c__siunitx_print_weight_ecl_tl ..	120
\c__siunitx_print_weight_ex_tl ..	120
\c__siunitx_print_weight_l_tl ..	122
\c__siunitx_print_weight_m_tl ..	122
\c__siunitx_print_weight_sc_tl ..	120
\c__siunitx_print_weight_sx_tl ..	120
\c__siunitx_print_weight_uc_tl ..	120
\c__siunitx_print_weight_ux_tl ..	120
\c__siunitx_print_weight_x_tl ..	120
__siunitx_product_aux:	423
__siunitx_product_aux:n	423
\l__siunitx_product_exp_tl ..	404 , 447
\l__siunitx_product_phrase_bool ..	404 , 439 , 452
\l__siunitx_product_phrase_tl ..	404 , 440
\l__siunitx_product_symbol_tl ..	404 , 441
\l__siunitx_product_units_tl ..	404 , 453
\l__siunitx_quantity_bracket_-	
close_tl	5 , 113
\l__siunitx_quantity_bracket_-	
open_tl	5 , 111
\l__siunitx_quantity_break_bool ..	9 , 145
__siunitx_quantity_extract_-	
exp:nNN	73 , 131
__siunitx_quantity_extract_-	
exp:nnnnnnnNN	131
__siunitx_quantity_non_latin:n ..	159 , 181
__siunitx_quantity_non_latin:nnnn	159
\l__siunitx_quantity_number_tl ..	38 , 53 , 56 , 64 , 66 , 67 , 68 , 72 , 74 , 82 , 85 , 87 , 91
__siunitx_quantity_parsed:nm ...	40
__siunitx_quantity_parsed_-	
aux:nnnn	40
__siunitx_quantity_parsed_-	
aux:nnnw	40
__siunitx_quantity_parsed_aux:w ..	40
__siunitx_quantity_parsed_-	
combine-exponent:n	40
__siunitx_quantity_parsed_-	
input:n	40
\l__siunitx_quantity_product_tl ..	9 , 144
\l__siunitx_quantity_tmp_fp	3 , 74 , 76 , 81 , 85
\l__siunitx_quantity_tmp_tl	3
\l__siunitx_quantity_uncert_-	
bracket_bool	9 , 106
\l__siunitx_quantity_uncert_-	
repeat_bool	9 , 119
\l__siunitx_quantity_unit_tl ..	38 , 46 , 47 , 54 , 56 , 76 , 81 , 92 , 104 , 116 , 122 , 124 , 127
__siunitx_range_aux:	470
\l__siunitx_range_exp_tl ..	458 , 488
\l__siunitx_range_units_tl ..	458 , 491
__siunitx_symbol_if_replace:Nn ..	31
__siunitx_symbol_if_replace:NnTF	31 , 51 , 56 , 61 , 94 , 114
__siunitx_symbol_non_latin:n ...	10 , 34 , 69 , 75 , 89 , 108 , 123 , 158
__siunitx_symbol_non_latin:nnnn	10
\l__siunitx_symbol_tmpa_tl	3 , 34 , 35 , 36 , 39 , 75 , 76 , 77 , 80 , 132 , 134 , 135 , 137
\l__siunitx_symbol_tmpp_tl	3 , 38 , 39 , 79 , 80 , 136 , 137
\l__siunitx_table_after_box ..	450 , 477 , 479 , 483 , 490 , 493 , 504 , 567 , 580
\l__siunitx_table_after_model_tl	284 , 297 , 402 , 566
\l__siunitx_table_after_tl	102 , 113 , 120 , 160
\l__siunitx_table_align_after_-	
bool	453 , 570
__siunitx_table_align_auxi:nn ..	196
__siunitx_table_align_auxii:nn	196
\l__siunitx_table_align_before_-	
bool	453 , 589 , 634
__siunitx_table_align_center:n	196
\l__siunitx_table_align_comparator_-	
bool	453 , 619
\l__siunitx_table_align_exponent_-	
bool	453 , 712
__siunitx_table_align_left:n ..	196
\l__siunitx_table_align_mode_tl ..	270 , 366 , 370 , 467
\l__siunitx_table_align_number_-	
tl	270 , 442 , 575 , 759
__siunitx_table_align_right:n ..	196
\l__siunitx_table_align_text_tl ..	229 , 239
\l__siunitx_table_align_uncertainty_-	
bool	453 , 701

<code>\l_siunitx_table_auto_round_-</code>	<code>_siunitx_table_direct_format_-</code>
bool 270 , 544	switch: 357
<code>\l_siunitx_table_before_box</code> ...	<code>_siunitx_table_direct_marker:</code> 357
.... 450 , 471 , 472 , 474 , 480 , 482 ,	<code>_siunitx_table_direct_marker_-</code>
486 , 491 , 497 , 530 , 531 , 533 , 536 ,	end: 357
540 , 577 , 601 , 603 , 609 , 655 , 657 , 664	<code>_siunitx_table_direct_marker_-</code>
<code>\l_siunitx_table_before_model_-</code>	switch: 357
tl 282 , 295 , 409 , 529	<code>_siunitx_table_direct_none:</code> .. 357
<code>\l_siunitx_table_before_tl</code>	<code>_siunitx_table_direct_none_-</code>
..... 102 , 111 , 115 , 118	end: 357
<code>\l_siunitx_table_carry_dim</code>	<code>_siunitx_table_fil:</code> .. 244 , 258 ,
.... 452 , 568 , 571 , 675 , 730 , 738 , 750	265 , 377 , 416 , 426 , 440 , 485 , 494 ,
<code>_siunitx_table_center_marker:</code> .	539 , 573 , 594 , 608 , 629 , 642 , 663 , 735
..... 248 , 398 , 516	<code>\l_siunitx_table_fixed_width_-</code>
<code>_siunitx_table_cleanup_-</code>	bool 189 , 204
decimal:w 245 , 413	<code>\l_siunitx_table_format_tl</code> 294 ,
<code>_siunitx_table_collect_begin:</code> .	303 , 305 , 306 , 309 , 417 , 421 , 425 , 552
..... 11 , 24	<code>_siunitx_table_generate_-</code>
<code>_siunitx_table_collect_begin:N</code> 124	model:n 285 , 298
<code>_siunitx_table_collect_begin:w</code> 24	<code>_siunitx_table_generate_-</code>
<code>_siunitx_table_collect_end:</code> 19 , 105	model:nnnnnn 298 , 424
<code>_siunitx_table_collect_end:n</code> . 105	<code>_siunitx_table_generate_model_-</code>
<code>_siunitx_table_collect_end:w</code> . 105	S:nnn 298
<code>_siunitx_table_collect_end_-</code>	<code>_siunitx_table_generate_model_-</code>
aux:n 105	S:nnw 298
<code>_siunitx_table_collect_group:n</code> 39	<code>\l_siunitx_table_integer_box</code> ...
<code>_siunitx_table_collect_loop:</code> 242 , 251 , 255 , 262 ,
..... 31 , 38 , 39	266 , 378 , 399 , 418 , 444 , 509 , 517 ,
<code>_siunitx_table_collect_-</code>	578 , 591 , 600 , 605 , 620 , 622 , 624 ,
search:NnTF 39	628 , 636 , 638 , 643 , 649 , 650 , 652 , 660
<code>_siunitx_table_collect_search_-</code>	<code>\l_siunitx_table_model_tl</code>
aux:NNn 39 283 , 285 , 294 , 314 , 405 , 560
<code>\l_siunitx_table_collect_tl</code> ...	<code>\l_siunitx_table_number_tl</code>
.... 23 , 27 , 47 , 61 , 82 , 108 , 110 , 129 102 , 112 , 114 , 119
<code>_siunitx_table_collect_token:N</code> 39	<code>_siunitx_table_print:nnn</code> . 117 , 466
<code>\l_siunitx_table_column_width_-</code>	<code>_siunitx_table_print_format:nnn</code>
dim 189 , 205 466
<code>\l_siunitx_table_decimal_box</code> ...	<code>_siunitx_table_print_format:nnnnnn</code>
. 242 , 252 , 254 , 257 , 263 , 375 , 390 , 551 , 583
400 , 414 , 432 , 433 , 445 , 511 , 518 ,	<code>_siunitx_table_print_format_-</code>
579 , 674 , 678 , 727 , 729 , 734 , 745 , 747	after:N 466
<code>_siunitx_table_direct_begin:</code> ..	<code>_siunitx_table_print_format_-</code>
..... 12 , 357	auxi:w 564 , 585
<code>_siunitx_table_direct_begin:w</code> 357	<code>_siunitx_table_print_format_-</code>
<code>_siunitx_table_direct_end:</code> 20 , 357	auxii:w 612 , 614
<code>_siunitx_table_direct_format:</code> 357	<code>_siunitx_table_print_format_-</code>
<code>_siunitx_table_direct_format:nnnnnnn</code>	auxiii:w 668 , 670
..... 357	<code>_siunitx_table_print_format_-</code>
<code>_siunitx_table_direct_format:w</code> 357	auxiv:w 680 , 682
<code>_siunitx_table_direct_format_-</code>	<code>_siunitx_table_print_format_-</code>
aux:w 406 , 409	auxv:w 686 , 690
<code>_siunitx_table_direct_format_-</code>	<code>_siunitx_table_print_format_-</code>
end: 357	auxvi:w 687 , 694

<code>__siunitx_table_print_format_-</code>		<code>\l__siunitx_unit_combine_exp_fp .</code>	
<code>auxvii:w</code>	693, 702, 704	135,
<code>__siunitx_table_print_format_-</code>		142, 149, 156, 164, 172, 557, 572, 607	
<code>box:Nn</code>	466	<code>\l__siunitx_unit_current_tl</code> 561,	
<code>__siunitx_table_print_marker:nmn</code>		584, 722, 724, 740, 742, 782, 784,	
.....	466	787, 795, 833, 840, 848, 900, 908, 911	
<code>__siunitx_table_print_marker:w</code>	466	<code>\l__siunitx_unit_denominator_-</code>	
<code>__siunitx_table_print_marker_-</code>		<code>bracket_bool</code>	484, 872
<code>aux:w</code>	466	<code>\l__siunitx_unit_denominator_tl .</code>	
<code>__siunitx_table_print_marker_-</code>		563, 568, 873, 918, 959, 968, 977, 984	
<code>auxi:w</code>	466	<code>\l__siunitx_unit_font_bool</code>	
<code>__siunitx_table_print_marker_-</code>		550, 585, 837, 843, 906, 913
<code>auxii:w</code>	466	<code>\l__siunitx_unit_forbid_literal_-</code>	
<code>__siunitx_table_print_marker_-</code>		<code>bool</code>	192, 484
<code>auxiii:w</code>	466	<code>__siunitx_unit_format:nNN</code>	132
<code>__siunitx_table_print_none:nmn</code>	466	<code>__siunitx_unit_format_aux: . . .</code>	132
<code>__siunitx_table_print_text:n . . .</code>		<code>__siunitx_unit_format_bracket:N</code>	
.....	115, 236, 363	674, 724, 742, 968
<code>__siunitx_table_skip:n</code>		<code>__siunitx_unit_format_combine_-</code>	
.....	184, 208, 210, 222, 224	<code>exp:</code>	573, 602
<code>__siunitx_table_split:nNNN</code>		<code>__siunitx_unit_format_finalise:</code>	
.....	109, 135, 281	592, 916
<code>__siunitx_table_split_group:NNNn</code>		<code>__siunitx_unit_format_finalise_-</code>	
.....	135	<code>autofrac:</code>	916
<code>__siunitx_table_split_loop:NNN</code>	135	<code>__siunitx_unit_format_finalise_-</code>	
<code>__siunitx_table_split_tidy:N . . .</code>		<code>fraction:</code>	934, 940, 953
.....	141, 142, 173	<code>__siunitx_unit_format_finalise_-</code>	
<code>__siunitx_table_split_tidy:Nn .</code>	173	<code>fractional:</code>	916
<code>__siunitx_table_split_token:NNNN</code>		<code>__siunitx_unit_format_finalise_-</code>	
.....	135	<code>power:</code>	916
<code>\l__siunitx_table_text_bool</code>		<code>__siunitx_unit_format_finalise_-</code>	
.....	6, 9, 16, 238	<code>symbol:</code>	933, 943, 962
<code>\l__siunitx_table_tmp_box</code> 3, 373,		<code>__siunitx_unit_format_font: . . .</code>	
376, 412, 415, 417, 419, 529, 537,		686, 799, 811, 821, 852, 904
566, 568, 588, 592, 604, 617, 625,		<code>__siunitx_unit_format_literal:n</code>	
639, 658, 673, 677, 698, 699, 700,		194, 197, 211
709, 710, 711, 731, 736, 741, 748, 753		<code>__siunitx_unit_format_literal_-</code>	
<code>\l__siunitx_table_tmp_dim</code>		<code>auxi:w</code>	211
. . 130, 3, 648, 659, 699, 710, 740, 752		<code>__siunitx_unit_format_literal_-</code>	
<code>\l__siunitx_table_tmp_tl</code> 3,		<code>auxii:n</code>	251, 255
404, 407, 498, 499, 500, 501, 503,		<code>__siunitx_unit_format_literal_-</code>	
542, 555, 557, 558, 562, 565, 762, 763		<code>auxii:w</code>	211
<code>__siunitx_tmp:w</code>		<code>__siunitx_unit_format_literal_-</code>	
.....	229, 231, 584, 585, 589, 590	<code>auxiii:w</code>	211
<code>\l__siunitx_tmp_tl</code> 43, 113,		<code>__siunitx_unit_format_literal_-</code>	
114, 123, 124, 186, 597, 598, 608, 609		<code>auxiv:w</code>	211
<code>\l__siunitx_unit_autofrac_bool . .</code>		<code>__siunitx_unit_format_literal_-</code>	
497, 504, 511, 518, 525, 532, 551, 929		<code>auxv:w</code>	211
<code>\l__siunitx_unit_bracket_bool . . .</code>		<code>__siunitx_unit_format_literal_-</code>	
.....	545, 583, 676, 746, 853, 874	<code>subscript:</code>	211
<code>\l__siunitx_unit_bracket_close_-</code>		<code>__siunitx_unit_format_literal_-</code>	
<code>tl</code>	546, 680, 805	<code>superscript:</code>	211
<code>\l__siunitx_unit_bracket_open_tl</code>		<code>__siunitx_unit_format_literal_-</code>	
.....	546, 678, 802	<code>tilde:</code>	211

_siunitx_unit_format_mass_to-	_siunitx_unit_if_symbolic:nTF .
kilogram: 579, 650 11, 79, 185
_siunitx_unit_format_multiply:	_siunitx_unit_literal_power:nn
. 575, 634 43, 117, 209
_siunitx_unit_format_output: . .	_siunitx_unit_literal_special:nN
. 590, 850 111, 210
_siunitx_unit_format_output_-	\l_siunitx_unit_mass_kilogram_-
aux: 850	bool 122, 578, 759
_siunitx_unit_format_output_-	\c_siunitx_unit_math_subscript_-
aux:nn 850	tl 7, 221, 245, 274, 275,
_siunitx_unit_format_output_-	278, 279, 283, 284, 286, 287, 310, 824
denominator: 850	\l_siunitx_unit_multiple_fp 136,
_siunitx_unit_format_parsed: . .	143, 150, 157, 165, 173, 560, 574, 641
. 189, 565	_siunitx_unit_non_latin:n
_siunitx_unit_format_parsed_-	988, 1024, 1040, 1051, 1074, 1075, 1076
aux:n 565	_siunitx_unit_non_latin:nnnn . 988
_siunitx_unit_format_power: . . 684	\l_siunitx_unit_numerator_bool .
_siunitx_unit_format_power_- 167, 555, 586, 698, 857
aux:wTF 684	\l_siunitx_unit_options_bool . . .
_siunitx_unit_format_power_- 88, 91, 102
negative: 684	_siunitx_unit_parse:n 187, 334
_siunitx_unit_format_power_-	_siunitx_unit_parse_add:nnnn . .
negative_aux:w 684 347,
_siunitx_unit_format_power_-	364, 378, 396, 406, 415, 419, 424, 438
positive: 684	\l_siunitx_unit_parse_bool 183, 484
_siunitx_unit_format_power_-	_siunitx_unit_parse_finalise: .
superscript:w 715, 718 345, 474
_siunitx_unit_format_power_-	_siunitx_unit_parse_finalise:n
superscript: 684 344, 443
_siunitx_unit_format_prefix: . 748	_siunitx_unit_parse_per: . 106, 429
_siunitx_unit_format_prefix_-	_siunitx_unit_parse_power:nnN .
exp: 748 44, 47, 118, 121, 361
_siunitx_unit_format_prefix_-	_siunitx_unit_parse_prefix:Nn .
gram: 748 53, 361
_siunitx_unit_format_prefix_-	_siunitx_unit_parse_qualifier:nn
symbol: 748 68, 115, 361
_siunitx_unit_format_qualifier:	_siunitx_unit_parse_special:n .
. 788 109, 112, 361
_siunitx_unit_format_qualifier_-	_siunitx_unit_parse_unit:Nn 81, 410
bracket: 788	\l_siunitx_unit_parsed_prop . . .
_siunitx_unit_format_qualifier_- 153, 188,
combine: 788	331, 336, 350, 357, 375, 394, 446,
_siunitx_unit_format_qualifier_-	448, 452, 460, 464, 469, 480, 598,
phrase: 788	604, 608, 613, 623, 627, 636, 638,
_siunitx_unit_format_qualifier_-	643, 645, 654, 656, 663, 665, 763, 768
subscript: 788	\l_siunitx_unit_parsing_bool . . .
_siunitx_unit_format_special: 831 9, 34, 216, 323, 337, 659, 781
_siunitx_unit_format_unit: . . 845	\l_siunitx_unit_part_tl
\l_siunitx_unit_formatted_tl 454, 456, 457, 458,
. . . . 160, 131, 181, 201, 232, 240,	465, 561, 599, 688, 701, 704, 706,
241, 290, 293, 295, 569, 881, 882,	716, 757, 772, 778, 779, 787, 795,
927, 928, 942, 944, 948, 949, 950,	800, 804, 812, 816, 822, 827, 835, 848
955, 958, 964, 966, 973, 976, 980, 982	\l_siunitx_unit_per_bool
_siunitx_unit_if_symbolic:n . . . 11 156, 331, 338, 422, 433

\l__siunitx_unit_per_symbol_bool	498, 505, 512, 519, 526, 533, <u>551</u> , 880, 886, 932
\l__siunitx_unit_per_symbol_tl ..	<u>484</u> , 967
\l__siunitx_unit_position_int ...	<u>160</u> , <u>331</u> , 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, <u>551</u> , 699, 920
\l__siunitx_unit_prefix_exp_bool	<u>134</u> , 141, 148, 155, 163, 171, <u>558</u> , 577, 750
\l__siunitx_unit_prefix_fp	182, 203, <u>559</u> , 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	<u>122</u> , 252, 865, 876, 983
\l__siunitx_unit_qualifier_mode_- tl	<u>484</u> , <u>556</u> , 793
\l__siunitx_unit_qualifier_- phrase_tl	<u>484</u> , 814
\l__siunitx_unit_separator_tl ..	<u>211</u>
__siunitx_unit_set_symbolic:Nnn	23, 42, 45, 51, 66, 76, 104
__siunitx_unit_set_symbolic:Nnnn	<u>23</u>
__siunitx_unit_set_symbolic:Npnn	<u>23</u> , 107, 110, 113, 116, 119
\l__siunitx_unit_sticky_per_bool	<u>156</u> , <u>327</u> , 431
\l__siunitx_unit_test_bool	<u>10</u> , 14, 32, 339
\l__siunitx_unit_tmp_fp ..	<u>4</u> , 137, 151, 158, 174, 606, 621, 640, 642, 646
\l__siunitx_unit_tmp_int ..	<u>4</u> , 363, 365
\l__siunitx_unit_tmp_tl	<u>4</u> , 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	<u>564</u> , 567, 581, 652
\l__siunitx_unit_two_part_bool ..	500, 507, 514, 521, 528, 535, <u>551</u> , 869
skip commands:	
\skip_horizontal:N	250
\skip_horizontal:n	186, 225, 571
\c_zero_skip	187
\sp	<u>164</u>
\space	48, 50, 54, 56, 60, 62, 499, 501, 505, 507, 510, 516, 518, 522, 524, 527, 533, 535, 542, 544
space-before-unit	182
\SplitArgument	100
\SplitList	128, 137, 146, 156, 616
\square	<u>140</u> , <u>1078</u>
\squared	<u>141</u> , <u>1078</u>
\steradian	<u>140</u> , <u>1050</u>
sticky-per	<u>143</u>
str commands:	
\str_case_e:nnTF	176
\str_if_eq:nnTF	39, 80, 125, 157, 180, 236, 242, 274, 333, 385, 609, 706, 719, 751, 768, 813, 823, 853, 955, 1126, 1200, 1246, 1385, 1398, 1411, 1455, 1480, 1614, 1671, 1714, 1760, 1916
\str_if_eq_p:nn	319, 444, 544, 676, 678, 700, 702, 1376, 1378, 1582, 1588, 1659, 1661, 1840, 1843
\str_range:nnn	349, 351
sys commands:	
\sys_if_engine luatex_p:	11, 106, 121, 160, 636, 693, 709, 989
\sys_if_engine xetex:TF	310
\sys_if_engine xetex_p:	12, 107, 122, 161, 637, 694, 710, 990
T	
table-align-comparator	<u>113</u>
table-align-exponent	<u>113</u>
table-align-text-after	<u>113</u>
table-align-text-before	<u>113</u>
table-align-uncertainty	<u>113</u>
table-alignment	<u>114</u>
table-alignment-mode	<u>114</u>
table-auto-round	<u>114</u>
table-column-width	<u>114</u>
table-fixed-width	<u>114</u>
table-format	<u>114</u>
table-number-alignment	<u>114</u>
table-text-alignment	<u>114</u>
\tablenum	<u>198</u>
\tabularnewline	55, 57, 76, 78
\tebi	<u>181</u> , <u>2</u>
\tera	<u>139</u> , 13, 57, <u>1028</u>

<code>\tesla</code>	140, 1050	243, 255, 301, 313, 323, 349, 370,	
<code>\TeV</code>	176, 42	478, 502, 503, 523, 568, 569, 584, 688	
T _E X and L ^A T _E X 2 _ε commands:			
<code>\@ifl@t@r</code>	12, 42	<code>\tl_clear_new:N</code>	83
<code>\@ifpackagelater</code>	1	<code>\tl_const:Nn</code>	7, 91, 121, 123
<code>\@ifpackageloaded</code> 30, 42, 44, 63, 67,		<code>\tl_count:N</code>	599, 611
80, 83, 152, 215, 242, 246, 314, 320, 662		<code>\tl_count:n</code>	111, 165,
<code>\@ifundefined</code>	9	175, 182, 339, 599, 607, 680, 703,	
<code>\@maybe@unskip</code>	73	743, 980, 981, 1211, 1347, 1350,	
<code>\@temptokena</code>	234, 236	1357, 1367, 1379, 1426, 1435, 1440,	
<code>\f@family</code>	176	1457, 1685, 1696, 1766, 1824, 1832	
<code>\f@series</code>	129	<code>\tl_head:n</code>	97, 214,
<code>\FB@fg</code>	333	269, 322, 1091, 1109, 1262, 1411, 1752	
<code>\m@th</code>	365, 369	<code>\tl_head:w</code>	913, 937
<code>\math@version</code>	157	<code>\tl_if_blank:nTF</code>	
<code>\NC@do</code>	229, 230 3, 17, 44, 103, 106, 142,	
<code>\NC@find</code>	239	142, 158, 184, 192, 211, 300, 300,	
<code>\NC@list</code>	7, 230, 231	306, 308, 320, 322, 348, 350, 352,	
<code>\newcommand</code>	184	355, 357, 363, 385, 403, 482, 524,	
<code>\protected@edef</code>		595, 685, 707, 719, 720, 791, 803,	
..... 150, 15, 35, 76, 119, 128, 230		809, 831, 847, 907, 921, 931, 944,	
<code>\sf@size</code>	382	974, 1046, 1156, 1288, 1329, 1432,	
<code>\tab@setcr</code>	74	1602, 1608, 1628, 1669, 1749, 1856	
<code>\z@</code>	382	<code>\tl_if_blank_p:n</code>	4, 139,
tex commands:			143, 218, 219, 1407, 1584, 1660, 1823
<code>\tex_cr:D</code>	32	<code>\tl_if_empty:N</code>	93,
<code>\tex_hfil:D</code>	244, 247	105, 109, 113, 114, 125, 127, 130,	
<code>\tex_hss:D</code>	267, 269	135, 152, 156, 160, 165, 175, 233,	
<code>\tex_kern:D</code>	187	256, 261, 264, 294, 304, 306, 312,	
<code>\text</code>	89, 99, 49, 55, 61,	352, 470, 522, 577, 687, 898, 918,	
161, 256, 500, 506, 517, 523, 534, 543		927, 973, 1382, 1466, 1559, 1616, 1876	
<code>text-family-to-math</code>	90	<code>\tl_if_empty:nTF</code>	84, 693, 1564
<code>text-font-command</code>	91	<code>\tl_if_empty_p:N</code> 249, 264, 432, 873, 881	
<code>text-series-to-math</code>	91	<code>\tl_if_eq:NNTF</code>	137
<code>\textcolor</code>		<code>\tl_if_eq:nnTF</code>	413
.. 89–91, 135, 141, 95, 111, 112, 1641		<code>\tl_if_exist:N</code>	95
<code>\textminus</code>	89, 295	<code>\tl_if_head_eq_charcode:nNTF</code> ..	732
<code>\textmu</code>	124, 696	<code>\tl_if_head_eq_meaning:nNTF</code> ...	272
<code>\textohm</code>	109, 712	<code>\tl_if_in:NnTF</code> ... 5, 56, 149, 250,	
<code>\textpm</code>	89, 291	309, 340, 384, 390, 402, 405, 412,	
<code>\textsubscript</code>	89, 331, 362	417, 486, 512, 530, 541, 555, 561, 568	
<code>\textsuperscript</code>	89, 336	<code>\tl_if_novalue:nTF</code>	212, 215
<code>\texttimes</code>	143, 145	<code>\tl_if_single_token:nTF</code>	93
<code>\the</code>	231, 236	<code>\tl_map_function:nN</code>	101, 118
<code>\THz</code>	175, 8	<code>\tl_map_inline:Nn</code> .. 260, 270, 380, 421	
<code>tight-spacing</code>	41	<code>\tl_map_inline:nn</code>	54
<code>\times</code> 14, 20, 26, 32, 136, 143, 145, 537, 1942		<code>\tl_new:N</code>	3, 3, 3, 4,
tl commands:			4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 9, 9,
<code>\c_empty_tl</code>	55, 56, 1724	9, 10, 12, 13, 14, 23, 28, 32, 33, 38,	
<code>\c_space_tl</code> 273, 327, 352, 357, 396, 404		39, 43, 68, 69, 70, 71, 72, 73, 74, 75,	
<code>\tl_clear:N</code>		87, 88, 89, 90, 102, 103, 104, 131,	
. 27, 101, 112, 118, 137, 138, 139,		235, 292, 293, 294, 295, 296, 297,	
149, 152, 154, 177, 178, 181, 232,		320, 360, 361, 404, 406, 458, 459,	
		546, 547, 556, 561, 562, 563, 663,	
		664, 665, 1482, 1483, 1487, 1548, 1868	

<code>\tl_put_right:Nn</code>	3, 5, 5, 6, 88, 104, 115, 215, 218, 218, 220, 247, 272, 307
<code>\tl_replace_all:Nnn</code>	798, 1183, 1184, 1190
<code>\tl_set:Nn</code>	60, 150, 169, 7, 8, 15, 16, 21, 24, 26, 34, 51, 53, 53, 66, 75, 82, 85, 111, 114, 122, 127, 127, 128, 131, 132, 134, 136, 136, 137, 146, 147, 161, 163, 181, 200, 211, 212, 213, 217, 233, 235, 240, 246, 252, 253, 263, 264, 268, 280, 286, 290, 297, 299, 305, 309, 310, 314, 319, 320, 321, 332, 349, 354, 355, 369, 392, 402, 404, 409, 417, 433, 445, 450, 456, 464, 471, 478, 500, 514, 518, 519, 521, 548, 549, 557, 558, 588, 596, 605, 616, 619, 620, 639, 660, 662, 667, 673, 701, 712, 721, 722, 729, 740, 761, 766, 770, 778, 782, 784, 800, 812, 822, 833, 882, 895, 908, 928, 942, 944, 955, 960, 964, 980, 997, 1007, 1017, 1448, 1484, 1485
<code>\tl_set_eq:NN</code>	18, 45, 69, 125, 159, 252, 276, 290, 366, 372, 411, 421, 464, 468, 541, 640, 656, 787, 975, 1010, 1542
<code>\tl_tail:n</code>	98, 914, 938
<code>\tl_use:N</code>	100, 144, 147, 221, 237, 271, 282, 297, 299, 310, 321
<code>\l_tmpa_tl</code>	135, 136
token commands:	
<code>\c_math_toggle_token</code> ...	379, 388, 391, 396, 420, 430, 434, 439, 448, 449
<code>\token_if_eq_charcode:p:NN</code>	516
<code>\token_if_eq_meaning:NNTF</code> ..	98, 1478
<code>\token_to_str:N</code>	30, 83, 85, 85, 95, 98, 105, 180, 219, 221, 248, 325, 338, 354, 387, 401, 1093, 1096
<code>\tonne</code>	140, 1061
<code>\tothe</code>	141, 116
<code>\TrimSpaces</code>	91, 117, 128, 146, 156, 164, 616, 625
<code>\ttdefault</code>	180
U	
<code>\uA</code>	175, 2
<code>\uF</code>	177, 70
<code>\uG</code>	175, 35
<code>\uH</code>	75
<code>\uJ</code>	176, 42
<code>\uL</code>	176, 27
<code>\uI</code>	176, 184, 27, 48
<code>\uM</code>	175, 60
<code>\uMol</code>	176, 14
<code>uncertainty-mode</code>	41
<code>uncertainty-separator</code>	41
<code>\unit</code>	198, 100, 279, 313
<code>unit-color</code>	91
<code>unit-font-command</code>	143
<code>unit-mode</code>	91
<code>unit-optional-argument</code>	182
<code>\unskip</code>	54, 75
<code>\upOmega</code>	99, 100, 702, 703
<code>\upshape</code>	90
<code>\us</code>	175, 89
use commands:	
<code>\use:N</code>	26, 65, 97, 164, 168, 239, 322, 366, 370, 442, 467, 575, 600, 759, 790, 854, 859, 883, 946, 986, 1202, 1687, 1693, 1752, 1772
<code>\use:n</code>	70, 94, 117, 156, 162, 180, 181, 196, 206, 206, 258, 267, 275, 284, 383, 838, 1673, 1716, 1849
<code>\use_i:nn</code>	77, 859, 946
<code>\use_i:nnnn</code>	148
<code>\use_i_delimit_by_q-recursion-stop:nw</code>	1230, 1275, 1918, 1920
<code>\use_i_delimit_by_q-stop:nw</code>	99
<code>\use_ii:nn</code>	1415
<code>\use_iv:nnnn</code>	140, 144
<code>\use_none:n</code>	482, 733, 1264, 1734
<code>\use_none:nn</code>	1738
<code>\use_none:nnnn</code>	88
<code>use-xspace</code>	182
<code>\uV</code>	176, 21
<code>\uW</code>	176, 42
V	
<code>\V</code>	176, 21
<code>\volt</code>	140, 21, 22, 23, 24, 25, 26, 1050
W	
<code>\W</code>	176, 42
<code>\watt</code> ..	140, 42, 43, 44, 45, 46, 47, 58, 1050
<code>\weber</code>	140, 1050
X	
<code>\xspace</code>	88
Y	
<code>\yobi</code>	181, 2
<code>\yocto</code>	139, 1018
<code>\yotta</code>	139, 1028

	Z		
\zebi	181, <u>2</u>	\zepto	139, <u>1018</u>
		\zetta	139, <u>1028</u>