

siunitx – A comprehensive (SI) units package*

Joseph Wright[†]

Released 2021-09-06

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	31
1.3	Products	32
1.4	Ranges	33
1.5	Standard settings for module options	34

*This file describes v3.0.29, last revised 2021-09-06.

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

IV	siunitx-locale – Localisation	36
1	siunitx-locale implementation	36
1.1	Locales	36
1.2	Localisation	37
V	siunitx-number – Parsing and formatting numbers	38
1	Formatting numbers	38
1.1	Key-value options	40
2	siunitx-number implementation	43
2.1	Initial set-up	43
2.2	Main formatting routine	43
2.3	Parsing numbers	44
2.4	Processing numbers	60
2.5	Number modification	81
2.6	Outputting parsed numbers	81
2.7	Miscellaneous tools	90
2.8	Messages	91
2.9	Standard settings for module options	92
VI	siunitx-print – Printing material with font control	93
1	Printing quantities	93
1.1	Key-value options	94
2	siunitx-print implementation	96
2.1	Initial set up	96
2.2	Printing routines	96
2.3	Standard settings for module options	105
VII	siunitx-quantity – Quantities	107
1	siunitx-quantity implementation	107
1.1	Initial set-up	108
1.2	Main formatting routine	108
1.3	Standard settings for module options	112
1.4	Adjustments to units	112
VIII	siunitx-symbol – Symbol-related settings	113
1	siunitx-symbol implementation	113
1.1	Bookmark definitions	116
IX	siunitx-table – Formatting numbers in tables	117

1	Numbers in tables	117
1.1	Key-value options	117
2	siunitx-table implementation	119
2.1	Interface functions	119
2.2	Collecting tokens	119
2.3	Separating collected material	122
2.4	Printing numbers in cells: spacing	124
2.5	Printing just text	125
2.6	Number alignment: core ideas	126
2.7	Directly printing without collection	130
2.8	Printing numbers in cells: main functions	132
2.9	Standard settings for module options	139
X	siunitx-unit – Parsing and formatting units	141
1	Formatting units	141
2	Defining symbolic units	143
3	Per-unit options	144
4	Units in (PDF) strings	144
5	Pre-defined symbolic unit components	144
5.1	Key-value options	147
6	siunitx-unit implementation	149
6.1	Initial set up	149
6.2	Defining symbolic unit	150
6.3	Applying unit options	152
6.4	Non-standard symbolic units	153
6.5	Main formatting routine	154
6.6	Formatting literal units	156
6.7	(PDF) String creation	159
6.8	Parsing symbolic units	159
6.9	Formatting parsed units	164
6.10	Non-Latin character support	177
6.11	Pre-defined unit components	177
6.12	Messages	179
6.13	Standard settings for module options	180
XI	siunitx-abbreviations – Abbreviations	182
1	siunitx-abbreviation implementation	184
XII	siunitx-binary – Binary units	188

1	siunitx-binary implementation	188
XIII	siunitx-command – Units as document command	189
1	Creating units as document commands	189
1.1	Key–value options	189
2	siunitx-command implementation	190
2.1	Options	190
2.2	Creation of unit document commands	191
2.3	Standard settings for module options	192
XIV	siunitx-emulation – Emulation	193
1	siunitx-emulation implementation	193
1.1	Load-time option	194
1.2	Angle options	194
1.3	Combination functions options	194
1.4	Command options	195
1.5	Print options	195
1.6	Symbol options	197
1.7	Number options	197
1.7.1	Table options	201
1.8	Unit options	203
1.9	Quantity units	204
1.10	Preamble commands	205
1.11	Document commands	206
1.12	Symbol commands	207
1.13	Unit commands	208
1.14	Communication with pgf	210
	Index	212

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{2020-01-09}
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} {2021-09-06} {3.0.29}
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 {#3} {#2} 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 \@ifpackageloaded { units }
118 {
119   \msg_new:nnn { siunitx } { units-pkg }
120   {
121     Detected~the~"units"~package: \\\
122     Omitting~definition~of~\token_to_str:N \unit.
123   }
124   \msg_warning:nn { siunitx } { units-pkg }
125   \use_none:nnnn
126 }
127 { }
128 \NewDocumentCommand \unit { 0 { } } > { \TrimSpaces } m }
129 {
130   \mode_leave_vertical:
131   \group_begin:
132     \siunitx_unit_options_apply:n {#2}
133     \keys_set:nn { siunitx } {#1}
134     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
135     \siunitx_print_unit:V \l__siunitx_tmp_tl
136   \group_end:
137 }

```

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


```

\qtylist Interfaces for compound values.
\numlist 138 \NewDocumentCommand \qtylist
\qtyproduct 139 { 0 { } > { \SplitList { ; } } m > { \TrimSpaces } m }
\numproduct 140 {
\qtyrange 141 \mode_leave_vertical:
142 \group_begin:
143 \siunitx_unit_options_apply:n {#3}
144 \keys_set:nn { siunitx } {#1}
145 \siunitx_quantity_list:nn {#2} {#3}
146 \group_end:
147 }
148 \NewDocumentCommand \numlist { 0 { } > { \SplitList { ; } } m }
149 {
150 \mode_leave_vertical:
151 \group_begin:
152 \keys_set:nn { siunitx } {#1}
153 \siunitx_number_list:nn {#2}
154 \group_end:
155 }
156 \NewDocumentCommand \qtyproduct
157 { 0 { } > { \SplitList { x } } m > { \TrimSpaces } m }
158 {
159 \mode_leave_vertical:
160 \group_begin:
161 \siunitx_unit_options_apply:n {#3}
162 \keys_set:nn { siunitx } {#1}
163 \siunitx_quantity_product:nn {#2} {#3}
164 \group_end:
165 }
166 \NewDocumentCommand \numproduct
167 { 0 { } > { \SplitList { x } } m > { \TrimSpaces } m }
168 {
169 \mode_leave_vertical:
170 \group_begin:
171 \keys_set:nn { siunitx } {#1}
172 \siunitx_number_product:n {#2}
173 \group_end:
174 }
175 \NewDocumentCommand \qtyrange { 0 { } m m > { \TrimSpaces } m }
176 {
177 \mode_leave_vertical:
178 \group_begin:
179 \siunitx_unit_options_apply:n {#4}
180 \keys_set:nn { siunitx } {#1}
181 \siunitx_quantity_range:nnn {#2} {#3} {#4}
182 \group_end:
183 }
184 \NewDocumentCommand \numrange { 0 { } m m }
185 {
186 \mode_leave_vertical:
187 \group_begin:
188 \keys_set:nn { siunitx } {#1}
189 \siunitx_number_range:nn {#2} {#3}
190 \group_end:

```

191 }

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

\complexnum Interfaces for complex numbers.

```

192 \NewDocumentCommand \complexnum { 0 { } m }
193 {
194   \mode_leave_vertical:
195   \group_begin:
196     \keys_set:nn { siunitx } {#1}
197     \siunitx_complex_number:n {#2} \l__siunitx_tmp_tl
198   \group_end:
199 }
200 \NewDocumentCommand \complexqty { 0 { } m m }
201 {
202   \mode_leave_vertical:
203   \group_begin:
204     \siunitx_unit_options_apply:n {#3}
205     \keys_set:nn { siunitx } {#1}
206     \siunitx_complex_quantity:nn {#2} {#3}
207   \group_end:
208 }

```

(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.

```

209 \NewDocumentCommand \tablenum { 0 { } m }
210 {
211   \mode_leave_vertical:
212   \group_begin:
213     \keys_set:nn { siunitx } {#1}
214     \siunitx_cell_begin:w
215     \ignorespaces #2
216     \siunitx_cell_end:
217   \group_end:
218 }

```

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

\sisetup A very thin wrapper.

```

219 \NewDocumentCommand \sisetup { m }
220 { \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.

```

221 \cs_new_protected:Npn \__siunitx_angle:nnn #1#2#3
222 {
223   \tl_if_novalue:nTF {#2}
224     { \siunitx_angle:n {#1} }
225     {

```

```

226         \tl_if_novalue:nTF {#3}
227         { \siunitx_angle:nnn {#1} {#2} { } }
228         { \siunitx_angle:nnn {#1} {#2} {#3} }
229     }
230 }

```

(End definition for `_siunitx_angle:nnn`.)

1.9 Table column

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

```

231 \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.

```

232 \cs_new_protected:Npn \_siunitx_declare_column:Nnn #1#2#3
233 {
234     \cs_if_exist:cT { NC@find@ #1 }
235     {
236         \cs_undefine:c { NC@find@ #1 }
237         \cs_set_protected:Npn \_siunitx_tmp:w ##1 \NC@do #1 ##2 \q_stop
238         { \NC@list {##1##2} }
239         \exp_after:wN \_siunitx_tmp:w \the \NC@list \q_stop
240         \msg_warning:nnn { siunitx } { column-overwritten } {#1}
241     }
242     \newcolumntype {#1} { }
243     \cs_set_protected:Npn \_siunitx_tmp:w \NC@do ##1##2 \NC@do #1
244     { \NC@list { \NC@do ##1 \NC@do #1 ##2 } }
245     \exp_after:wN \_siunitx_tmp:w \the \NC@list
246     \exp_args:NNc \renewcommand * { NC@rewrite@ #1 } [ 1 ] [ ]
247     {
248         \@temptokena \expandafter
249         {
250             \the \@temptokena
251             > {#2} c < {#3}
252         }
253         \NC@find
254     }
255 }
256 \msg_new:nnn { siunitx } { column-overwritten }
257 { Tabular~column~type~"#1"~overwritten~with~siunitx~definition. }

```

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

```

258 \AtBeginDocument
259 {
260     \ifpackageloaded { mdwtab }
261     {
262         \cs_set_protected:Npn \_siunitx_declare_column:Nnn #1#2#3
263         {
264             \cs_if_exist:cT { NC@find@ #1 }
265             {
266                 \cs_undefine:c { NC@find@ #1 }

```

```

267         \msg_warning:nnn { siunitx } { column-overwritten } {#1}
268     }
269     \newcolumnntype {#1} [ 1 ] [ ]
270     { > {#2} c < {#3} }
271 }
272 }
273 { }
274 \tl_map_inline:Nn \l__siunitx_column_type_tl
275 {
276     \__siunitx_declare_column:Nnn #1
277     {
278         \keys_set:nn { siunitx } {##1}
279         \siunitx_cell_begin:w
280     }
281     { \siunitx_cell_end: }
282 }
283 }

```

(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.

```

284 \cs_new_protected:Npn \__siunitx_bookmark_cmd:Nnn #1#2#3
285 {
286     \exp_args:Nc \DeclareExpandableDocumentCommand
287     { \cs_to_str:N #1 \c_space_tl ( pdfstring ~ context ) }
288     {#2} {#3}
289 }
290 \__siunitx_bookmark_cmd:Nnn \qty { o m m } { #2 ~ #3 }
291 \__siunitx_bookmark_cmd:Nnn \ang { m } { \__siunitx_angle:n {#1} }
292 \__siunitx_bookmark_cmd:Nnn \num { o m } { #2 }
293 \__siunitx_bookmark_cmd:Nnn \unit { o m } { #2 }
294 \__siunitx_bookmark_cmd:Nnn \numlist { o m }
295 {
296     \__siunitx_list_use:nnVVV {#2} { }
297     \l_siunitx_list_separator_pair_tl
298     \l_siunitx_list_separator_tl
299     \l_siunitx_list_separator_final_tl
300 }
301 \__siunitx_bookmark_cmd:Nnn \qtylist { o m m }
302 {
303     \__siunitx_list_use:nnVVV {#2} {#3}
304     \l_siunitx_list_separator_pair_tl
305     \l_siunitx_list_separator_tl
306     \l_siunitx_list_separator_final_tl
307 }
308 \__siunitx_bookmark_cmd:Nnn \numproduct { o m } { }
309 \__siunitx_bookmark_cmd:Nnn \qtyproduct { o m m } { }

```

```

310 \__siunitx_bookmark_cmd:Nnn \numrange { o m m }
311 { #2 \tl_use:N \l_siunitx_range_phrase_tl #3 }
312 \__siunitx_bookmark_cmd:Nnn \qtyrange { o m m m }
313 { #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.

314 \__siunitx_bookmark_cmd:Nnn \si { o m } { #2 }
315 \__siunitx_bookmark_cmd:Nnn \SI { o m 0 { } m } { #3 #2 ~ #4 }
316 \__siunitx_bookmark_cmd:Nnn \SIlist { o m m }
317 {
318   \__siunitx_list_use:nnVVV {#2} {#3}
319   \l_siunitx_list_separator_pair_tl
320   \l_siunitx_list_separator_tl
321   \l_siunitx_list_separator_final_tl
322 }
323 \__siunitx_bookmark_cmd:Nnn \SIrange { o m m m }
324 { #2 ~ #4 \tl_use:N \l_siunitx_range_phrase_tl #3 ~ #4 }

```

\c__siunitx_bookmark_seq Commands usable in bookmarks

```

325 \seq_const_from_clist:Nn \c__siunitx_bookmark_seq
326 {
327   \ang , \qty , \num , \unit ,
328   \numlist , \qtylist ,
329   \numrange , \qtyrange ,
330   \si , \SI , \SIlist , \SIrange
331 }

(End definition for \c__siunitx_bookmark_seq.)

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

332 \AtBeginDocument
333 {
334   \ifpackageloaded { hyperref }
335   {
336     \pdfstringdefDisableCommands
337     {
338       \seq_map_inline:Nn \c__siunitx_bookmark_seq
339       {
340         \cs_set_eq:Nc #1
341         { \cs_to_str:N #1 \c_space_tl ( pdfstring ~ context ) }
342       }
343     }
344     \pdfstringdefDisableCommands
345     {
346       \siunitx_unit_pdfstring_context:
347       \cs_if_exist:NT \FB@fg { \def \fg { \FB@fg } }
348       \edef \H
349       {
350         \exp_not:c { PU-cmd }
351         \exp_not:N \H
352         \exp_not:c { PU \token_to_str:N \H }
353       }
354     }
355   }

```

```

356     { }
357 }

```

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

```

358 \cs_new:Npn \_siunitx_angle:n #1
359 { \_siunitx_angle:w #1 ; ; \q_stop }
360 \cs_new:Npn \_siunitx_angle:w #1 ; #2 ; #3 ; #4 \q_stop
361 {
362   \tl_if_blank:nF {#1}
363   { #1 \degree }
364   \tl_if_blank:nF {#2}
365   {
366     \tl_if_blank:nF {#1} { \c_space_tl }
367     #2 \arcminute
368   }
369   \tl_if_blank:nF {#3}
370   {
371     \tl_if_blank:nF {#1#2} { \c_space_tl }
372     #3 \arcsecond
373   }
374 }

```

(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
\_siunitx_list_use_auxi:w
\siunitx_list_use_auxii:nnw
\siunitx_list_use_auxiii:nnw
\_siunitx_list_count:n
\_siunitx_list_count:w
375 \cs_new:Npn \_siunitx_list_use:nnnnn #1#2#3#4#5
376 {
377   \tl_if_blank:nTF {#2}
378   { \_siunitx_list_use_aux:nnnnn {#1} { } }
379   { \_siunitx_list_use_aux:nnnnn {#1} { ~ #2 } }
380   {#3} {#4} {#5}
381 }
382 \cs_generate_variant:Nn \_siunitx_list_use:nnnnn { nnVVV }
383 \cs_new:Npn \_siunitx_list_use_aux:nnnnn #1#2#3#4#5
384 {
385   \int_case:nnF { \_siunitx_list_count:n {#1} }
386   {
387     { 0 } { }
388     { 1 } { \_siunitx_list_use_auxi:nw {#2} #1 ; ; { } }
389     { 2 } { \_siunitx_list_use_auxi:nw {#2} #1 ; {#3} }
390   }
391   {
392     \_siunitx_list_use_auxii:nnw {#2} { } #1 ;
393     \q_mark ; { \_siunitx_list_use_auxii:nnw {#2} {#4} }
394     \q_mark ; { \_siunitx_list_use_auxiii:nnw {#2} {#5} }
395     \q_stop { }
396   }
397 }
398 \cs_new:Npn \_siunitx_list_use_auxi:nw #1#2 ; #3 ; #4
399 { #2 #1 #4 #3 \tl_if_blank:nF {#3} {#1} }
400 \cs_new:Npn \_siunitx_list_use_auxii:nnw
401 #1#2#3 ; #4 ; #5 ; #6 \q_mark ; #7#8 \q_stop #9
402 { #7 {#4} ; {#5} ; #6 \q_mark ; {#7} #8 \q_stop { #9 #2 #3 #1 } }

```

```

403 \cs_new:Npn \__siunitx_list_use_auxiii:nnw #1#2#3 ; #4 \q_stop #5
404 { #5 #2 #3 #1 }
405 \cs_new:Npx \__siunitx_list_count:n #1
406 {
407   \exp_not:N \int_eval:n
408   {
409     0
410     \exp_not:N \__siunitx_list_count:w \c_space_tl
411     #1 \exp_not:n { ; \q_recursion_tail ; \q_recursion_stop }
412   }
413 }
414 \cs_new:Npx \__siunitx_list_count:w #1 ;
415 {
416   \exp_not:n { \exp_args:Nf \quark_if_recursion_tail_stop:n } {#1}
417   \exp_not:N \tl_if_blank:nF {#1} { + 1 }
418   \exp_not:N \__siunitx_list_count:w \c_space_tl
419 }

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

420 \end{interfaces}

421 \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
\__siunitx_angle_arc_sign:nn shuffling. That means doing processing of each number manually.
\__siunitx_angle_extract_sign:nnnnnnnn
\__siunitx_angle_sign:nnnnnnnn
91 \cs_new_protected:Npn \__siunitx_angle_arc_sign:nnn #1#2#3
92 {
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     }

```

```

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

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

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

Start the DocStrip guards.

1 `<*package>`

1 siunitx-compound implementation

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

1.1 General mechanism

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

3 `<@@=siunitx_compound>`

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

Scratch space.

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

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

The first number in the list in internal format.

7 `\tl_new:N \l__siunitx_compound_first_tl`

(End definition for `\l__siunitx_compound_first_tl`.)

For storing the combined exponent, if present.

8 `\tl_new:N \l__siunitx_compound_exp_tl`

(End definition for `\l__siunitx_compound_exp_tl`.)

Data on the end-of-list.

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

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

Data on the length-of-list.

11 `\int_new:N \l__siunitx_compound_count_int`

(End definition for `\l__siunitx_compound_count_int`.)

`\l__siunitx_compound_unit_bool`

12 `\bool_new:N \l__siunitx_compound_unit_bool`
13 `\tl_new:N \l__siunitx_compound_unit_tl`

(End definition for \l__siunitx_compound_unit_bool and \l__siunitx_compound_unit_tl.)

Purely internal for the present.

```
\l__siunitx_compound_bracket_close_tl
\l__siunitx_compound_bracket_open_tl
14 \tl_new:N \l__siunitx_compound_bracket_close_tl
15 \tl_new:N \l__siunitx_compound_bracket_open_tl
16 \tl_set:Nn \l__siunitx_compound_bracket_open_tl { ( }
17 \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
18 \bool_new:N \l__siunitx_compound_exp_bracket_bool
19 \bool_new:N \l__siunitx_compound_exp_combine_bool
20 \bool_new:N \l__siunitx_compound_separator_text_bool
21 \bool_new:N \l__siunitx_compound_unit_bracket_bool
22 \bool_new:N \l__siunitx_compound_unit_power_bool
23 \bool_new:N \l__siunitx_compound_unit_repeat_bool
24 \keys_define:nn { siunitx }
25 {
26   compound-exponents .choice: ,
27   compound-exponents / combine .code:n =
28   {
29     \bool_set_false:N \l__siunitx_compound_exp_bracket_bool
30     \bool_set_true:N \l__siunitx_compound_exp_combine_bool
31   } ,
32   compound-exponents / combine-bracket .code:n =
33   {
34     \bool_set_true:N \l__siunitx_compound_exp_bracket_bool
35     \bool_set_true:N \l__siunitx_compound_exp_combine_bool
36   } ,
37   compound-exponents / individual .code:n =
38   {
39     \bool_set_false:N \l__siunitx_compound_exp_bracket_bool
40     \bool_set_false:N \l__siunitx_compound_exp_combine_bool
41   } ,
42   compound-final-separator .tl_set:N =
43   \l__siunitx_compound_separator_final_tl ,
44   compound-pair-separator .tl_set:N =
45   \l__siunitx_compound_separator_pair_tl ,
46   compound-separator .tl_set:N =
47   \l__siunitx_compound_separator_tl ,
48   compound-separator-mode .choice: ,
49   compound-separator-mode / number .code:n =
50   { \bool_set_false:N \l__siunitx_compound_separator_text_bool } ,
51   compound-separator-mode / text .code:n =
52   { \bool_set_true:N \l__siunitx_compound_separator_text_bool } ,
53   compound-units .choice: ,
54   compound-units / bracket .code:n =
55   {
56     \bool_set_true:N \l__siunitx_compound_unit_bracket_bool
57     \bool_set_false:N \l__siunitx_compound_unit_power_bool
58     \bool_set_false:N \l__siunitx_compound_unit_repeat_bool
59   } ,
```

```

60 compound-units / bracket-power .code:n =
61 {
62     \bool_set_true:N \l__siunitx_compound_unit_bracket_bool
63     \bool_set_true:N \l__siunitx_compound_unit_power_bool
64     \bool_set_false:N \l__siunitx_compound_unit_repeat_bool
65 } ,
66 compound-units / power .code:n =
67 {
68     \bool_set_false:N \l__siunitx_compound_unit_bracket_bool
69     \bool_set_true:N \l__siunitx_compound_unit_power_bool
70     \bool_set_false:N \l__siunitx_compound_unit_repeat_bool
71 } ,
72 compound-units / repeat .code:n =
73 {
74     \bool_set_false:N \l__siunitx_compound_unit_bracket_bool
75     \bool_set_false:N \l__siunitx_compound_unit_power_bool
76     \bool_set_true:N \l__siunitx_compound_unit_repeat_bool
77 } ,
78 compound-units / single .code:n =
79 {
80     \bool_set_false:N \l__siunitx_compound_unit_bracket_bool
81     \bool_set_false:N \l__siunitx_compound_unit_power_bool
82     \bool_set_false:N \l__siunitx_compound_unit_repeat_bool
83 }
84 }

```

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

```

\siunitx_compound_number:n
\__siunitx_compound_format:n
    \__siunitx_compound_format:nn
    \__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.

```

85 \cs_new_protected:Npn \siunitx_compound_number:n #1
86 {
87     \group_begin:
88     \bool_set_false:N \l__siunitx_compound_unit_bool
89     \__siunitx_compound_format:nn {#1} { }
90     \__siunitx_compound_print:N \siunitx_print_number:x
91     \group_end:
92 }
93 \cs_new_protected:Npn \__siunitx_compound_format:nn #1#2
94 {
95     \seq_clear:N \l__siunitx_compound_tmp_seq
96     \bool_if:NTF \l_siunitx_number_parse_bool
97     {
98         \exp_args:Nxx \__siunitx_compound_format:nnn
99         { \tl_head:n {#1} }
100         { \tl_tail:n {#1} }
101         {#2}
102     }
103     { \tl_map_function:nN {#1} \__siunitx_compound_unparsed:n }
104 }

```

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.

```

105 \cs_new_protected:Npn \__siunitx_compound_format:nnn #1#2#3
106 {
107   \siunitx_number_parse:nN {#1} \l__siunitx_compound_tmp_tl
108   \bool_if:NTF \l__siunitx_compound_unit_bool
109     { \__siunitx_compound_format_units:nn {#2} {#3} }
110     { \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
111   \bool_lazy_and:nnTF
112     { \l__siunitx_compound_exp_combine_bool }
113     { \int_compare_p:nNn { \tl_count:n {#2} } > 0 }
114     { \__siunitx_compound_extract_exponents: }
115     {
116       \bool_if:NTF \l__siunitx_compound_unit_bool
117       {
118         \tl_set:Nx \l__siunitx_compound_tmp_tl
119           { \siunitx_number_output:NN \l__siunitx_compound_first_tl \q_nil }
120         \tl_set:Nx \l__siunitx_compound_tmp_tl
121           { \__siunitx_compound_uncert_bracket:N \l__siunitx_compound_tmp_tl }
122       }
123       {
124         \tl_set:Nx \l__siunitx_compound_tmp_tl
125           { \siunitx_number_output:N \l__siunitx_compound_first_tl }
126       }
127       \seq_put_right:NV \l__siunitx_compound_tmp_seq \l__siunitx_compound_tmp_tl
128     }
129     \tl_map_function:nN {#2} \__siunitx_compound_parsed:n
130   }

```

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.

```

\__siunitx_compound_extract_exponents:
\__siunitx_compound_extract_exponents_auxi:w
\__siunitx_compound_extract_exponents_auxii:nw
\__siunitx_compound_extract_exponents_auxiii:nnnnnnnn
131 \cs_new_protected:Npn \__siunitx_compound_extract_exponents:
132 {
133   \tl_set:Nx \l__siunitx_compound_tmp_tl
134     { \siunitx_number_output:NN \l__siunitx_compound_first_tl \q_nil }
135   \exp_after:wN \__siunitx_compound_extract_exponents_auxi:w
136     \l__siunitx_compound_tmp_tl \q_stop
137 }
138 \cs_new_protected:Npn \__siunitx_compound_extract_exponents_auxi:w
139   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil #8
140   \q_nil #9 \q_stop
141 {
142   \__siunitx_compound_extract_exponents_auxii:nw {#1#2#3#4#5#6#7#8} #9 \q_stop
143 }
144 \cs_new_protected:Npn \__siunitx_compound_extract_exponents_auxii:nw
145   #1#2 \q_nil #3 \q_nil #4 \q_stop
146 {
147   \seq_put_right:Nn \l__siunitx_compound_tmp_seq { #1#2 }
148   \tl_set:Nn \l__siunitx_compound_exp_tl { #3#4 }
149   \exp_after:wN \__siunitx_compound_extract_exponents_auxiii:nnnnnnnn
150     \l__siunitx_compound_first_tl
151 }
152 \cs_new_protected:Npn \__siunitx_compound_extract_exponents_auxiii:nnnnnnnn
153   #1#2#3#4#5#6#7
154 {

```

```

155 \keys_set:nn { siunitx }
156 {
157     drop-exponent = true ,
158     exponent-mode = fixed ,
159     fixed-exponent = #6#7
160 }
161 }

```

(End definition for `\siunitx_compound_number:n` and others. This function is documented on page 20.)

```

\__siunitx_compound_parsed:n The simple cases for parsing (or not) all entries.
\__siunitx_compound_unparsed:n
162 \cs_new_protected:Npn \__siunitx_compound_parsed:n #1
163 {
164     \bool_if:NTF \l__siunitx_compound_unit_bool
165     {
166         \siunitx_number_parse:nN {#1} \l__siunitx_compound_tmp_tl
167         \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_tmp_tl
168         \tl_set:Nx \l__siunitx_compound_tmp_tl
169         { \siunitx_number_output:NN \l__siunitx_compound_tmp_tl \q_nil }
170         \tl_set:Nx \l__siunitx_compound_tmp_tl
171         { \__siunitx_compound_uncert_bracket:N \l__siunitx_compound_tmp_tl }
172     }
173     { \siunitx_number_format:nN {#1} \l__siunitx_compound_tmp_tl }
174     \seq_put_right:NV \l__siunitx_compound_tmp_seq \l__siunitx_compound_tmp_tl
175 }
176 \cs_new_protected:Npn \__siunitx_compound_unparsed:n #1
177 {
178     \seq_put_right:Nn \l__siunitx_compound_tmp_seq { \ensuremath {#1} }
179 }

```

(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:nnnnnnnn
180 \cs_new_protected:Npn \__siunitx_compound_format_units:nn #1#2
181 {
182     \bool_if:NTF \l__siunitx_compound_unit_power_bool
183     {
184         \use:c { __siunitx_compound_format_ \l__siunitx_quantity_prefix_mode_tl :nn }
185         {#2} { \tl_count:n {#1} + 1 }
186     }
187     {
188         \use:c { __siunitx_compound_format_ \l__siunitx_quantity_prefix_mode_tl :n } {#2}
189     }
190 }
191 \cs_new_protected:cpx { __siunitx_compound_format_combine-exponent:n } #1
192 {
193     \exp_not:c { __siunitx_compound_format_combine-exponent_aux:n }
194     {
195         \exp_not:N \siunitx_unit_format_combine_exponent:nnN
196         {#1}
197     }

```

```

198 }
199 \cs_new_protected:cpx { __siunitx_compound_format_combine-exponent:nn } #1#2
200 {
201   \exp_not:c { __siunitx_compound_format_combine-exponent_aux:n }
202   {
203     \exp_not:N \siunitx_unit_format_multiply_combine_exponent:nnnN
204     {#1} {#2}
205   }
206 }
207 \cs_new_protected:cpn { __siunitx_compound_format_combine-exponent_aux:n } #1
208 {
209   \bool_set_true:N \l__siunitx_compound_exp_combine_bool
210   \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
211   \exp_args:NV \__siunitx_compound_extract_exp:nN
212     \l__siunitx_compound_first_tl \l__siunitx_compound_tmp_fp
213   #1 \l__siunitx_compound_tmp_fp \l__siunitx_compound_unit_tl
214 }
215 \cs_new_protected:cpx { __siunitx_compound_format_extract-exponent:n } #1
216 {
217   \exp_not:c { __siunitx_compound_format_extract-exponent_aux:n }
218   { \exp_not:N \siunitx_unit_format_extract_prefixes:nnN {#1} }
219 }
220 \cs_new_protected:cpx { __siunitx_compound_format_extract-exponent:nn } #1#2
221 {
222   \exp_not:c { __siunitx_compound_format_extract-exponent_aux:n }
223   {
224     \exp_not:N \siunitx_unit_format_multiply_extract_prefixes:nnNN
225     {#1} {#2}
226   }
227 }
228 \cs_new_protected:cpn { __siunitx_compound_format_extract-exponent_aux:n } #1
229 {
230   #1 \l__siunitx_compound_unit_tl \l__siunitx_compound_tmp_fp
231   \tl_set:Nx \l__siunitx_compound_tmp_tl
232     { \siunitx_number_adjust_exponent:Nn \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl }
233   \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
234   \bool_set_true:N \l__siunitx_compound_exp_combine_bool
235 }
236 \cs_new_protected:Npn \__siunitx_compound_format_input:n #1
237 {
238   \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
239   \siunitx_unit_format:nN {#1} \l__siunitx_compound_unit_tl
240 }
241 \cs_new_protected:Npn \__siunitx_compound_format_input:nn #1#2
242 {
243   \siunitx_number_process:NN \l__siunitx_compound_tmp_tl \l__siunitx_compound_first_tl
244   \siunitx_unit_format_multiply:nnN {#1} {#2} \l__siunitx_compound_unit_tl
245 }
246 \cs_new_protected:Npn \__siunitx_compound_extract_exp:nN #1#2
247 { \__siunitx_compound_extract_exp:nnnnnnN #1 #2 }
248 \cs_new_protected:Npn \__siunitx_compound_extract_exp:nnnnnnN #1#2#3#4#5#6#7#8
249 { \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.

```

250 \cs_new_protected:Npn \siunitx_compound_quantity:nn #1#2
251 {
252   \group_begin:
253     \bool_if:NT \l__siunitx_compound_unit_bracket_bool
254     { \bool_set_true:N \l__siunitx_compound_exp_bracket_bool }
255     \bool_if:NT \l__siunitx_compound_unit_repeat_bool
256     { \bool_set_false:N \l__siunitx_compound_exp_combine_bool }
257     \bool_lazy_or:nnT
258     { \l__siunitx_compound_unit_bracket_bool }
259     { ! \l__siunitx_compound_unit_repeat_bool }
260     { \bool_set_false:N \l__siunitx_number_bracket_ambiguous_bool }
261     \bool_set_true:N \l__siunitx_compound_unit_bool
262     \__siunitx_compound_format:nn {#1} {#2}
263     \bool_if:NF \l__siunitx_number_parse_bool
264     { \siunitx_unit_format:nN {#2} \l__siunitx_compound_unit_tl }
265     \str_if_eq:VnT \l__siunitx_quantity_prefix_mode_tl { combine-exponent }
266     { \tl_clear:N \l__siunitx_compound_exp_tl }
267     \bool_if:NTF \l__siunitx_compound_unit_repeat_bool
268     { \__siunitx_compound_print:N \__siunitx_compound_print_quantity:x }
269     {
270       \bool_lazy_and:nnTF
271       { \l__siunitx_compound_unit_bracket_bool }
272       { \tl_if_empty_p:N \l__siunitx_compound_exp_tl }
273       {
274         \siunitx_print_number:V \l__siunitx_compound_bracket_open_tl
275         \__siunitx_compound_print:N \siunitx_print_number:x
276         \siunitx_print_number:V \l__siunitx_compound_bracket_close_tl
277       }
278       { \__siunitx_compound_print:N \siunitx_print_number:x }
279       \__siunitx_compound_print_quantity:n { }
280     }
281   \group_end:
282 }

```

(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_use:Nnnn` is that we want to be able to insert `\siunitx_print_...:n` in a controlled way.

```

283 \cs_new_protected:Npn \__siunitx_compound_print:N #1
284 {
285   \bool_lazy_and:nnTF
286   { \l__siunitx_compound_exp_bracket_bool }
287   { ! \tl_if_empty_p:N \l__siunitx_compound_exp_tl }
288   {
289     \__siunitx_compound_print:xxN
290     { \exp_not:V \l__siunitx_compound_bracket_open_tl }
291     {
292       \exp_not:V \l__siunitx_compound_bracket_close_tl
293       \exp_not:V \l__siunitx_compound_exp_tl
294     }
295     #1

```



```

296     }
297     { \_siunitx_compound_print:xxN { } { \exp_not:V \l\_siunitx_compound_exp_tl } #1 }
298   }
299 \cs_new_protected:Npn \_siunitx_compound_print:nnN #1#2#3
300 {
301   \exp_args:Nx \_siunitx_compound_print:nnnN
302   { \seq_count:N \l\_siunitx_compound_tmp_seq } {#1} {#2} #3
303 }
304 \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.

```

305 \cs_new_protected:Npn \_siunitx_compound_print:nnnN #1#2#3#4
306 {
307   \int_case:nnF {#1}
308   {
309     { 0 } { }
310     { 1 }
311     {
312       #4
313       { \seq_item:Nn \l\_siunitx_compound_tmp_seq { 1 } }
314     }
315     { 2 }
316     {
317       #4
318       {
319         \exp_not:n {#2}
320         \seq_item:Nn \l\_siunitx_compound_tmp_seq { 1 }
321       }
322       \_siunitx_compound_print_separator:V \l\_siunitx_compound_separator_pair_tl
323       #4
324       {
325         \seq_item:Nn \l\_siunitx_compound_tmp_seq { 2 }
326         \exp_not:n {#3}
327       }
328     }
329   }
330   {
331     \int_set:Nn \l\_siunitx_compound_count_int {#1}
332     \tl_set:Nn \l\_siunitx_compound_start_tl {#2}
333     \tl_set:Nn \l\_siunitx_compound_end_tl {#3}
334     \cs_set_eq:NN \_siunitx_compound_print_aux:n #4
335     \seq_map_indexed_function:NN
336       \l\_siunitx_compound_tmp_seq
337       \_siunitx_compound_print_aux:nn
338   }
339 }
340 \cs_new_protected:Npn \_siunitx_compound_print_aux:n #1 { }
341 \cs_new_protected:Npn \_siunitx_compound_print_aux:nn #1#2
342 {
343   \int_case:nnF {#1}
344   {

```

```

345     { 1 }
346     {
347         \__siunitx_compound_print_aux:n
348         {
349             \exp_not:V \l__siunitx_compound_start_tl
350             \exp_not:n {#2}
351         }
352         \__siunitx_compound_print_separator:V \l__siunitx_compound_separator_tl
353     }
354     { \l__siunitx_compound_count_int - 1 }
355     {
356         \__siunitx_compound_print_aux:n { \exp_not:n {#2} }
357         \__siunitx_compound_print_separator:V \l__siunitx_compound_separator_final_tl
358     }
359     { \l__siunitx_compound_count_int }
360     {
361         \__siunitx_compound_print_aux:n
362         {
363             \exp_not:n {#2}
364             \exp_not:V \l__siunitx_compound_end_tl
365         }
366     }
367 }
368 {
369     \__siunitx_compound_print_aux:n { \exp_not:n {#2} }
370     \__siunitx_compound_print_separator:V \l__siunitx_compound_separator_tl
371 }
372 }
373 \cs_new_protected:Npn \__siunitx_compound_print_quantity:n #1
374 { \siunitx_quantity_print:nV {#1} \l__siunitx_compound_unit_tl }
375 \cs_generate_variant:Nn \__siunitx_compound_print_quantity:n { x }
376 \cs_new_protected:Npn \__siunitx_compound_print_separator:n #1
377 {
378     \bool_if:NTF \l__siunitx_compound_separator_text_bool
379     { #1 }
380     { \siunitx_print_number:n {#1} }
381 }
382 \cs_generate_variant:Nn \__siunitx_compound_print_separator:n { V }

```

(End definition for __siunitx_compound_print:N and others.)

__siunitx_compound_uncert_bracket:N
__siunitx_compound_uncert_bracket:w
__siunitx_compound_uncert_bracket:nnw

Check for the case where there is a separate uncertainty but not exponent, when we are handling units.

```

383 \cs_new:Npn \__siunitx_compound_uncert_bracket:N #1
384 { \exp_after:wN \__siunitx_compound_uncert_bracket:w #1 \q_stop }
385 \cs_new:Npn \__siunitx_compound_uncert_bracket:w
386 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
387 #8 \q_nil #9 \q_stop
388 { \__siunitx_compound_uncert_bracket:nnw {#1#2#3#4#5#6} {#7#8} #9 \q_stop }
389 \cs_new:Npn \__siunitx_compound_uncert_bracket:nnw #1#2 #3 \q_nil #4 \q_nil #5 \q_stop
390 {
391     \bool_lazy_or:nnTF
392     { \tl_if_blank_p:n {#2#3} }
393     { ! \tl_if_blank_p:n {#5} }

```

```

394     { \exp_not:n {#1#2#3#4#5} }
395     {
396       \exp_not:V \l__siunitx_compound_bracket_open_tl
397       \exp_not:n {#1#2#3}
398       \exp_not:V \l__siunitx_compound_bracket_close_tl
399       \exp_not:n {#4#5}
400     }
401   }

```

(End definition for `__siunitx_compound_uncert_bracket:N`, `__siunitx_compound_uncert_bracket:w`, and `__siunitx_compound_uncert_bracket:nnw`.)

1.2 Lists

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

```

402 <@@=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
403 \tl_new:N \l__siunitx_list_exp_tl
404 \tl_new:N \l__siunitx_list_units_tl
405 \keys_define:nn { siunitx }
406   {
407     list-exponents .choices:nn =
408       { combine , combine-bracket , individual }
409       { \tl_set_eq:NN \l__siunitx_list_exp_tl \l_keys_choice_tl } ,
410     list-final-separator .tl_set:N = \l_siunitx_list_separator_final_tl ,
411     list-pair-separator .tl_set:N = \l_siunitx_list_separator_pair_tl ,
412     list-separator .tl_set:N = \l_siunitx_list_separator_tl ,
413     list-units .choices:nn =
414       { bracket , repeat , single }
415       { \tl_set_eq:NN \l__siunitx_list_units_tl \l_keys_choice_tl }
416   }

```

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

```

\siunitx_number_list:nn
\siunitx_quantity_list:nn
\__siunitx_list_aux:
417 \cs_new_protected:Npn \siunitx_number_list:nn #1
418   {
419     \group_begin:
420       \__siunitx_list_aux:
421       \siunitx_compound_number:n {#1}
422     \group_end:
423   }
424 \cs_new_protected:Npn \siunitx_quantity_list:nn #1#2
425   {
426     \group_begin:
427       \__siunitx_list_aux:
428       \siunitx_compound_quantity:nn {#1} {#2}
429     \group_end:
430   }
431 \cs_new_protected:Npn \__siunitx_list_aux:
432   {

```

```

433 \keys_set:nx { siunitx }
434 {
435     compound-exponents      = \l__siunitx_list_exp_tl ,
436     compound-final-separator =
437     { \exp_not:V \l_siunitx_list_separator_final_tl } ,
438     compound-pair-separator =
439     { \exp_not:V \l_siunitx_list_separator_pair_tl } ,
440     compound-separator      =
441     { \exp_not:V \l_siunitx_list_separator_tl } ,
442     compound-separator-mode = text ,
443     compound-units          = \l__siunitx_list_units_tl
444 }
445 }

```

(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.

```

446 <@@=siunitx_product>

```

```

\l__siunitx_product_exp_tl
\l__siunitx_product_phrase_bool
\l__siunitx_product_phrase_tl
\l__siunitx_product_symbol_tl
\l__siunitx_product_units_tl
Options for products.
447 \tl_new:N \l__siunitx_product_exp_tl
448 \bool_new:N \l__siunitx_product_phrase_bool
449 \tl_new:N \l__siunitx_product_units_tl
450 \keys_define:nn { siunitx }
451 {
452     product-exponents .choices:nn =
453     { combine , combine-bracket , individual }
454     { \tl_set_eq:NN \l__siunitx_product_exp_tl \l_keys_choice_tl } ,
455     product-mode .choice: ,
456     product-mode / phrase .code:n =
457     { \bool_set_true:N \l__siunitx_product_phrase_bool } ,
458     product-mode / symbol .code:n =
459     { \bool_set_false:N \l__siunitx_product_phrase_bool } ,
460     product-phrase .tl_set:N = \l__siunitx_product_phrase_tl ,
461     product-symbol .tl_set:N = \l__siunitx_product_symbol_tl ,
462     product-units .choices:nn =
463     { bracket , bracket-power , power , repeat , single }
464     { \tl_set_eq:NN \l__siunitx_product_units_tl \l_keys_choice_tl }
465 }

```

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

```

\siunitx_number_product:n
\siunitx_quantity_product:nn
Simply recover the settings and use as a list.
466 \cs_new_protected:Npn \siunitx_number_product:n #1
467 {
468     \group_begin:
469     \_siunitx_product_aux:
470     \siunitx_compound_number:n {#1}
471     \group_end:
472 }

```

```

473 \cs_new_protected:Npn \siunitx_quantity_product:nn #1#2
474 {
475   \group_begin:
476     \__siunitx_product_aux:
477     \siunitx_compound_quantity:nn {#1} {#2}
478   \group_end:
479 }
480 \cs_new_protected:Npn \__siunitx_product_aux:
481 {
482   \bool_if:NTF \l__siunitx_product_phrase_bool
483     { \__siunitx_product_aux:x { \exp_not:V \l__siunitx_product_phrase_tl } }
484     { \__siunitx_product_aux:x { { } \exp_not:V \l__siunitx_product_symbol_tl { } } }
485 }
486 \cs_new_protected:Npn \__siunitx_product_aux:n #1
487 {
488   \keys_set:nx { siunitx }
489   {
490     compound-exponents      = \l__siunitx_product_exp_tl ,
491     compound-final-separator = { \exp_not:n {#1} } ,
492     compound-pair-separator = { \exp_not:n {#1} } ,
493     compound-separator      = { \exp_not:n {#1} } ,
494     compound-separator-mode =
495       \bool_if:NTF \l__siunitx_product_phrase_bool { text } { number } ,
496     compound-units          = \l__siunitx_product_units_tl
497   }
498 }
499 \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.

```
500 <@@=siunitx_range>
```

Options for products.

```

\l__siunitx_range_exp_tl
\l__siunitx_range_phrase_tl
\l__siunitx_range_units_tl
501 \tl_new:N \l__siunitx_range_exp_tl
502 \tl_new:N \l__siunitx_range_units_tl
503 \keys_define:nn { siunitx }
504 {
505   range-exponents .choices:nn =
506     { combine , combine-bracket , individual }
507     { \tl_set_eq:NN \l__siunitx_range_exp_tl \l_keys_choice_tl } ,
508   range-phrase .tl_set:N = \l__siunitx_range_phrase_tl ,
509   range-units .choices:nn =
510     { bracket , repeat , single }
511     { \tl_set_eq:NN \l__siunitx_range_units_tl \l_keys_choice_tl }
512 }

```

(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 Simply recover the settings and use as a list.
\siunitx_quantity_range:nnn
  \__siunitx_range_aux:
513 \cs_new_protected:Npn \siunitx_number_range:nn #1#2
514 {
515   \group_begin:
516     \__siunitx_range_aux:
517     \siunitx_compound_number:n { {#1} {#2} }
518   \group_end:
519 }
520 \cs_new_protected:Npn \siunitx_quantity_range:nnn #1#2#3
521 {
522   \group_begin:
523     \__siunitx_range_aux:
524     \siunitx_compound_quantity:nn { {#1} {#2} } {#3}
525   \group_end:
526 }
527 \cs_new_protected:Npn \__siunitx_range_aux:
528 {
529   \keys_set:nx { siunitx }
530   {
531     compound-exponents      = \l__siunitx_range_exp_tl ,
532     compound-pair-separator = { \exp_not:V \l_siunitx_range_phrase_tl } ,
533     compound-separator-mode = text ,
534     compound-units          = \l__siunitx_range_units_tl
535   }
536 }

```

(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.

```

537 \keys_set:nn { siunitx }
538 {
539   compound-exponents      = individual ,
540   compound-final-separator =
541   {
542     \ifmmode \ \else \space \fi
543     \text { and }
544     \ifmmode \ \else \space \fi
545   } ,
546   compound-pair-separator =
547   {
548     \ifmmode \ \else \space \fi
549     \text { and }
550     \ifmmode \ \else \space \fi
551   } ,
552   compound-separator      =
553   { , \ifmmode \ \else \space \fi } ,
554   compound-separator-mode = text ,
555   compound-units          = repeat ,
556   list-exponents          = individual ,
557   list-final-separator    =

```

```

558     {
559         \ifmmode \ \else \space \fi
560         \text { and }
561         \ifmmode \ \else \space \fi
562     } ,
563 list-pair-separator      =
564     {
565         \ifmmode \ \else \space \fi
566         \text { and }
567         \ifmmode \ \else \space \fi
568     } ,
569 list-separator           =
570     { , \ifmmode \ \else \space \fi } ,
571 list-units                = repeat ,
572 product-exponents        = individual ,
573 product-mode             = symbol ,
574 product-phrase           =
575     {
576         \ifmmode \ \else \space \fi
577         \text { by }
578         \ifmmode \ \else \space \fi
579     } ,
580 product-symbol           = \times ,
581 product-units            = repeat ,
582 range-exponents         = individual ,
583 range-phrase             =
584     {
585         \ifmmode \ \else \space \fi
586         \text { to }
587         \ifmmode \ \else \space \fi
588     } ,
589 range-units              = repeat
590 }
591 \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 { Catalan } { and } { i }
41     \DeclareTranslation { Catalan } { to~(numerical~range) } { a }
42     \DeclareTranslation { English } { to~(numerical~range) } { to }
43     \DeclareTranslation { French } { to~(numerical~range) } { à }
44     \DeclareTranslation { German } { to~(numerical~range) } { bis }
45     \DeclareTranslation { Spanish } { to~(numerical~range) } { a }
46     \keys_set:nn { siunitx }
47     {
48         list-final-separator =
49         {
50             \ifmmode \ \else \space \fi
51             \text { \GetTranslation { and } }
52             \ifmmode \ \else \space \fi
53         } ,
54         list-pair-separator =
55         {
56             \ifmmode \ \else \space \fi
57             \text { \GetTranslation { and } }
58             \ifmmode \ \else \space \fi
59         } ,
60         range-phrase =
61         {
62             \ifmmode \ \else \space \fi
63             \text { \GetTranslation { to~(numerical~range) } }
64             \ifmmode \ \else \space \fi
65         }
66     }
67 }

```

Part V

siunitx-number – Parsing and formatting numbers

This submodule is dedicated to parsing and formatting numbers. A small number of L^AT_EX 2_ε math mode commands are assumed to be available as part of the formatted output. The sign commands `\mp`, `\pm`, `\ll`, `\le`, `\gg` and `\ge` are used to replace two-character input; `\pm` is also required for the output of uncertainties. The standard settings require `\times`. For the display of colored negative numbers, the command `\color` is assumed to be available. Where the latter may apply, numbers should be printed inside a group: note that T_EX grouping is not added *within* formatted numbers as they may need to be decomposed into parts (see `\siunitx_number_output:NN`). Such a color will be the *first* part of the result, meaning that a test for an initial `\color` and following brace group may be used to detect/remove/adjust this part.

1 Formatting numbers

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

```
\siunitx_number_parse:nN {<number>} <tl var>
```

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

The structure of a valid number is:

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

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 *<integer>* and *<exponent>* parts: these are required. The *<uncertainty>* part should either be blank or contain an *<identifier>* (as a brace group), followed by one or more data entries. Valid *<identifiers>* currently are

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

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

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` if 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.

 $\backslash\text{siunitx_number_normalize_symbols:N}$ $\backslash\text{siunitx_number_normalize_symbols:N} \langle \text{tl var} \rangle$

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

 $\backslash\text{siunitx_if_number_p:n}$ \star $\backslash\text{siunitx_if_number_token:NTF}$ $\{\langle \text{tokens} \rangle\}$
 $\backslash\text{siunitx_if_number:nTF}$ \star $\{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}$

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

 $\backslash\text{siunitx_if_number_token:NTF}$ $\backslash\text{siunitx_if_number_token:NTF} \{\langle \text{token} \rangle\}$
 $\{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}$

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

 $\backslash\text{l_siunitx_bracket_ambiguous_bool}$

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

 $\backslash\text{l_siunitx_number_parse_bool}$

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

 $\backslash\text{l_siunitx_number_comparator_tl}$
 $\backslash\text{l_siunitx_number_exponent_tl}$
 $\backslash\text{l_siunitx_number_sign_tl}$

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

 $\backslash\text{l_siunitx_number_input_decimal_tl}$
 $\backslash\text{l_siunitx_number_output_decimal_tl}$

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 `l3keys 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`

<u>evaluate-expression</u>	evaluate-expression = true false
<u>exponent-base</u>	exponent-base = $\langle base \rangle$
<u>exponent-mode</u>	exponent-mode = engineering fixed input scientific
<u>exponent-product</u>	exponent-product = $\langle symbol \rangle$
<u>expression</u>	expression = $\langle expression \rangle$
<u>fixed-exponent</u>	fixed-exponent = $\langle exponent \rangle$
<u>group-digits</u>	group-digits = all decimal integer none
<u>group-minimum-digits</u>	group-minimum-digits = $\langle value \rangle$
<u>group-separator</u>	group-separator = $\langle symbol \rangle$
<u>input-close-uncertainty</u>	input-close-uncertainty = $\langle tokens \rangle$
<u>input-comparators</u>	input-comparators = $\langle tokens \rangle$
<u>input-close-uncertainty</u>	input-close-uncertainty = $\langle tokens \rangle$
<u>input-decimal-markers</u>	input-decimal-markers = $\langle tokens \rangle$
<u>input-digits</u>	input-digits = $\langle tokens \rangle$
<u>input-exponent-markers</u>	input-exponent-markers = $\langle tokens \rangle$
<u>input-open-uncertainty</u>	input-open-uncertainty = $\langle tokens \rangle$
<u>input-signs</u>	input-signs = $\langle tokens \rangle$
<u>input-uncertainty-signs</u>	input-uncertainty-signs = $\langle tokens \rangle$

<u>minimum-decimal-digits</u>	minimum-decimal-digits = $\langle min \rangle$
<u>minimum-integer-digits</u>	minimum-integer-digits = $\langle min \rangle$
<u>negative-color</u>	negative-color = $\langle color \rangle$
<u>output-close-uncertainty</u>	output-close-uncertainty = $\langle symbol \rangle$
<u>output-decimal-marker</u>	output-decimal-marker = $\langle symbol \rangle$
<u>output-open-uncertainty</u>	output-open-uncertainty = $\langle symbol \rangle$
<u>parse-numbers</u>	parse-numbers = true false
<u>print-implicit-plus</u>	print-implicit-plus = true false
<u>print-unity-mantissa</u>	print-unity-mantissa = true false
<u>print-zero-exponent</u>	print-zero-exponent = true false
<u>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>tight-spacing</u>	tight-spacing = true false

uncertainty-mode	uncertainty-mode = compact compact-marker full separate
------------------	---------------------------------------------------------

uncertainty-separator	uncertainty-separator = $\langle separator \rangle$
-----------------------	-----------------------------------------------------

2 siunitx-number implementation

Start the DocStrip guards.

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

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

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

2.1 Initial set-up

Variants not provided by expl3.

```
3 \cs_generate_variant:Nn \tl_if_blank:nTF { f }
4 \cs_generate_variant:Nn \tl_if_blank_p:n { f }
5 \cs_generate_variant:Nn \tl_if_in:NnTF { NV }
6 \cs_generate_variant:Nn \tl_remove_all:Nn { NV }
7 \cs_generate_variant:Nn \tl_replace_all:Nnn { NnV }
```

$\backslash l_siunitx_number_tmp_tl$ Scratch space.

```
8 \tl_new:N \l_siunitx_number_tmp_tl
```

(End definition for $\backslash l_siunitx_number_tmp_tl$.)

2.2 Main formatting routine

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

```
9 \tl_new:N \l_siunitx_number_outputted_tl
```

(End definition for $\backslash l_siunitx_number_outputted_tl$.)

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

```
10 \tl_new:N \l_siunitx_number_parse_bool
```

(End definition for $\backslash l_siunitx_number_parse_bool$. This variable is documented on page 40.)

$\backslash l_siunitx_number_parse_bool$ Top-level options.

```
11 \keys_define:nn { siunitx }
12 {
13   parse-numbers .bool_set:N = \l_siunitx_number_parse_bool
14 }
```

(End definition for $\backslash l_siunitx_number_parse_bool$. This variable is documented on page 40.)

`\siunitx_number_format:nN`

```

15 \cs_new_protected:Npn \siunitx_number_format:nN #1#2
16 {
17   \group_begin:
18     \bool_if:NTF \l_siunitx_number_parse_bool
19     {
20       \siunitx_number_parse:nN {#1} \l__siunitx_number_parsed_tl
21       \siunitx_number_process:NN \l__siunitx_number_parsed_tl \l__siunitx_number_parsed_tl
22       \tl_set:Nx \l__siunitx_number_outputted_tl
23         { \siunitx_number_output:N \l__siunitx_number_parsed_tl }
24     }
25     { \tl_set:Nn \l__siunitx_number_outputted_tl { \ensuremath {#1} } }
26   \exp_args:NNNV \group_end:
27   \tl_set:Nn #2 \l__siunitx_number_outputted_tl
28 }

```

(End definition for `\siunitx_number_format:nN`. This function is documented on page 39.)

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 `\integer` and `\exponent` parts.

A non-empty `\uncertainty` 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).

```

29 \tl_new:N \l_siunitx_number_input_decimal_tl

```

(End definition for `\l_siunitx_number_input_decimal_tl`. This variable is documented on page 40.)


```

\l_siunitx_number_expression_bool
\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

```

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

```

30 \keys_define:nn { siunitx }
31 {
32   evaluate-expression .bool_set:N =
33     \l_siunitx_number_expression_bool ,
34   expression .code:n =
35     \cs_set:Npn \__siunitx_number_expression:n ##1 {#1} ,
36   input-close-uncertainty .tl_set:N =
37     \l_siunitx_number_input_uncert_close_tl ,
38   input-comparators .tl_set:N =
39     \l_siunitx_number_input_comparator_tl ,
40   input-decimal-markers .tl_set:N =
41     \l_siunitx_number_input_decimal_tl ,
42   input-digits .tl_set:N =
43     \l_siunitx_number_input_digit_tl ,
44   input-exponent-markers .tl_set:N =
45     \l_siunitx_number_input_exponent_tl ,
46   input-ignore .tl_set:N =
47     \l_siunitx_number_input_ignore_tl ,
48   input-open-uncertainty .tl_set:N =
49     \l_siunitx_number_input_uncert_open_tl ,
50   input-signs .tl_set:N =
51     \l_siunitx_number_input_sign_tl ,
52   input-uncertainty-signs .code:n =
53     {
54       \tl_set:Nn \l_siunitx_number_input_uncert_sign_tl {#1}
55       \tl_map_inline:nn {#1}
56       {
57         \tl_if_in:NnF \l_siunitx_number_input_sign_tl {##1}
58         { \tl_put_right:Nn \l_siunitx_number_input_sign_tl {##1} }
59       }
60     } ,
61   parse-numbers .bool_set:N =
62     \l_siunitx_number_parse_bool ,
63   retain-explicit-plus .bool_set:N =
64     \l_siunitx_number_explicit_plus_bool ,
65   retain-zero-uncertainty .bool_set:N =
66     \l_siunitx_number_zero_uncert_bool
67 }
68 \cs_new:Npn \__siunitx_number_expression:n #1 { }
69 \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.

```
70 \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.

```
71 \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.

72 `\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.

73 `\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.

74 `\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.

75 `\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.

76 `\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.

77 `\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.

`_siunitx_number_normalize_aux:nN`
`_siunitx_number_normalize_sign:N`
`\c_siunitx_number_normalize_tl`

```
78 \cs_new_protected:Npn \siunitx_number_normalize_symbols:N #1
79 {
80   \_siunitx_number_normalize_minus:N #1
81   \exp_after:wN \_siunitx_number_normalize_aux:NnN \exp_after:wN #1
82   \c_siunitx_number_normalize_tl
83   { ? } \q_recursion_tail
84   \q_recursion_stop
85 }
86 \cs_set_protected:Npn \_siunitx_number_normalize_aux:NnN #1#2#3
87 {
88   \quark_if_recursion_tail_stop:N #3
89   \tl_replace_all:Nnn #1 {#2} {#3}
90   \_siunitx_number_normalize_aux:NnN #1
91 }
92 \tl_const:Nn \c_siunitx_number_normalize_tl
```

```

93 {
94   { -+ } \mp
95   { +- } \pm
96   { << } \ll
97   { <= } \le
98   { >> } \gg
99   { >= } \ge
100 }
101 \group_begin:
102   \char_set_catcode_active:N \-
103   \cs_new_protected:Npx \__siunitx_number_normalize_minus:N #1
104   {
105     \tl_replace_all:Nnn #1
106     { \exp_not:N - } { \token_to_str:N - }
107   }
108 \group_end:

```

(End definition for \siunitx_number_normalize_symbols:N and others. This function is documented on page 40.)

\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.

```

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

```

```

139   \tl_set:Nn #2 \l__siunitx_number_parsed_tl
140 }

```

(End definition for `\siunitx_number_parse:nN` and `_siunitx_number_parse:nN`. This function is documented on page 38.)

`_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

```

141 \cs_new_protected:Npn \_siunitx_number_parse_check:
142 {
143   \tl_if_empty:NF \l__siunitx_number_flex_tl
144   {
145     \bool_lazy_and:nnTF
146     {
147       \tl_if_blank_p:f
148       { \exp_after:wN \use_iv:nnnn \l__siunitx_number_parsed_tl }
149     }
150     {
151       \tl_if_blank_p:f
152       { \exp_after:wN \use_iv:nnnn \l__siunitx_number_flex_tl }
153     }
154     {
155       \tl_set:Nx \l__siunitx_number_tmp_tl
156       { \exp_after:wN \use_i:nnnn \l__siunitx_number_flex_tl }
157       \tl_if_in:NVTF \l__siunitx_number_input_uncert_sign_tl
158       \l__siunitx_number_tmp_tl
159       { \_siunitx_number_parse_combine_uncert: }
160       { \tl_clear:N \l__siunitx_number_parsed_tl }
161     }
162     { \tl_clear:N \l__siunitx_number_parsed_tl }
163   }
164   \tl_if_empty:NTF \l__siunitx_number_parsed_tl
165   {
166     \bool_if:NF \l__siunitx_number_validate_bool
167     {
168       \msg_error:nnx { siunitx } { invalid-number }
169       { \exp_not:V \l__siunitx_number_input_tl }
170     }
171   }
172   { \_siunitx_number_parse_finalise: }
173 }

```

(End definition for `_siunitx_number_parse_check:`.)

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

```

174 \cs_new_protected:Npn \_siunitx_number_parse_combine_uncert:
175 {
176   \exp_after:wN \exp_after:wN \exp_after:wN
177   \_siunitx_number_parse_combine_uncert_auxi:nnnnnnnn
178   \exp_after:wN \l__siunitx_number_parsed_tl \l__siunitx_number_flex_tl
179 }

```

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.

```

180 \cs_new_protected:Npn
181   \__siunitx_number_parse_combine_uncert_auxi:nnnnnnn #1#2#3#4#5#6#7#8
182   {
183     \__siunitx_number_parse_combine_uncert_auxii:fnnnn
184     { \int_eval:n { \tl_count:n {#3} - \tl_count:n {#7} } }
185     {#2} {#3} {#6} {#7}
186   }
187 \cs_new_protected:Npn
188   \__siunitx_number_parse_combine_uncert_auxii:nnnnn #1
189   {
190     \__siunitx_number_parse_combine_uncert_auxiii:fnnnnn
191     { \prg_replicate:nn { \int_abs:n {#1} } { 0 } }
192     {#1}
193   }
194 \cs_generate_variant:Nn \__siunitx_number_parse_combine_uncert_auxii:nnnnn { f }
195 \cs_new_protected:Npn
196   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn #1#2#3#4#5#6
197   {
198     \int_compare:nNnTF {#2} > 0
199     {
200       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
201       {#3} {#4} {#5} { #6 #1 }
202     }
203     {
204       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
205       {#3} { #4 #1 } {#5} {#6}
206     }
207   }
208 \cs_generate_variant:Nn
209   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn { f }
210 \cs_new_protected:Npn
211   \__siunitx_number_parse_combine_uncert_auxiv:nnnn #1#2#3#4
212   {
213     \tl_set:Nx \l__siunitx_number_parsed_tl
214     {
215       { \tl_head:V \l__siunitx_number_parsed_tl }
216       { \exp_not:n {#1} }
217       {
218         \bool_lazy_and:nnTF
219         { \tl_if_blank_p:n {#2} }
220         { ! \tl_if_blank_p:n {#4} }
221         { 0 }
222         { \exp_not:n {#2} }
223       }
224     }
225     \__siunitx_number_parse_combine_uncert_auxv:w #3#4

```

```

226         \q_recursion_tail \q_recursion_stop
227     }
228 }
229 }

```

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

```

230 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxv:w #1
231 {
232     \quark_if_recursion_tail_stop_do:Nn #1
233     {
234         \bool_if:NT \l__siunitx_number_zero_uncert_bool
235         { { S } { 0 } }
236     }
237     \str_if_eq:nnTF {#1} { 0 }
238     { \__siunitx_number_parse_combine_uncert_auxv:w }
239     { \__siunitx_number_parse_combine_uncert_auxvi:w #1 }
240 }
241 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxvi:w
242 #1 \q_recursion_tail \q_recursion_stop
243 { { 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.

```

244 \cs_new_protected:Npn \__siunitx_number_parse_comparator:
245 {
246     \exp_after:wN \__siunitx_number_parse_comparator_aux:Nw
247     \l__siunitx_number_arg_tl \q_stop
248 }
249 \cs_new_protected:Npn \__siunitx_number_parse_comparator_aux:Nw #1#2 \q_stop
250 {
251     \tl_if_in:NnTF \l__siunitx_number_input_comparator_tl {#1}
252     {
253         \tl_set:Nn \l__siunitx_number_comparator_tl {#1}
254         \tl_set:Nn \l__siunitx_number_arg_tl {#2}
255     }
256     { \tl_clear:N \l__siunitx_number_comparator_tl }
257     \tl_if_empty:NF \l__siunitx_number_arg_tl
258     { \__siunitx_number_parse_sign: }
259 }

```

(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.

```

260 \cs_new_protected:Npn \__siunitx_number_parse_exponent:
261 {
262   \tl_if_empty:NTF \l_siunitx_number_input_exponent_tl
263   {
264     \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 }
265     \tl_if_empty:NF \l__siunitx_number_parsed_tl
266     { \__siunitx_number_parse_loop: }
267   }
268   {
269     \tl_set:Nx \l__siunitx_number_tmp_tl
270     { \tl_head:V \l_siunitx_number_input_exponent_tl }
271     \tl_map_inline:Nn \l_siunitx_number_input_exponent_tl
272     {
273       \tl_replace_all:NnV \l__siunitx_number_arg_tl
274       {##1} \l__siunitx_number_tmp_tl
275     }
276     \use:x
277     {
278       \cs_set_protected:Npn
279       \exp_not:N \__siunitx_number_parse_exponent_auxi:w
280       ####1 \exp_not:V \l__siunitx_number_tmp_tl
281       ####2 \exp_not:V \l__siunitx_number_tmp_tl
282       ####3 \exp_not:N \q_stop
283     }
284     { \__siunitx_number_parse_exponent_auxii:nn {##1} {##2} }
285     \use:x
286     {
287       \__siunitx_number_parse_exponent_auxi:w
288       \exp_not:V \l__siunitx_number_arg_tl
289       \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_nil
290       \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_stop
291     }
292   }
293 }
294 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxi:w { }
295 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxii:nn #1#2
296 {
297   \quark_if_nil:nTF {#2}
298   { \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 } }
299   {
300     \tl_set:Nn \l__siunitx_number_arg_tl {#1}
301     \tl_if_blank:nTF {#2}
302     { \tl_clear:N \l__siunitx_number_parsed_tl }
303     { \__siunitx_number_parse_exponent_auxiii:Nw #2 \q_stop }
304   }
305   \tl_if_empty:NF \l__siunitx_number_parsed_tl
306   { \__siunitx_number_parse_loop: }
307 }
308 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxiii:Nw #1#2 \q_stop
309 {
310   \tl_if_in:NnTF \l_siunitx_number_input_sign_tl {#1}

```

```

311     { \_siunitx_number_parse_exponent_auxiv:nn {#1} {#2} }
312     { \_siunitx_number_parse_exponent_auxiv:nn { } {#1#2} }
313     \tl_if_empty:NT \l__siunitx_number_exponent_tl
314     { \tl_clear:N \l__siunitx_number_parsed_tl }
315   }
316 \cs_new_protected:Npn \_siunitx_number_parse_exponent_auxiv:nn #1#2
317 {
318   \bool_lazy_or:nnTF
319   { \l__siunitx_number_explicit_plus_bool }
320   { ! \str_if_eq_p:nn {#1} { + } }
321   { \tl_set:Nn \l__siunitx_number_exponent_tl { {#1} } }
322   { \tl_set:Nn \l__siunitx_number_exponent_tl { { } } }
323   \tl_if_blank:nTF {#2}
324   { \tl_clear:N \l__siunitx_number_parsed_tl }
325   {
326     \_siunitx_number_parse_exponent_zero_test:N #2
327     \q_recursion_tail \q_recursion_stop
328   }
329 }
330 \cs_new_protected:Npn \_siunitx_number_parse_exponent_zero_test:N #1
331 {
332   \quark_if_recursion_tail_stop_do:Nn #1
333   { \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 } }
334   \str_if_eq:nnTF {#1} { 0 }
335   { \_siunitx_number_parse_exponent_zero_test:N }
336   { \_siunitx_number_parse_exponent_check:N #1 }
337 }
338 \cs_new_protected:Npn \_siunitx_number_parse_exponent_check:N #1
339 {
340   \quark_if_recursion_tail_stop:N #1
341   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#1}
342   {
343     \tl_put_right:Nn \l__siunitx_number_exponent_tl {#1}
344     \_siunitx_number_parse_exponent_check:N
345   }
346   { \_siunitx_number_parse_exponent_cleanup:wN }
347 }
348 \cs_new_protected:Npn \_siunitx_number_parse_exponent_cleanup:wN
349 #1 \q_recursion_stop
350 { \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.

```

351 \cs_new_protected:Npn \_siunitx_number_parse_finalise:
352 {
353   \tl_if_empty:NF \l__siunitx_number_parsed_tl
354   {
355     \tl_set:Nx \l__siunitx_number_parsed_tl
356     {
357       { \exp_not:V \l__siunitx_number_comparator_tl }
358       \exp_not:V \l__siunitx_number_parsed_tl
359       \exp_after:wN \_siunitx_number_parse_finalise:nw

```



```

360         \l__siunitx_number_exponent_tl \q_stop
361     }
362 }
363 }
364 \cs_new:Npn \__siunitx_number_parse_finalise:nw #1#2 \q_stop
365 {
366     { \exp_not:n {#1} }
367     { \exp_not:n {#2} }
368 }

```

(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.

```

369 \cs_new_protected:Npn \__siunitx_number_parse_loop:
370 {
371     \tl_clear:N \l__siunitx_number_partial_tl
372     \exp_after:wN \__siunitx_number_parse_loop_first:NNN
373     \exp_after:wN \l__siunitx_number_parsed_tl \exp_after:wN \c_true_bool
374     \l__siunitx_number_arg_tl
375     \q_recursion_tail \q_recursion_stop
376 }

```

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.

```

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

```

```

393         \tl_put_right:Nn #1 { { 0 } }
394         \__siunitx_number_parse_loop_after_decimal:NNN #1 #2
395     }
396     { \__siunitx_number_parse_loop_break:wN }
397 }
398 }

```

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.

```

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

```

```

425         { \_siunitx_number_parse_loop_break:wN }
426     }
427 }
428 }
429 }

```

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.

```

430 \cs_new_protected:Npn \_siunitx_number_parse_loop_main_end:NN #1#2
431 {
432     \bool_lazy_and:nnT
433     {#2} { \tl_if_empty_p:N \l__siunitx_number_partial_tl }
434     { \tl_set:Nn \l__siunitx_number_partial_tl { 0 } }
435     \tl_put_right:Nx #1
436     {
437         { \exp_not:V \l__siunitx_number_partial_tl }
438         \bool_if:NT #2 { { } }
439     }
440 }
441 }

```

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.

```

442 \cs_new_protected:Npn \_siunitx_number_parse_loop_main_digit:NNNN #1#2#3#4#5
443 {
444     \bool_lazy_or:nnTF
445     {#3} { ! \str_if_eq_p:nn {#5} { 0 } }
446     {
447         \tl_put_right:Nn \l__siunitx_number_partial_tl {#5}
448         \_siunitx_number_parse_loop_main:NNNN #1 #2 \c_true_bool #4
449     }
450     { \_siunitx_number_parse_loop_main:NNNN #1 #2 \c_false_bool #4 }
451 }

```

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.

```

452 \cs_new_protected:Npn \_siunitx_number_parse_loop_main_decimal:NN #1#2
453 {
454     \_siunitx_number_parse_loop_main_store:NNN #1 \c_false_bool \c_false_bool
455     \_siunitx_number_parse_loop_after_decimal:NNN #1 #2
456 }

```

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.

```

457 \cs_new_protected:Npn \_siunitx_number_parse_loop_main_uncert:NNN #1#2#3
458 {
459     \_siunitx_number_parse_loop_main_store:NNN #1 #2 \c_false_bool

```

```

460   \__siunitx_number_parse_uncert:NN #1
461 }

```

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

```

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

```

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

```

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

```

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

```

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

```

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.

```

500 \cs_new_protected:Npn \__siunitx_number_parse_loop_break:wN

```

```

501 #1 \q_recursion_stop
502 {
503   \tl_clear:N \l__siunitx_number_flex_tl
504   \tl_clear:N \l__siunitx_number_parsed_tl
505 }

```

(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.

```

506 \cs_new_protected:Npn \__siunitx_number_parse_sign:
507 {
508   \exp_after:wN \__siunitx_number_parse_sign_aux:Nw
509   \l__siunitx_number_arg_tl \q_stop
510 }
511 \cs_new_protected:Npn \__siunitx_number_parse_sign_aux:Nw #1#2 \q_stop
512 {
513   \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#1}
514   {
515     \tl_set:Nn \l__siunitx_number_arg_tl {#2}
516     \bool_lazy_and:nnTF
517     { \token_if_eq_charcode_p:NN #1 + }
518     { ! \l__siunitx_number_explicit_plus_bool }
519     { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
520     { \tl_set:Nn \l__siunitx_number_parsed_tl { {#1} } }
521   }
522   { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
523   \tl_if_empty:NTF \l__siunitx_number_arg_tl
524   { \tl_clear:N \l__siunitx_number_parsed_tl }
525   { \__siunitx_number_parse_exponent: }
526 }

```

(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_extend:nnnN
__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.

```

527 \cs_new_protected:Npn \__siunitx_number_parse_uncert:NN #1#2
528 {
529   \quark_if_recursion_tail_stop_do:Nn #2
530   { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
531   \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#2}
532   { \__siunitx_number_parse_loop_break:wN }
533   {
534     \__siunitx_number_parse_uncert:NNNN
535     #1 \c_false_bool \__siunitx_number_parse_uncert_auxi:NN #2
536   }
537 }

```

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

```

538 \cs_new_protected:Npn \__siunitx_number_parse_uncert:NNNN #1#2#3#4
539 {

```

```

540 \quark_if_recursion_tail_stop_do:Nn #4
541 { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
542 \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#4}
543 {
544   \bool_lazy_or:nnTF
545     {#2} { ! \str_if_eq_p:nn {#4} { 0 } }
546   {
547     \tl_put_right:Nn \l__siunitx_number_partial_tl {#4}
548     \__siunitx_number_parse_uncert:NNNN #1 \c_true_bool #3
549   }
550   { \__siunitx_number_parse_uncert:NNNN #1 \c_false_bool #3 }
551 }
552 { #3 #1#4 }
553 }

```

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

```

554 \cs_new_protected:Npn \__siunitx_number_parse_uncert_auxi:NN #1#2
555 {
556   \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#2}
557   {
558     \__siunitx_number_parse_uncert_auxiii:N #1
559     \__siunitx_number_parse_uncert_after:N
560   }
561   {
562     \tl_if_in:NnTF \l__siunitx_number_input_decimal_tl {#2}
563     { \__siunitx_number_parse_uncert_marker:N #1 }
564     { \__siunitx_number_parse_loop_break:wN }
565   }
566 }
567 \cs_new_protected:Npn \__siunitx_number_parse_uncert_auxii:NN #1#2
568 {
569   \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#2}
570   {
571     \__siunitx_number_parse_uncert_auxiii:N #1
572     \__siunitx_number_parse_uncert_after:N
573   }
574   { \__siunitx_number_parse_loop_break:wN }
575 }

```

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

```

576 \cs_new_protected:Npn \__siunitx_number_parse_uncert_auxiii:N #1
577 {
578   \tl_if_empty:NnTF \l__siunitx_number_partial_tl
579   {
580     \tl_put_right:Nx #1
581     {
582       {
583         \bool_if:NT \l__siunitx_number_zero_uncert_bool
584         { { S } { 0 } }
585       }
586     }
587   }
588   {

```

```

589      \tl_set:Nx \l__siunitx_number_partial_tl
590      { { S } { \exp_not:V \l__siunitx_number_partial_tl } }
591      \__siunitx_number_parse_loop_main_store:NNN #1
592      \c_false_bool \c_false_bool
593    }
594  }

```

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, and extend the decimal if it is. 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.

```

595 \cs_new_protected:Npn \__siunitx_number_parse_uncert_marker:N #1
596 { \exp_after:wN \__siunitx_number_parse_uncert_marker:nnnN #1 #1 }
597 \cs_new_protected:Npn \__siunitx_number_parse_uncert_marker:nnnN #1#2#3#4
598 {
599   \int_compare:nNnTF
600     { \tl_count:N \l__siunitx_number_partial_tl } > { \tl_count:n {#2} }
601     { \__siunitx_number_parse_loop_break:wN }
602     { \__siunitx_number_parse_uncert_marker:nNw {#3} #4 }
603 }
604 \cs_new_protected:Npn \__siunitx_number_parse_uncert_marker:nNw
605   #1#2#3 \q_recursion_tail \q_recursion_stop
606 {
607   \int_compare:nNnTF
608     { \tl_count:n {#3} - 1 } = { \tl_count:n {#1} }
609     {
610       \str_if_eq:eeTF
611         { \exp_not:V \l__siunitx_number_partial_tl }
612         { \prg_replicate:nn { \tl_count:N \l__siunitx_number_partial_tl } { 0 } }
613         {
614           \__siunitx_number_parse_uncert:NNNN
615             #2 \c_false_bool
616         }
617         {
618           \__siunitx_number_parse_uncert:NNNN
619             #2 \c_true_bool
620         }
621       \__siunitx_number_parse_uncert_auxii:NN
622     }
623     { \exp_after:wN \__siunitx_number_parse_uncert_extend:nnnN #2 #2 }
624     #3 \q_recursion_tail \q_recursion_stop
625 }
626 \cs_new_protected:Npn \__siunitx_number_parse_uncert_extend:nnnN #1#2#3#4
627 {
628   \tl_set:Nn #4 { {#1} {#2} { #3 0 } }
629   \__siunitx_number_parse_uncert:NNNN #4 \c_true_bool
630   \__siunitx_number_parse_uncert_auxii:NN
631 }

```

No further tokens are allowed after an uncertainty in parenthesis.

```

632 \cs_new_protected:Npn \__siunitx_number_parse_uncert_after:N #1

```

```

633 {
634   \quark_if_recursion_tail_stop:N #1
635   \__siunitx_number_parse_loop_break:wN
636 }

```

(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

```

```

637 \keys_define:nn { siunitx }
638 {
639   drop-exponent .bool_set:N =
640     \l__siunitx_number_drop_exponent_bool ,
641   drop-uncertainty .bool_set:N =
642     \l__siunitx_number_drop_uncertainty_bool ,
643   drop-zero-decimal .bool_set:N =
644     \l__siunitx_number_drop_zero_decimal_bool ,
645   exponent-mode .choices:nn =
646     { engineering , fixed , input , scientific }
647     { \tl_set_eq:NN \l__siunitx_number_exponent_mode_tl \l_keys_choice_tl } ,
648   fixed-exponent .int_set:N =
649     \l__siunitx_number_exponent_fixed_int ,
650   minimum-decimal-digits .int_set:N =
651     \l__siunitx_number_min_decimal_int ,
652   minimum-integer-digits .int_set:N =
653     \l__siunitx_number_min_integer_int ,
654   round-half .choice: ,
655   round-half / even .code:n =
656     { \bool_set_true:N \l__siunitx_number_round_half_even_bool } ,
657   round-half / up .code:n =
658     { \bool_set_false:N \l__siunitx_number_round_half_even_bool } ,
659   round-minimum .code:n =
660     { \__siunitx_number_set_round_min:n {#1} } ,
661   round-mode .choices:nn =
662     { figures , none , places , uncertainty }
663     { \tl_set_eq:NN \l__siunitx_number_round_mode_tl \l_keys_choice_tl } ,
664   round-pad .bool_set:N =
665     \l__siunitx_number_round_pad_bool ,
666   round-precision .int_set:N =
667     \l__siunitx_number_round_precision_int ,
668 }
669 \bool_new:N \l__siunitx_number_round_half_even_bool
670 \tl_new:N \l__siunitx_number_exponent_mode_tl
671 \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.

```

672 \tl_new:N \l__siunitx_number_round_min_tl

```

(End definition for \l__siunitx_number_round_min_tl.)


```

\__siunitx_number_set_round_min:n
\__siunitx_number_set_round_min:nnnnnnn

```

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

```

673 \cs_new_protected:Npn \__siunitx_number_set_round_min:n #1
674 {
675   \siunitx_number_parse:nN {#1} \l__siunitx_number_tmp_tl
676   \exp_after:wN \__siunitx_number_set_round_min:nnnnnnn \l__siunitx_number_tmp_tl
677 }
678 \cs_new:Npn \__siunitx_number_set_round_min:nnnnnnn #1#2#3#4#5#6#7
679 {
680   \tl_set:Nx \l__siunitx_number_round_min_tl
681   {
682     \bool_lazy_and:nnF
683     { \str_if_eq_p:nn {#3} { 0 } }
684     {
685       \str_if_eq_p:ee
686       { \exp_not:n {#4} }
687       { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
688     }
689     { \exp_not:n { {#3} {#4} } }
690   }
691 }

```

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

```

\siunitx_number_process:NN
\__siunitx_number_process:nnnnnnnNN

```

A top-level interface for the processing tools. Rounding happens in all cases, but exponents are only processed if the value is not 0.

```

692 \cs_new_protected:Npn \siunitx_number_process:NN #1#2
693 {
694   \tl_if_empty:NTF #1
695   { \tl_clear:N #2 }
696   {
697     \__siunitx_number_drop_uncertainty:NN #1 #2
698     \exp_after:wN \__siunitx_number_process:nnnnnnnNN #2 #2 #2
699     \__siunitx_number_drop_exponent:NN #2 #2
700     \__siunitx_number_zero_decimal:NN #2 #2
701     \__siunitx_number_digits:NN #2 #2
702   }
703 }
704 \cs_new_protected:Npn \__siunitx_number_process:nnnnnnnNN #1#2#3#4#5#6#7#8#9
705 {
706   \bool_lazy_and:nnTF
707   { \str_if_eq_p:nn {#3} { 0 } }
708   {
709     \str_if_eq_p:ee
710     { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
711   }
712   { \__siunitx_number_round:NN #8 #9 }
713   {
714     \__siunitx_number_exponent:NN #8 #9
715     \__siunitx_number_round:NN #9 #9
716   }
717 }

```

(End definition for `\siunitx_number_process:NN` and `_siunitx_number_process:nnnnnnnNN`. This function is documented on page 39.)

```

\siunitx_number_exponent:NN
siunitx_number_exponent_engineering:nnnnnnn
\_siunitx_number_exponent_fixed:nnnnnnn
\_siunitx_number_exponent_input:nnnnnnn
siunitx_number_exponent_scientific:nnnnnnn
\_siunitx_number_exponent_fixed:nnnnnnnn
siunitx_number_exponent_scientific:nnnnnnn
\_siunitx_number_exponent_scientific:nnnw
\_siunitx_number_exponent_shift:nnn
\_siunitx_number_exponent_shift:nnf
\_siunitx_number_exponent_shift_down:nnnw
\_siunitx_number_exponent_shift_down:nnn
\_siunitx_number_exponent_shift_down:nw
\_siunitx_number_exponent_shift_up:nnn
\_siunitx_number_exponent_shift_up:nnw
\_siunitx_number_exponent_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
itx_number_exponent_engineering_aux:nnnnnnn
\_siunitx_number_exponent_engineering_0:nnnn
\_siunitx_number_exponent_engineering_1:nnnn
\_siunitx_number_exponent_engineering_2:nnnn
\_siunitx_number_exponent_engineering:nnNw
unitx_number_exponent_engineering_uncert:nn
tx_number_exponent_engineering_uncert_S:nnn

718 \cs_new_protected:Npn \_siunitx_number_exponent:NN #1#2
719 {
720   \tl_set:Nx #2
721   {
722     \cs:w
723       __siunitx_number_exponent_ \l__siunitx_number_exponent_mode_tl :nnnnnnn
724       \exp_after:wN
725       \cs_end: #1
726   }
727   \str_if_eq:VnT \l__siunitx_number_exponent_mode_tl { engineering }
728   {
729     \tl_set:Nx #2
730     { \exp_after:wN \_siunitx_number_exponent_engineering_aux:nnnnnnn #2 }
731   }
732 }
733 \cs_new:Npn \_siunitx_number_exponent_fixed:nnnnnnn #1#2#3#4#5#6#7
734 {
735   \exp_args:Nf \_siunitx_number_exponent_fixed:nnnnnnnn
736   { \int_eval:n { \l__siunitx_number_exponent_fixed_int - (#6#7) } }
737   {#1} {#2} {#3} {#4} {#5} {#6} {#7}
738 }
739 \cs_new:Npn \_siunitx_number_exponent_fixed:nnnnnnnn #1#2#3#4#5#6#7#8
740 {
741   \exp_not:n { {#2} {#3} }
742   \_siunitx_number_exponent_shift:nnn {#1} {#4} {#5}
743   \_siunitx_number_exponent_uncert:n {#6}
744   \exp_not:n { {#7} } { \int_use:N \l__siunitx_number_exponent_fixed_int }
745 }
746 \cs_new:Npn \_siunitx_number_exponent_input:nnnnnnn #1#2#3#4#5#6#7
747 { \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.

```

748 \cs_new:Npn \_siunitx_number_exponent_scientific:nnnnnnn #1#2#3#4#5#6#7
749 {
750   \exp_args:Nf \_siunitx_number_exponent_scientific:nnnnnnnn
751   { \int_eval:n { \tl_count:n {#3} } }
752   {#1} {#2} {#3} {#4} {#5} {#6} {#7}
753 }
754 \cs_new:Npn \_siunitx_number_exponent_scientific:nnnnnnnn #1#2#3#4#5#6#7#8
755 {
756   \exp_not:n { {#2} {#3} }
757   \int_compare:nNnTF {#1} = 1
758   {
759     \str_if_eq:nnTF {#4} { 0 }
760     {

```

```

761         \__siunitx_number_exponent_scientific:nnnw
762         { 0 } {#6} { #7#8 } #5 \q_stop
763     }
764     { \exp_not:n { {#4} {#5} {#6} {#7} {#8} } }
765 }
766 {
767     \__siunitx_number_exponent_shift:nnn { #1 - 1 } {#4} {#5}
768     \__siunitx_number_exponent_uncert:n {#6}
769     \__siunitx_number_exponent_finalise:n { #1 + #7#8 - 1 }
770 }
771 }
772 \cs_new_eq:NN \__siunitx_number_exponent_engineering:nnnnnnn
773 \__siunitx_number_exponent_scientific:nnnnnnn
774 \cs_new:Npn \__siunitx_number_exponent_scientific:nnnw #1#2#3#4#5 \q_stop
775 {
776     \str_if_eq:nnTF {#4} { 0 }
777     {
778         \__siunitx_number_exponent_scientific:nnnw
779         { #1 - 1 } {#2} {#3} #5 \q_stop
780     }
781     {
782         \exp_not:n { {#4} {#5} {#2} }
783         \__siunitx_number_exponent_finalise:n { #1 + #3 - 1 }
784     }
785 }

```

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

```

786 \cs_new:Npn \__siunitx_number_exponent_shift:nnn #1#2#3
787 {
788     \int_compare:nNnTF {#1} > 0
789     { \__siunitx_number_exponent_shift_down:nnnw {#1} {#3} { } #2 \q_stop }
790     {
791         \int_compare:nNnTF {#1} < 0
792         { \__siunitx_number_exponent_shift_up:nnn {#1} {#2} {#3} }
793         { {#2} {#3} }
794     }
795 }
796 \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.

```

797 \cs_new:Npn \__siunitx_number_exponent_shift_down:nnnw #1#2#3#4#5 \q_stop
798 {
799     \tl_if_blank:nTF {#5}
800     { \__siunitx_number_exponent_shift_down:nnn {#1} { #4 #3 } {#2} }
801     { \__siunitx_number_exponent_shift_down:nnnw {#1} {#2} { #4 #3 } #5 \q_stop }
802 }
803 \cs_new:Npn \__siunitx_number_exponent_shift_down:nnn #1#2#3
804 {
805     \int_compare:nNnTF {#1} = 0
806     { { \tl_reverse:n {#2} } \exp_not:n { {#3} } }
807     { \__siunitx_number_exponent_shift_down:nw {#1} #2 \q_stop {#3} }
808 }

```

```

809 \cs_new:Npn \__siunitx_number_exponent_shift_down:nw #1#2#3 \q_stop #4
810 {
811   \tl_if_blank:nTF {#3}
812   { \__siunitx_number_exponent_shift_down:nnn { #1 - 1 } { 0 } { #2#4 } }
813   { \__siunitx_number_exponent_shift_down:nnn { #1 - 1 } {#3} { #2#4 } }
814 }

```

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.

```

815 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnn #1#2#3
816 {
817   \tl_if_blank:nTF {#3}
818   {
819     \__siunitx_number_exponent_shift_up_aux:ffn
820     { \int_eval:n { #1 + 1 } }
821     { \str_if_eq:nnF {#2} { 0 } {#2} 0 }
822     { }
823     \__siunitx_number_exponent_shift_uncert:nw { 1 }
824   }
825   { \__siunitx_number_exponent_shift_up:nnw {#1} {#2} #3 \q_stop }
826 }
827 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnw #1#2#3#4 \q_stop
828 {
829   \__siunitx_number_exponent_shift_up_aux:ffn
830   { \int_eval:n { #1 + 1 } }
831   { \str_if_eq:nnF {#2} { 0 } {#2} #3 }
832   {#4}
833 }
834 \cs_new:Npn \__siunitx_number_exponent_shift_up_aux:nnn #1#2#3
835 {
836   \int_compare:nNnTF {#1} = 0
837   { \exp_not:n { {#2} {#3} } }
838   {
839     \tl_if_blank:nTF {#3}
840     {
841       {
842         \exp_not:n {#2}
843         \prg_replicate:nn { \int_abs:n {#1} } { 0 }
844       }
845       { }
846       \__siunitx_number_exponent_shift_uncert:nw { \int_abs:n {#1} }
847     }
848     { \__siunitx_number_exponent_shift_up:nnn {#1} {#2} {#3} }
849   }
850 }
851 \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.

```

852 \cs_new:Npn \__siunitx_number_exponent_shift_uncert:nw
853   #1#2 \__siunitx_number_exponent_uncert:n #3
854 {
855   \tl_if_blank:nTF {#3}

```

```

856 {
857   #2
858   \__siunitx_number_exponent_uncert:n { }
859 }
860 {
861   \str_if_eq:nnTF {#3} { 0 }
862   {
863     #2
864     \__siunitx_number_exponent_uncert:n { { S } { 0 } }
865   }
866   {
867     \use:c { __siunitx_number_exponent_shift_uncert_ \use_i:nn #3 :fnnn }
868     { \prg_replicate:nn {#1} { 0 } }
869     {#2}
870     #3
871   }
872 }
873 }
874 \cs_new:Npn \__siunitx_number_exponent_shift_uncert_S:nnnn #1#2#3#4
875 {
876   #2
877   \__siunitx_number_exponent_uncert:n { { S } { #4#1 } }
878 }
879 \cs_generate_variant:Nn \__siunitx_number_exponent_shift_uncert_S:nnnn { f }
880 \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.

```

881 \cs_new:Npn \__siunitx_number_exponent_finalise:n #1
882 {
883   \int_compare:nNnTF {#1} < 0
884   { { - } }
885   { { } }
886   { \int_abs:n {#1} }
887 }

```

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.

```

888 \cs_new:Npn \__siunitx_number_exponent_engineering_aux:nnnnnnn #1#2#3#4#5#6#7
889 {
890   \exp_not:n { {#1} {#2} }
891   \use:c
892   {
893     __siunitx_number_exponent_engineering_
894     \int_compare:nNnTF {#6#7} < 0
895     {
896       \int_case:nnF { \int_mod:nn { #7 } { 3 } }
897       {
898         { 1 } { 2 }
899         { 2 } { 1 }
900       }
901       { 0 }
902     }
903     { \int_mod:nn {#7} { 3 } }

```

```

904         :nnnn
905     }
906     {#3} {#4} {#5} {#6#7}
907 }
908 \cs_new:cpn { __siunitx_number_exponent_engineering_0:nnnn } #1#2#3#4
909 {
910     \exp_not:n { {#1} {#2} {#3} }
911     \__siunitx_number_exponent_finalise:n {#4}
912 }
913 \cs_new:cpn { __siunitx_number_exponent_engineering_1:nnnn } #1#2#3#4
914 {
915     \tl_if_blank:nTF {#2}
916     {
917         { \exp_not:n { #1 0 } } { }
918         { \__siunitx_number_exponent_engineering_uncert:nn {#3} { 0 } }
919     }
920     {
921         { \exp_not:n {#1} \exp_not:o { \tl_head:w #2 \q_stop } }
922         { \exp_not:f { \tl_tail:n {#2} } }
923         { \exp_not:n {#3} }
924     }
925     \__siunitx_number_exponent_finalise:n { #4 - 1 }
926 }
927 \cs_new:cpn { __siunitx_number_exponent_engineering_2:nnnn } #1#2#3#4
928 {
929     \tl_if_blank:nTF {#2}
930     {
931         { \exp_not:n { #1 00 } } { }
932         { \__siunitx_number_exponent_engineering_uncert:nn {#3} { 00 } }
933     }
934     { \__siunitx_number_exponent_engineering:nnNw {#1} {#3} #2 \q_stop }
935     \__siunitx_number_exponent_finalise:n { #4 - 2 }
936 }
937 \cs_new:Npn \__siunitx_number_exponent_engineering:nnNw #1#2#3#4 \q_stop
938 {
939     \tl_if_blank:nTF {#4}
940     {
941         { \exp_not:n { #1#3 0 } } { }
942         { \__siunitx_number_exponent_engineering_uncert:nn {#2} { 0 } }
943     }
944     {
945         { \exp_not:n {#1#3} \exp_not:o { \tl_head:w #4 \q_stop } }
946         { \exp_not:f { \tl_tail:n {#4} } }
947         { \exp_not:n {#2} }
948     }
949 }
950 \cs_new:Npn \__siunitx_number_exponent_engineering_uncert:nn #1#2
951 {
952     \tl_if_blank:nF {#1}
953     {
954         \use:c { __siunitx_number_exponent_engineering_uncert_ \use_i:nn #1 :nnn }
955         #1 {#2}
956     }
957 }

```

```

958 \cs_new:Npn \__siunitx_number_exponent_engineering_uncert_S:nnn #1#2#3
959 {
960   { S }
961   {
962     \exp_not:n {#2}
963     \str_if_eq:nnF {#2} { 0 } {#3}
964   }
965 }

```

(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:nnnnnnn
\__siunitx_number_digits:Nn
\__siunitx_number_digits:nn
\__siunitx_number_digits_S:n
966 \cs_new_protected:Npn \__siunitx_number_digits:NN #1#2
967 {
968   \tl_set:Nx #2
969     { \exp_after:wN \__siunitx_number_digits:nnnnnnn #1 }
970 }
971 \cs_new:Npn \__siunitx_number_digits:nnnnnnn #1#2#3#4#5#6#7
972 {
973   \exp_not:n { {#1} {#2} }
974   {
975     \__siunitx_number_digits:Nn \l__siunitx_number_min_integer_int {#3}
976     \exp_not:n {#3}
977   }
978   {
979     \exp_not:n {#4}
980     \__siunitx_number_digits:Nn \l__siunitx_number_min_decimal_int {#4}
981   }
982   { \tl_if_blank:nF {#5} { \__siunitx_number_digits_uncert:nn #5 } }
983   \exp_not:n { {#6} {#7} }
984 }
985 \cs_new:Npn \__siunitx_number_digits:Nn #1#2
986 {
987   \int_compare:nNnT
988     { #1 - \tl_count:n {#2} } > 0
989     { \prg_replicate:nn { #1 - \tl_count:n {#2} } { 0 } }
990 }
991 \cs_new:Npn \__siunitx_number_digits_uncert:nn #1#2
992 {
993   { #1 }
994   { \use:c { __siunitx_number_digits_uncert_ #1 :n } {#2} }
995 }
996 \cs_new:Npn \__siunitx_number_digits_uncert_S:n #1
997 {
998   \exp_not:n {#1}
999   \__siunitx_number_digits:Nn \l__siunitx_number_min_decimal_int {#1}
1000 }

```

(End definition for __siunitx_number_digits:NN and others.)

__siunitx_number_drop_exponent:NN Simple stripping of the exponent.

```

\__siunitx_number_drop_exponent:nnnnnnn
1001 \cs_new_protected:Npn \__siunitx_number_drop_exponent:NN #1#2

```

```

1002 {
1003   \bool_if:NT \l__siunitx_number_drop_exponent_bool
1004   {
1005     \tl_set:Nx #2
1006     { \exp_after:wN \__siunitx_number_drop_exponent:nnnnnnn #1 }
1007   }
1008 }
1009 \cs_new:Npn \__siunitx_number_drop_exponent:nnnnnnn #1#2#3#4#5#6#7
1010 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} { } { } { 0 } } }

```

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

Simple stripping of the uncertainty.

```

\__siunitx_number_drop_uncertainty:NN
\__siunitx_number_drop_uncertainty:nnnnnnn
1011 \cs_new_protected:Npn \__siunitx_number_drop_uncertainty:NN #1#2
1012 {
1013   \bool_if:NTF \l__siunitx_number_drop_uncertainty_bool
1014   {
1015     \tl_set:Nx #2
1016     { \exp_after:wN \__siunitx_number_drop_uncertainty:nnnnnnn #1 }
1017   }
1018   { \tl_set_eq:NN #2 #1 }
1019 }
1020 }
1021 \cs_new:Npn \__siunitx_number_drop_uncertainty:nnnnnnn #1#2#3#4#5#6#7
1022 { \exp_not:n { {#1} {#2} {#3} {#4} { } {#6} {#7} } }

```

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

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

```

\__siunitx_number_round:NN
\__siunitx_number_round_none:nnnnnnn
1023 \cs_new_protected:Npn \__siunitx_number_round:NN #1#2
1024 {
1025   \tl_set:Nx #2
1026   {
1027     \cs:w
1028     __siunitx_number_round_ \l__siunitx_number_round_mode_tl :nnnnnnn
1029     \exp_after:wN
1030     \cs_end: #1
1031   }
1032 }
1033 \cs_new:Npn \__siunitx_number_round_none:nnnnnnn #1#2#3#4#5#6#7
1034 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }

```

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

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

```

\__siunitx_number_round:nnn
\__siunitx_number_round:fnn
\__siunitx_number_round_auxi:nnnN
\__siunitx_number_round_auxii:nnnN
\__siunitx_number_round_auxiii:nnnN
\__siunitx_number_round_auxiv:nnN
\__siunitx_number_round_auxv:nnN
\__siunitx_number_round_auxvi:nN
\__siunitx_number_round_auxvii:nnN
\__siunitx_number_round_auxviii:nnN
\__siunitx_number_round_final_integer:nnw
\__siunitx_number_round_final_decimal:nnw
\__siunitx_number_round_final_significant:n
\__siunitx_number_round_final_significant:N
\__siunitx_number_round_final_significant:w
\__siunitx_number_round_final_output:nn
\__siunitx_number_round_final_output:ff
\__siunitx_number_round_final:nn
\__siunitx_number_round_final:fn
1035 \cs_new:Npn \__siunitx_number_round:nnn #1#2#3
1036 {
1037   \__siunitx_number_round_auxi:nnnN {#1} {#2} { }
1038   #3 \q_recursion_tail \q_recursion_stop
1039 }

```



```

1040 \cs_generate_variant:Nn \__siunitx_number_round:nnn { f }
1041 \cs_new:Npn \__siunitx_number_round_auxi:nnnN #1#2#3#4
1042 {
1043   \quark_if_recursion_tail_stop_do:Nn #4
1044   {
1045     \__siunitx_number_round_auxii:nnnN {#1} {#3} { } #2
1046     \q_recursion_tail \q_recursion_stop
1047   }
1048   \__siunitx_number_round_auxi:nnnN {#1} {#2} {#4#3}
1049 }
1050 \cs_new:Npn \__siunitx_number_round_auxii:nnnN #1#2#3#4
1051 {
1052   \quark_if_recursion_tail_stop_do:Nn #4
1053   {
1054     \tl_if_blank:nTF {#2}
1055     {
1056       \__siunitx_number_round_auxiv:nnnN {#1} { } { } #3
1057       \q_recursion_tail \q_recursion_stop
1058     }
1059     {
1060       \__siunitx_number_round_auxiii:nnnN {#1} {#3} { } #2
1061       \q_recursion_tail \q_recursion_stop
1062     }
1063   }
1064   \__siunitx_number_round_auxii:nnnN {#1} {#2} {#4#3}
1065 }

```

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. We also have to cover the case where we round to exactly place above the length of the integer: that may product a value of 1...: we tidy up the case where it comes out as 0 later.

```

1066 \cs_new:Npn \__siunitx_number_round_auxiii:nnnN #1#2#3#4
1067 {
1068   \quark_if_recursion_tail_stop_do:Nn #4
1069   {
1070     \__siunitx_number_round_auxiv:nnnN {#1} { } {#3} #2
1071     \q_recursion_tail \q_recursion_stop
1072   }
1073   \int_compare:nNnTF {#1} > 0
1074   {
1075     \exp_args:Nf \__siunitx_number_round_auxiii:nnnN
1076     { \int_eval:n { #1 - 1 } } {#2} { #4#3 }
1077   }
1078   { \__siunitx_number_round_auxv:nnN {#3} {#2} #4 }
1079 }
1080 \cs_new:Npn \__siunitx_number_round_auxiv:nnnN #1#2#3#4
1081 {
1082   \quark_if_recursion_tail_stop_do:Nn #4
1083   {
1084     \int_compare:nNnTF {#1} = 0
1085     {

```

```

1086         \_siunitx_number_round_auxvi:nnN {#3} {#2}
1087         0 \q_recursion_tail \q_recursion_stop
1088     }
1089     { { 0 } { } }
1090 }
1091 \int_compare:nNnTF {#1} > 0
1092 {
1093     \exp_args:Nf \_siunitx_number_round_auxiv:nnnN
1094     { \int_eval:n { #1 - 1 } } { #2 0 } { #4#3 }
1095 }
1096 { \_siunitx_number_round_auxvi:nnN {#3} {#2} #4 }
1097 }

```

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.

```

1098 \cs_new:Npn \_siunitx_number_round_auxv:nnN #1#2#3
1099 {
1100     \quark_if_recursion_tail_stop_do:Nn #3
1101     {
1102         \_siunitx_number_round_auxvi:nnN
1103         {#1} { } #2 \q_recursion_tail \q_recursion_stop
1104     }
1105     \bool_lazy_or:nnTF
1106     { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
1107     {
1108         \bool_lazy_all_p:n
1109         {
1110             { \l_siunitx_number_round_half_even_bool }
1111             { ! \int_if_odd_p:n {#3} }
1112             { \_siunitx_number_round_if_half_p:n {#1} }
1113         }
1114     }
1115     { \_siunitx_number_round_final_decimal:nnw }
1116     { \_siunitx_number_round_auxviii:nnN }
1117     {#2} { } #3
1118 }
1119 \cs_new:Npn \_siunitx_number_round_auxvi:nnN #1#2#3
1120 {
1121     \quark_if_recursion_tail_stop_do:Nn #3
1122     { { 0 } { } }
1123     \bool_lazy_or:nnTF
1124     { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
1125     {
1126         \bool_lazy_all_p:n
1127         {
1128             { \l_siunitx_number_round_half_even_bool }
1129             { ! \int_if_odd_p:n {#3} }
1130             { \_siunitx_number_round_if_half_p:n {#1} }
1131         }
1132     }
1133     { \_siunitx_number_round_final_integer:nnw }
1134     { \_siunitx_number_round_auxviii:nnN }
1135     { } {#2} #3
1136 }

```

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.

```

1137 \cs_new:Npn \__siunitx_number_round_auxvii:nnN #1#2#3
1138 {
1139   \quark_if_recursion_tail_stop_do:Nn #3
1140   {
1141     \str_if_eq:nnTF {#1} { 0 }
1142     {
1143       \__siunitx_number_round_final_output:ff
1144       { 1 }
1145       { \__siunitx_number_round_truncate:n {#2} }
1146     }
1147     {
1148       \__siunitx_number_round_auxviii:nnN {#2} { } #1
1149       \q_recursion_tail \q_recursion_stop
1150     }
1151   }
1152   \int_compare:nNnTF {#3} = 9
1153   { \__siunitx_number_round_auxvii:nnN {#1} { 0 #2 } }
1154   {
1155     \int_compare:nNnTF {#3} = 0
1156     {
1157       \__siunitx_number_round_final_decimal:nnw
1158       {#1} { 1 \__siunitx_number_round_truncate:n {#2} }
1159     }
1160     {
1161       \__siunitx_number_round_final:fn
1162       { \int_eval:n { #3 + 1 } }
1163       { \__siunitx_number_round_final_decimal:nnw {#1} {#2} }
1164     }
1165   }
1166 }
1167 \cs_new:Npn \__siunitx_number_round_auxviii:nnN #1#2#3
1168 {
1169   \quark_if_recursion_tail_stop_do:Nn #3
1170   {
1171     \tl_if_blank:nTF {#1}
1172     {
1173       \__siunitx_number_round_final_shift:ff
1174       {
1175         \exp_last_unbraced:Nf 1
1176         { \__siunitx_number_round_truncate_direct:n {#2} } 0
1177       }
1178       { }
1179     }
1180     {
1181       \__siunitx_number_round_final_shift:ff
1182       { 1 #2 }
1183       { \__siunitx_number_round_truncate:n {#1} }
1184     }

```

```

1185     }
1186     \int_compare:nNnTF {#3} = 9
1187     { \__siunitx_number_round_auxviii:nnN {#1} { 0 #2 } }
1188     {
1189         \__siunitx_number_round_final:fn
1190         { \int_eval:n { #3 + 1 } }
1191         { \__siunitx_number_round_final_integer:nnw {#1} {#2} }
1192     }
1193 }

```

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

```

1194 \cs_new:Npn \__siunitx_number_round_final_decimal:nnw
1195   #1#2#3 \q_recursion_tail \q_recursion_stop
1196   {
1197     \__siunitx_number_round_final_output:ff
1198     { \tl_reverse:n {#1} }
1199     { \tl_reverse:n {#3} #2 }
1200   }
1201 \cs_new:Npn \__siunitx_number_round_final_integer:nnw
1202   #1#2#3 \q_recursion_tail \q_recursion_stop
1203   {
1204     \__siunitx_number_round_final_output:ff
1205     { \exp_args:Ne \__siunitx_number_round_final_significant:n { \tl_reverse:n {#3} #2 } }
1206     {#1}
1207   }
1208 \cs_new:Npn \__siunitx_number_round_final_significant:n #1
1209   {
1210     \__siunitx_number_round_final_significant:N #1
1211     \q_recursion_tail \q_recursion_stop
1212   }
1213 \cs_new:Npn \__siunitx_number_round_final_significant:N #1
1214   {
1215     \quark_if_recursion_tail_stop_do:Nn #1 { 0 }
1216     \int_compare:nNnTF {#1} = 0
1217     { \__siunitx_number_round_final_significant:N }
1218     {
1219       #1
1220       \__siunitx_number_round_final_significant:w
1221     }
1222   }
1223 \cs_new:Npn \__siunitx_number_round_final_significant:w
1224   #1 \q_recursion_tail \q_recursion_stop
1225   {#1}
1226 \cs_new:Npn \__siunitx_number_round_final_output:nn #1#2 { {#1} {#2} }
1227 \cs_generate_variant:Nn \__siunitx_number_round_final_output:nn { ff }
1228 \cs_new:Npn \__siunitx_number_round_final:nn #1#2
1229   { #2 #1 }
1230 \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.

```

1231 \cs_new:Npn \__siunitx_number_round_final_shift:nn #1#2
1232 {
1233   \str_if_eq:VnTF \l__siunitx_number_round_mode_tl { places }
1234   {
1235     \use:c
1236       { __siunitx_number_round_ \l__siunitx_number_exponent_mode_tl :nn }
1237       {#1} {#2}
1238   }
1239   { {#1} {#2} }
1240 }
1241 \cs_generate_variant:Nn \__siunitx_number_round_final_shift:nn { ff }
1242 \cs_new:Npn \__siunitx_number_round_engineering:nn #1#2
1243 {
1244   \int_compare:nNnTF { \tl_count:n {#1} } = 4
1245   {
1246     \__siunitx_number_round_engineering:NNNNn #1 {#2}
1247     \__siunitx_number_round_final_shift:Nw 3
1248   }
1249   { {#1} {#2} }
1250 }
1251 \cs_new:Npn \__siunitx_number_round_engineering:NNNNn #1#2#3#4#5
1252 {
1253   {#1}
1254   \exp_args:NV \__siunitx_number_round_engineering:nnN
1255     { \l__siunitx_number_round_precision_int } { }
1256     #2#3#4#5 \q_recursion_tail \q_recursion_stop
1257 }
1258 \cs_new:Npn \__siunitx_number_round_engineering:nnN #1#2#3
1259 {
1260   \quark_if_recursion_tail_stop_do:Nn #3 { {#2} }
1261   \int_compare:nNnTF {#1} = { 0 }
1262   { \use_i_delimit_by_q_recursion_stop:nw { {#2} } }
1263   { \__siunitx_number_round_engineering:nnN { #1 - 1 } { #2#3 } }
1264 }
1265 \cs_new:Npn \__siunitx_number_round_fixed:nn #1#2 { {#1} {#2} }
1266 \cs_new:Npn \__siunitx_number_round_input:nn #1#2 { {#1} {#2} }
1267 \cs_new:Npn \__siunitx_number_round_scientific:nn #1#2
1268 {
1269   \__siunitx_number_exponent_shift:nnf
1270     { 1 } {#1} { \__siunitx_number_round_truncate_direct:n {#2} }
1271   \__siunitx_number_round_final_shift:Nw 1
1272 }
1273 \cs_new:Npn \__siunitx_number_round_final_shift:Nw #1#2 \__siunitx_number_round_places_end:nn
1274 {
1275   \exp_not:n { {#3} }
1276   \__siunitx_number_exponent_finalise:n { #4#5 + #1 }
1277 }

```

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.

```

1278 \cs_new:Npn \__siunitx_number_round_truncate:n #1
1279 {
1280   \str_if_eq:VnTF \l__siunitx_number_round_mode_tl { figures }

```

```

1281     { \__siunitx_number_round_truncate_direct:n {#1} }
1282     {#1}
1283   }
1284   \cs_new:Npn \__siunitx_number_round_truncate_direct:n #1
1285   {
1286     \__siunitx_number_round_truncate:nnN { } { }
1287     #1 \q_recursion_tail \q_recursion_stop
1288   }
1289   \cs_new:Npn \__siunitx_number_round_truncate:nnN #1#2#3
1290   {
1291     \quark_if_recursion_tail_stop_do:Nn #3 { #1 }
1292     \__siunitx_number_round_truncate:nnN {#1#2} {#3}
1293   }

```

(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.

```

1294   \prg_new_conditional:Npnn \__siunitx_number_round_if_half:n #1 { p }
1295   {
1296     \int_compare:nNnTF { \tl_head:n { #1 0 } } = 5
1297     {
1298       \exp_after:wN \__siunitx_number_round_if_half:N \use_none:n #1 0
1299       \q_recursion_tail \q_recursion_stop
1300     }
1301     { \prg_return_false: }
1302   }
1303   \cs_new:Npn \__siunitx_number_round_if_half:N #1
1304   {
1305     \quark_if_recursion_tail_stop_do:Nn #1
1306     { \prg_return_true: }
1307     \int_compare:nNnTF {#1} = 0
1308     { \__siunitx_number_round_if_half:N }
1309     { \use_i_delimit_by_q_recursion_stop:nw { \prg_return_false: } }
1310   }

```

(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.

```

1311   \cs_new:Npn \__siunitx_number_round_pad:nnn #1#2#3
1312   {
1313     {#2}
1314     {
1315       #3
1316       \bool_if:NT \l__siunitx_number_round_pad_bool
1317       { \prg_replicate:nn {#1} { 0 } }
1318     }
1319   }

```

(End definition for __siunitx_number_round_pad:nnn.)

_siunitx_number_round_figures:nnnnnn
_siunitx_number_round_figures_aux:nnnnnn
_siunitx_number_round_figures_count:nnN
_siunitx_number_round_figures_count:nnN

Rounding to figures only makes sense if the number is not 0, so we start by filtering out that case. We then check that there is no uncertainty, and that the number of figures requested is positive: if not, the result is always fixed at zero.

```

1320 \cs_new:Npn \__siunitx_number_round_figures:nnnnnnn #1#2#3#4#5#6#7
1321 {
1322   \bool_lazy_and:nnTF
1323     { \str_if_eq_p:nn {#3} { 0 } }
1324     {
1325       \str_if_eq_p:ee
1326         { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1327     }
1328     { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1329     { \__siunitx_number_round_figures_aux:nnnnnnn {#1} {#2} {#3} {#4} {#5} {#6} {#7} }
1330 }
1331 \cs_new:Npn \__siunitx_number_round_figures_aux:nnnnnnn #1#2#3#4#5#6#7
1332 {
1333   \tl_if_blank:nTF {#5}
1334   {
1335     \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
1336     {
1337       \exp_not:n { {#1} {#2} }
1338       \__siunitx_number_round_figures_count:nnN {#3} {#4} #3#4
1339       \q_recursion_tail \q_recursion_stop
1340       \exp_not:n { { } {#6} {#7} }
1341     }
1342     { { } { } { } { 0 } { } { } { } { } { 0 } }
1343   }
1344   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1345 }

```

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

```

1346 \cs_new:Npn \__siunitx_number_round_figures_count:nnN #1#2#3
1347 {
1348   \quark_if_recursion_tail_stop_do:Nn #3
1349   { { } { } { } { 0 } { } { } { } { } { 0 } }
1350   \int_compare:nNnTF {#3} = 0
1351   { \__siunitx_number_round_figures_count:nnN {#1} {#2} }
1352   { \__siunitx_number_round_figures_count:nnnN { 1 } {#1} {#2} }
1353 }
1354 \cs_new:Npn \__siunitx_number_round_figures_count:nnnN #1#2#3#4
1355 {
1356   \quark_if_recursion_tail_stop_do:Nn #4
1357   {
1358     \int_compare:nNnTF {#1} > \l__siunitx_number_round_precision_int
1359     {
1360       \__siunitx_number_round:fnn
1361       { \int_eval:n { #1 - \l__siunitx_number_round_precision_int } }
1362       {#2} {#3}
1363     }
1364     {
1365       \__siunitx_number_round_pad:nnn
1366       { \l__siunitx_number_round_precision_int - (#1) } {#2} {#3}
1367     }
1368   }
1369   \exp_args:Nf \__siunitx_number_round_figures_count:nnnN
1370   { \int_eval:n { #1 + 1 } } {#2} {#3}

```

1371 }

(End definition for _siunitx_number_round_figures:nnnnnnn and others.)

_siunitx_number_round_places:nnnnnnn
 _siunitx_number_round_places_end:nnn
 _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.

```

1372 \cs_new:Npn \_siunitx_number_round_places:nnnnnnn #1#2#3#4#5#6#7
1373 {
1374   \tl_if_blank:nTF {#5}
1375   {
1376     \exp_args:Ne \_siunitx_number_round_places_finalise:n
1377     {
1378       \exp_not:n { {#1} {#2} }
1379       \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
1380       { \_siunitx_number_round_places_decimal:nn }
1381       { \_siunitx_number_round_places_integer:nn }
1382       {#3} {#4}
1383       \_siunitx_number_round_places_end:nnn { } {#6} {#7}
1384     }
1385   }
1386   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1387 }
1388 \cs_new:Npn \_siunitx_number_round_places_end:nnn #1#2#3 { \exp_not:n { {#1} {#2} {#3} } }
1389 \cs_new:Npn \_siunitx_number_round_places_decimal:nn #1#2
1390 {
1391   \int_compare:nNnTF
1392   { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} } > 0
1393   {
1394     \_siunitx_number_round_pad:nnn
1395     { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} }
1396     {#1} {#2}
1397   }
1398   {
1399     \_siunitx_number_round:fnn
1400     {
1401       \int_eval:n
1402       { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }
1403     }
1404     {#1} {#2}
1405   }
1406 }
1407 \cs_new:Npn \_siunitx_number_round_places_integer:nn #1#2
1408 {
1409   \_siunitx_number_round:fnn
1410   {
1411     \int_eval:n
1412     { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }
1413   }
1414   {#1} {#2}
1415 }

```

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.

```

1416 \cs_new:Npn \__siunitx_number_round_places_finalise:n #1
1417 { \__siunitx_number_round_places_finalise:nnnnnnn #1 }
1418 \cs_new:Npn \__siunitx_number_round_places_finalise:nnnnnnn #1#2#3#4#5#6#7
1419 {
1420   \bool_lazy_and:nnTF
1421     { \str_if_eq_p:nn {#3} { 0 } }
1422     {
1423       \str_if_eq_p:ee
1424         { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1425     }
1426   {
1427     \tl_if_empty:NTF \l__siunitx_number_round_min_tl
1428     {
1429       \exp_not:n { {#1} }
1430       { \str_if_eq:nnF {#2} { - } { \exp_not:n {#2} } }
1431       \exp_not:n { {#3} {#4} {#5} {#6} {#7} }
1432     }
1433     {
1434       \exp_after:wN \__siunitx_number_round_places_finalise:nnnnn
1435       \l__siunitx_number_round_min_tl {#2} {#6} {#7}
1436     }
1437   }
1438   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1439 }
1440 \cs_new:Npn \__siunitx_number_round_places_finalise:nnnnn #1#2#3#4#5
1441 {
1442   {
1443     \str_if_eq:nnTF {#3} { - }
1444     { > }
1445     { < }
1446   }
1447   \exp_not:n { {#3} {#1} {#2} { } {#4} {#5} }
1448 }

```

(End definition for __siunitx_number_round_places:nnnnnnn and others.)

_siunitx_number_round_uncertainty:nnnnnnn
_siunitx_number_round_uncertainty_end:nn
_siunitx_number_round_uncertainty:nnn
_siunitx_number_round_uncertainty:nnnn
_siunitx_number_round_uncertainty:nnnnn
_siunitx_number_round_uncertainty_simple:nnnnnn
_siunitx_number_round_uncertainty_shift:nnnnnn
_siunitx_number_round_uncertainty_shift:nn
_siunitx_number_round_uncertainty_shift:nnnw
_siunitx_number_round_uncertainty_shift_aux:nnnnn
_siunitx_number_round_uncertainty_engineering:nnn
_siunitx_number_round_uncertainty_fixed:nnn
_siunitx_number_round_uncertainty_input:nnn
_siunitx_number_round_uncertainty_scientific:nnn
_siunitx_number_round_uncertainty_engineering_2:n
_siunitx_number_round_uncertainty_engineering_3:n
_siunitx_number_round_uncertainty_engineering_4:n

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.

```

1449 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnnnn #1#2#3#4#5#6#7
1450 {
1451   \bool_lazy_or:nnTF
1452     { \tl_if_blank_p:n {#5} }
1453     { ! \int_compare_p:nNn \l__siunitx_number_round_precision_int > 0 }
1454     { \exp_not:n { {#1} {#2} {#3} {#4} { } {#6} {#7} } }
1455   {
1456     \str_if_eq:eeTF { \tl_head:n {#5} } { S }
1457     {
1458       \exp_not:n { {#1} {#2} }
1459       \exp_args:Nno \__siunitx_number_round_uncertainty:nnn
1460       {#3} {#4} { \use_ii:nn #5 }
1461       \__siunitx_number_round_uncertainty_end:nn {#6} {#7}
1462     }

```

```

1463         { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1464     }
1465 }
1466 \cs_new:Npn \__siunitx_number_round_uncertainty_end:nn #1#2
1467 { \exp_not:n { {#1} {#2} } }

```

Round the uncertainty first: this is needed to get the number of places correct. Once that is done, it's just a question of working out the digits in the main part.

```

1468 \cs_new:Npn \__siunitx_number_round_uncertainty:nnn #1#2#3
1469 {
1470     \exp_args:Nf \__siunitx_number_round_uncertainty:nnnn
1471     {
1472         \int_eval:n
1473         { \tl_count:n {#3} - \l__siunitx_number_round_precision_int }
1474     }
1475     {#1} {#2} {#3}
1476 }
1477 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnn #1#2#3#4
1478 {
1479     \exp_last_unbraced:Nf \__siunitx_number_round_uncertainty:nnnnn
1480     { \__siunitx_number_round:nnn {#1} { } {#4} }
1481     {#2} {#3} {#1}
1482 }

```

Here, we need to work out how many digits to zero-fill the uncertainty, for the case where it crosses into the integer part. This depends on whether the uncertainty rounded up, which also links to additional treatment. We therefore split paths here.

```

1483 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnn #1#2#3#4#5
1484 {
1485     \tl_if_blank:nTF {#1}
1486     {
1487         \exp_args:Nf \__siunitx_number_round_uncertainty_simple:nnnnn
1488         {
1489             \prg_replicate:nn
1490             {
1491                 \int_max:nn
1492                 { 0 - ( \tl_count:n {#4} - \tl_count:n {#2} - #5 + 1 ) }
1493                 { 0 }
1494             }
1495             { 0 }
1496         }
1497         {#3} {#4} {#2} {#5}
1498     }
1499     { \__siunitx_number_round_uncertainty_shift:nnnnn {#3} {#4} {#1} {#2} {#5} }
1500 }
1501 }

```

The simple case: round and pad out the uncertainty as required.

```

1502 \cs_new:Npn \__siunitx_number_round_uncertainty_simple:nnnnn #1#2#3#4#5
1503 {
1504     \__siunitx_number_round:nnn
1505     {#5}
1506     {#2} {#3}
1507     { { S } { #4 #1 } }
1508 }

```

```

1509 \cs_new:Npn \__siunitx_number_round_uncertainty_shift:nnnnn #1#2#3#4#5
1510 {
1511   \exp_args:Ne \__siunitx_number_round_uncertainty_shift:nn
1512   { \__siunitx_number_round:fnn { \int_eval:n { #5 + 1 } } {#1} {#2} }
1513   {#5}
1514 }

```

If the padding length is the same as the number of integer digits, then we have a pathological case and need to filter out. They show up as they have the either the precision greater than the length of the integer part, or have a value of exactly 10 when the exponent mode is scientific.

```

1515 \cs_new:Npn \__siunitx_number_round_uncertainty_shift:nn #1#2
1516 { \__siunitx_number_round_uncertainty_shift:nnnw #1 {#2} }
1517 \cs_new:Npn \__siunitx_number_round_uncertainty_shift:nnnw
1518 #1#2#3 #4 \__siunitx_number_round_uncertainty_end:nn #5#6
1519 {
1520   #4
1521   \bool_lazy_and:nnTF
1522   { \int_compare_p:nNn {#3} > { \tl_count:n {#1} } }
1523   {
1524     ! \bool_lazy_and_p:nn
1525     { \str_if_eq_p:Vn \l__siunitx_number_exponent_mode_tl { scientific } }
1526     { \str_if_eq_p:nn { #1.#2 } { 10. } }
1527   }
1528   {
1529     \exp_args:Nf \__siunitx_number_round_uncertainty_shift_aux:nnnnn
1530     {
1531       \prg_replicate:nn
1532       {
1533         \tl_if_blank:nTF {#2}
1534         { \int_min:nn {#3} { \tl_count:n {#1} - 1 } }
1535         { 0 }
1536       }
1537       { 0 }
1538     }
1539     {#1} {#2}
1540   }
1541   {
1542     \use:c
1543     { \__siunitx_number_round_uncertainty_ \l__siunitx_number_exponent_mode_tl :nnn }
1544     {#1}
1545   }
1546   {#5} {#6}
1547 }
1548 \cs_new:Npn \__siunitx_number_round_uncertainty_shift_aux:nnnnn #1#2#3#4#5
1549 {
1550   \use:c
1551   { \__siunitx_number_round_ \l__siunitx_number_exponent_mode_tl :nn }
1552   {#2} {#3}
1553   \__siunitx_number_round_places_end:nnn { { S } { 1 #1 } } {#4} {#5}
1554 }

```

With the data available, adjust the output such that the uncertainty is of the right length.

```

1555 \cs_new:Npn \__siunitx_number_round_uncertainty_engineering:nnn #1#2#3

```

```

1556 {
1557   \use:c
1558   {
1559     __siunitx_number_round_uncertainty_
1560     \l__siunitx_number_exponent_mode_tl _
1561     \tl_count:n {#1}
1562     :n
1563   }
1564   {#2#3}
1565 }
1566 \cs_new:cpn { __siunitx_number_round_uncertainty_engineering_2:n } #1
1567 {
1568   { 10 } { } { { S } { 10 } }
1569   \__siunitx_number_exponent_finalise:n {#1}
1570 }
1571 \cs_new:cpn { __siunitx_number_round_uncertainty_engineering_3:n } #1
1572 {
1573   { 100 } { } { { S } { 100 } }
1574   \__siunitx_number_exponent_finalise:n {#1}
1575 }
1576 \cs_new:cpn { __siunitx_number_round_uncertainty_engineering_4:n } #1
1577 {
1578   { 1 } { } { { S } { 1 } }
1579   \__siunitx_number_exponent_finalise:n { #1 + 3 }
1580 }
1581 \cs_new:Npn \__siunitx_number_round_uncertainty_fixed:nnn #1#2#3
1582 {
1583   {#1} { }
1584   { { S } { 1 \prg_replicate:nn { \tl_count:n {#1} - 1 } { 0 } } }
1585   {#2} {#3}
1586 }
1587 \cs_new_eq:NN \__siunitx_number_round_uncertainty_input:nnn
1588 \__siunitx_number_round_uncertainty_fixed:nnn
1589 \cs_new:Npn \__siunitx_number_round_uncertainty_scientific:nnn #1#2#3
1590 {
1591   { 1 } { } { { S } { 1 } }
1592   \__siunitx_number_exponent_finalise:n { #2#3 + \tl_count:n {#1} - 1 }
1593 }

```

(End definition for __siunitx_number_round_uncertainty:nnnnnnn and others.)

Simple stripping of the decimal part if zero.

```

\__siunitx_number_zero_decimal:NN
\__siunitx_number_zero_decimal:nnnnnnn
1594 \cs_new_protected:Npn \__siunitx_number_zero_decimal:NN #1#2
1595 {
1596   \bool_if:NT \l__siunitx_number_drop_zero_decimal_bool
1597   {
1598     \tl_set:Nx #2
1599     { \exp_after:wN \__siunitx_number_zero_decimal:nnnnnnn #1 }
1600   }
1601 }
1602 \cs_new:Npn \__siunitx_number_zero_decimal:nnnnnnn #1#2#3#4#5#6#7
1603 {
1604   \exp_not:n { {#1} {#2} {#3} }
1605   \str_if_eq:eeTF

```

```

1606 { \exp_not:n {#4} }
1607 { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1608 { { } }
1609 { \exp_not:n { {#4} } }
1610 \exp_not:n { {#5} {#6} {#7} }
1611 }

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

```

2.5 Number modification

A simply case of breaking down and rebuilding the number.

```

\siunitx_number_adjust_exponent:nn
\siunitx_number_adjust_exponent:Nn
\__siunitx_number_adjust_exp:nnnnnnnn
\__siunitx_number_adjust_exp:nn
\__siunitx_number_adjust_exp:nNw
1612 \cs_new:Npn \siunitx_number_adjust_exponent:nn #1#2
1613 { \__siunitx_number_adjust_exp:nnnnnnnn #1 {#2} }
1614 \cs_new:Npn \siunitx_number_adjust_exponent:Nn #1#2
1615 {
1616   \tl_if_empty:NF #1
1617   { \exp_args:NV \siunitx_number_adjust_exponent:nn #1 {#2} }
1618 }
1619 \cs_new:Npn \__siunitx_number_adjust_exp:nnnnnnnn #1#2#3#4#5#6#7#8
1620 {
1621   \exp_not:n { {#1} {#2} {#3} {#4} {#5} }
1622   \exp_args:Ne \__siunitx_number_adjust_exp:nn { \fp_eval:n { #6#7 + #8 } } {#6}
1623 }
1624 \cs_new:Npn \__siunitx_number_adjust_exp:nn #1#2
1625 { \__siunitx_number_adjust_exp:nNw {#2} #1 \q_stop }
1626 \cs_new:Npn \__siunitx_number_adjust_exp:nNw #1#2#3 \q_stop
1627 {
1628   \token_if_eq_meaning:NNTF #2 -
1629   { { - } { \exp_not:n {#3} } }
1630   { { \str_if_eq:nnT {#1} { + } { + } } { \exp_not:n {#2#3} } }
1631 }

```

(End definition for \siunitx_number_adjust_exponent:nn and others. These functions are documented on page 39.)

2.6 Outputting parsed numbers

Purely internal for the present.

```

\l__siunitx_number_bracket_close_tl
\l__siunitx_number_bracket_open_tl
1632 \tl_new:N \l__siunitx_number_bracket_close_tl
1633 \tl_new:N \l__siunitx_number_bracket_open_tl
1634 \tl_set:Nn \l__siunitx_number_bracket_open_tl { ( }
1635 \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

```

1636 \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

```

1637 \tl_new:N \l__siunitx_number_output_decimal_tl

```

(End definition for \l_siunitx_number_output_decimal_tl. This variable is documented on page 40.)

Keys producing tokens in the output.

```

\l_siunitx_number_bracket_negative_bool
\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_exp_marker_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

1638 \keys_define:nn { siunitx }
1639 {
1640   bracket-ambiguous-numbers .bool_set:N =
1641     \l_siunitx_number_bracket_ambiguous_bool ,
1642   bracket-negative-numbers .bool_set:N =
1643     \l_siunitx_number_bracket_negative_bool ,
1644   exponent-base .tl_set:N =
1645     \l_siunitx_number_exponent_base_tl ,
1646   exponent-product .tl_set:N =
1647     \l_siunitx_number_exponent_product_tl ,
1648   group-digits .choice: ,
1649   group-digits / all .code:n =
1650     {
1651       \bool_set_true:N \l_siunitx_number_group_decimal_bool
1652       \bool_set_true:N \l_siunitx_number_group_integer_bool
1653     } ,
1654   group-digits / decimal .code:n =
1655     {
1656       \bool_set_true:N \l_siunitx_number_group_decimal_bool
1657       \bool_set_false:N \l_siunitx_number_group_integer_bool
1658     } ,
1659   group-digits / integer .code:n =
1660     {
1661       \bool_set_false:N \l_siunitx_number_group_decimal_bool
1662       \bool_set_true:N \l_siunitx_number_group_integer_bool
1663     } ,
1664   group-digits / none .code:n =
1665     {
1666       \bool_set_false:N \l_siunitx_number_group_decimal_bool
1667       \bool_set_false:N \l_siunitx_number_group_integer_bool
1668     } ,
1669   group-digits .default:n = all ,
1670   group-minimum-digits .int_set:N =
1671     \l_siunitx_number_group_minimum_int ,
1672   group-separator .tl_set:N =
1673     \l_siunitx_number_group_separator_tl ,
1674   negative-color .tl_set:N =
1675     \l_siunitx_number_negative_color_tl ,
1676   output-close-uncertainty .tl_set:N =
1677     \l_siunitx_number_output_uncert_close_tl ,
1678   output-decimal-marker .tl_set:N =
1679     \l_siunitx_number_output_decimal_tl ,
1680   output-exponent-marker .tl_set:N =
1681     \l_siunitx_number_output_exp_marker_tl ,
1682   output-open-uncertainty .tl_set:N =
1683     \l_siunitx_number_output_uncert_open_tl ,
1684   print-implicit-plus .bool_set:N =
1685     \l_siunitx_number_implicit_plus_bool ,
1686   print-unity-mantissa .bool_set:N =
1687     \l_siunitx_number_unity_mantissa_bool ,
1688   print-zero-exponent .bool_set:N =

```

```

1689 \l__siunitx_number_zero_exponent_bool ,
1690 tight-spacing .bool_set:N =
1691 \l__siunitx_number_tight_bool ,
1692 uncertainty-mode .choices:nn =
1693 { compact , compact-marker , full , separate }
1694 { \tl_set_eq:NN \l__siunitx_number_uncert_mode_tl \l_keys_choice_tl } ,
1695 uncertainty-separator .tl_set:N =
1696 \l__siunitx_number_uncert_separator_tl
1697 }
1698 \bool_new:N \l__siunitx_number_group_decimal_bool
1699 \bool_new:N \l__siunitx_number_group_integer_bool
1700 \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
1701 \cs_new:Npn \siunitx_number_output:N #1
1702 { \__siunitx_number_output:Nn #1 { } }
1703 \cs_new:Npn \siunitx_number_output:n #1
1704 { \__siunitx_number_output:nn #1 { } }
1705 \cs_new:Npn \siunitx_number_output:NN #1#2
1706 { \__siunitx_number_output:Nn #1 {#2} }
1707 \cs_new:Npn \siunitx_number_output:nN #1#2
1708 { \__siunitx_number_output:nn #1 {#2} }
1709 \cs_new:Npn \__siunitx_number_output:Nn #1#2
1710 {
1711   \tl_if_empty:NF #1
1712   { \exp_after:wN \__siunitx_number_output:nnnnnnn #1 {#2} }
1713 }
1714 \cs_new:Npn \__siunitx_number_output:nn #1#2
1715 {
1716   \tl_if_empty:NF {#1}
1717   { \__siunitx_number_output:nnnnnnn #1 {#2} }
1718 }
1719 \cs_new:Npn \__siunitx_number_output:nnnnnnn #1#2#3#4#5#6#7#8
1720 {
1721   \__siunitx_number_output_color:n {#2}
1722   \__siunitx_number_output_comparator:nn {#1} {#8}
1723   \__siunitx_number_output_bracket:nn {#5} {#7}
1724   \__siunitx_number_output_sign:nnn {#1} {#2} {#8}
1725   \__siunitx_number_output_integer:nnn {#3} {#4} {#7}
1726   \__siunitx_number_output_decimal:nn {#4} {#8}
1727   \__siunitx_number_output_uncertainty:nnn {#5} {#4} {#8}
1728   \__siunitx_number_output_exponent:nnnn {#6} {#7} { #3 . #4 } {#8}
1729   \__siunitx_number_output_end:
1730 }

```

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

```

1731 \cs_new:Npn \__siunitx_number_output_bracket:nn #1#2
1732 {

```

```

1733 \bool_lazy_all:nT
1734 {
1735   { \str_if_eq_p:Vn \l__siunitx_number_uncert_mode_tl { separate } }
1736   { \l__siunitx_number_bracket_ambiguous_bool }
1737   { ! \tl_if_blank_p:n {#1} }
1738   {
1739     \bool_lazy_or_p:nn
1740     { \l__siunitx_number_zero_exponent_bool }
1741     { ! \str_if_eq_p:nn {#2} { 0 } }
1742   }
1743 }
1744 \__siunitx_number_output_bracket:w
1745 }
1746 \cs_new:Npn \__siunitx_number_output_bracket:w #1 \__siunitx_number_output_exponent:nnnn
1747 {
1748   \exp_not:V \l__siunitx_number_bracket_open_tl
1749   #1
1750   \exp_not:V \l__siunitx_number_bracket_close_tl
1751   \__siunitx_number_output_exponent:nnnn
1752 }

```

As color for negative values applies to the *whole* output, we have to deal with it before anything else.

```

1753 \cs_new:Npn \__siunitx_number_output_color:n #1
1754 {
1755   \bool_lazy_and:nnT
1756   { \str_if_eq_p:nn {#1} { - } }
1757   { ! \tl_if_empty_p:N \l__siunitx_number_negative_color_tl }
1758   { \exp_not:N \color { \exp_not:V \l__siunitx_number_negative_color_tl } }
1759 }

```

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

```

1760 \cs_new:Npn \__siunitx_number_output_comparator:nn #1#2
1761 {
1762   \tl_if_blank:nF {#1}
1763   { \exp_not:n { \mathord {#1} } }
1764   \exp_not:n {#2}
1765 }

```

Formatting signs has to deal with some additional formatting requirements for negative numbers. Making such numbers by 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.

```

1766 \cs_new:Npn \__siunitx_number_output_sign:nnn #1#2#3
1767 {
1768   \tl_if_blank:nTF {#2}
1769   {
1770     \bool_if:NT \l__siunitx_number_implicit_plus_bool
1771     { \__siunitx_number_output_sign:nN {#1} + }
1772   }
1773   {
1774     \str_if_eq:nnTF {#2} { - }
1775     {
1776       \bool_if:NTF \l__siunitx_number_bracket_negative_bool

```



```

1777         { \_siunitx_number_output_sign_brackets:w }
1778         { \_siunitx_number_output_sign:nN {#1} #2 }
1779     }
1780     { \_siunitx_number_output_sign:nN {#1} #2 }
1781 }
1782 \exp_not:n {#3}
1783 }
1784 \cs_new:Npn \_siunitx_number_output_sign:nN #1#2
1785 {
1786     \tl_if_blank:nTF {#1}
1787     { \_siunitx_number_output_sign:N #2 }
1788     { \exp_not:n { \mathord {#2} } }
1789 }
1790 \cs_new:Npn \_siunitx_number_output_sign:N #1
1791 {
1792     \bool_if:NTF \l__siunitx_number_tight_bool
1793     { \exp_not:n { \mathord {#1} } }
1794     { \exp_not:n {#1} }
1795 }
1796 \cs_new:Npn
1797     \_siunitx_number_output_sign_brackets:w #1 \_siunitx_number_output_end:
1798 {
1799     \exp_not:V \l__siunitx_number_bracket_open_tl
1800     #1
1801     \exp_not:V \l__siunitx_number_bracket_close_tl
1802     \_siunitx_number_output_end:
1803 }

```

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.

```

1804 \cs_new:Npn \_siunitx_number_output_integer:nnn #1#2#3
1805 {
1806     \bool_lazy_any:nT
1807     {
1808         { \l__siunitx_number_unity_mantissa_bool }
1809         { ! \str_if_eq_p:nn { #1 . #2 } { 1. } }
1810         {
1811             \bool_lazy_and_p:nn
1812             { \str_if_eq_p:nn {#3} { 0 } }
1813             { ! \l__siunitx_number_zero_exponent_bool }
1814         }
1815     }
1816     { \_siunitx_number_output_digits:nn { integer } {#1} }
1817 }
1818 \cs_new:Npn \_siunitx_number_output_decimal:nn #1#2
1819 {
1820     \exp_not:n {#2}
1821     \tl_if_blank:nF {#1}
1822     {
1823         \str_if_eq:VnTF \l__siunitx_number_output_decimal_tl { , }
1824         { \exp_not:N \mathord }
1825         { \use:n }
1826         { \exp_not:V \l__siunitx_number_output_decimal_tl }

```

```

1827     }
1828     \exp_not:n {#2}
1829     \__siunitx_number_output_digits:nn { decimal } {#1}
1830   }
1831   \cs_generate_variant:Nn \__siunitx_number_output_decimal:nn { f }
1832   \cs_new:Npn \__siunitx_number_output_digits:nn #1#2
1833   {
1834     \bool_if:cTF { l__siunitx_number_group_ #1 _ bool }
1835     {
1836       \int_compare:nNnTF
1837       { \tl_count:n {#2} } < \l__siunitx_number_group_minimum_int
1838       { \exp_not:n {#2} }
1839       { \use:c { __siunitx_number_output_ #1 _aux:n } {#2} }
1840     }
1841     { \exp_not:n {#2} }
1842   }

```

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.

```

1843   \cs_new:Npn \__siunitx_number_output_integer_aux:n #1
1844   {
1845     \use:c
1846     {
1847       __siunitx_number_output_integer_aux_
1848       \int_eval:n { \int_mod:nn { \tl_count:n {#1} } { 3 } }
1849       :n
1850     } {#1}
1851   }
1852   \cs_new:cpn { __siunitx_number_output_integer_aux_0:n } #1
1853   { \__siunitx_number_output_integer_first:nnNN #1 \q_nil }
1854   \cs_new:cpn { __siunitx_number_output_integer_aux_1:n } #1
1855   { \__siunitx_number_output_integer_first:nnNN { } { } #1 \q_nil }
1856   \cs_new:cpn { __siunitx_number_output_integer_aux_2:n } #1
1857   { \__siunitx_number_output_integer_first:nnNN { } #1 \q_nil }
1858   \cs_new:Npn \__siunitx_number_output_integer_first:nnNN #1#2#3#4
1859   {
1860     \exp_not:n {#1#2#3}
1861     \quark_if_nil:NF #4
1862     { \__siunitx_number_output_integer_loop:NNNN #4 }
1863   }
1864   \cs_new:Npn \__siunitx_number_output_integer_loop:NNNN #1#2#3#4
1865   {
1866     \str_if_eq:VnTF \l__siunitx_number_group_separator_tl { , }
1867     { \exp_not:N \mathord }
1868     { \use:n }
1869     { \exp_not:V \l__siunitx_number_group_separator_tl }
1870     \exp_not:n {#1#2#3}
1871     \quark_if_nil:NF #4
1872     { \__siunitx_number_output_integer_loop:NNNN #4 }
1873   }

```

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.

```

1874 \cs_new:Npn \__siunitx_number_output_decimal_aux:n #1
1875 {
1876   \__siunitx_number_output_decimal_loop:NNNN \c_empty_tl
1877   #1 \q_nil \q_nil \q_nil
1878 }
1879 \cs_new:Npn \__siunitx_number_output_decimal_loop:NNNN #1#2#3#4
1880 {
1881   \quark_if_nil:NF #2
1882   {
1883     \exp_not:V #1
1884     \exp_not:n {#2}
1885     \quark_if_nil:NTF #3
1886     { \use_none:n }
1887     {
1888       \exp_not:n {#3}
1889       \quark_if_nil:NTF #4
1890       { \use_none:nn }
1891       {
1892         \exp_not:n {#4}
1893         \__siunitx_number_output_decimal_loop:NNNN
1894         \l__siunitx_number_group_separator_tl
1895       }
1896     }
1897   }
1898 }

```

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.

```

1899 \cs_new:Npn \__siunitx_number_output_uncertainty:nnn #1#2#3
1900 {
1901   \tl_if_blank:NTF {#1}
1902   { \__siunitx_number_output_uncertainty_unaligned:n {#3} }
1903   {
1904     \use:c { \__siunitx_number_output_uncert_ \tl_head:n {#1} :nnnw }
1905     {#2} {#3} #1
1906   }
1907 }
1908 \cs_new:Npn \__siunitx_number_output_uncertainty_unaligned:n #1
1909 { \exp_not:n { #1 #1 #1 #1 } }
1910 \cs_new:Npn \__siunitx_number_output_uncert_S:nnnw #1#2#3#4
1911 {
1912   \str_if_eq:VnTF \l__siunitx_number_uncert_mode_tl { separate }
1913   {
1914     \exp_not:n {#2}
1915     \__siunitx_number_output_sign:N \pm
1916     \exp_not:n {#2}
1917     \__siunitx_number_output_uncert_S_aux:nnn
1918     { \int_eval:n { \tl_count:n {#4} - \tl_count:n {#1} } }
1919     {#4} {#2}

```

```

1920     }
1921     {
1922         \exp_not:V \l__siunitx_number_uncert_separator_tl
1923         \exp_not:V \l__siunitx_number_output_uncert_open_tl
1924         \use:c { __siunitx_number_output_uncert_S_ \l__siunitx_number_uncert_mode_tl :nn } {#}
1925         \exp_not:V \l__siunitx_number_output_uncert_close_tl
1926         \__siunitx_number_output_uncertainty_unaligned:n {#2}
1927     }
1928 }
1929 \cs_new:Npn \__siunitx_number_output_uncert_S_aux:nnn #1#2#3
1930 {
1931     \int_compare:nNnTF {#1} > 0
1932     {
1933         \__siunitx_number_output_uncert_S_aux:fnnw
1934         { \int_eval:n { #1 - 1 } }
1935         {#3}
1936         { }
1937         #2 \q_nil
1938     }
1939     {
1940         0
1941         \__siunitx_number_output_decimal:fn
1942         {
1943             \prg_replicate:nn { \int_abs:n {#1} } { 0 }
1944             #2
1945         }
1946         {#3}
1947     }
1948 }
1949 \cs_generate_variant:Nn \__siunitx_number_output_uncert_S_aux:nnn { f }
1950 \cs_new:Npn \__siunitx_number_output_uncert_S_aux:nnnw #1#2#3#4
1951 {
1952     \quark_if_nil:NF #4
1953     {
1954         \int_compare:nNnTF {#1} = 0
1955         { \__siunitx_number_output_uncert_S_aux:nnw {#3#4} {#2} }
1956         {
1957             \__siunitx_number_output_uncert_S_aux:fnnw
1958             { \int_eval:n { #1 - 1 } }
1959             {#2}
1960             {#3#4}
1961         }
1962     }
1963 }
1964 \cs_generate_variant:Nn \__siunitx_number_output_uncert_S_aux:nnnw { f }
1965 \cs_new:Npn \__siunitx_number_output_uncert_S_aux:nnw #1#2#3 \q_nil
1966 {
1967     \__siunitx_number_output_digits:nn { integer } {#1}
1968     \__siunitx_number_output_decimal:nn {#3} {#2}
1969 }

```

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

```

1970 \cs_new:Npn \__siunitx_number_output_uncert_S_compact:nn #1#2
1971 { \exp_not:n {#2} }
1972 \cs_new:cpn { __siunitx_number_output_uncert_S_compact-marker:nn } #1#2

```

```

1973 {
1974   \bool_lazy_or:nnTF
1975     { \tl_if_blank_p:n {#1} }
1976     { ! \int_compare_p:nNn { \tl_count:n {#2} } > { \tl_count:n {#1} } }
1977     { \__siunitx_number_output_uncert_S_compact:nn }
1978     { \__siunitx_number_output_uncert_S_full:nn }
1979     {#1} {#2}
1980 }
1981 \cs_new:Npn \__siunitx_number_output_uncert_S_full:nn #1#2
1982 {
1983   \__siunitx_number_output_uncert_S_aux:fnn
1984     { \int_eval:n { \tl_count:n {#2} - \tl_count:n {#1} } }
1985     {#2} { }
1986 }

```

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.

```

1987 \cs_new:Npn \__siunitx_number_output_exponent:nnnn #1#2#3#4
1988 {
1989   \exp_not:n {#4}
1990   \bool_lazy_or:nnTF
1991     { \l__siunitx_number_zero_exponent_bool }
1992     { ! \str_if_eq_p:nn {#2} { 0 } }
1993   {
1994     \tl_if_empty:NTF \l__siunitx_number_output_exp_marker_tl
1995       { \__siunitx_number_output_exponent_auxi:nnnn }
1996       { \__siunitx_number_output_exponent_auxii:nnnn }
1997       {#1} {#2} {#3} {#4}
1998   }
1999   { \exp_not:n {#4} }
2000 }
2001 \cs_new:Npn \__siunitx_number_output_exponent_auxi:nnnn #1#2#3#4
2002 {
2003   \bool_lazy_or:nnTF
2004     { \l__siunitx_number_unity_mantissa_bool }
2005     { ! \str_if_eq_p:nn {#3} { 1. } }
2006   {
2007     \bool_if:NTF \l__siunitx_number_tight_bool
2008       { \exp_not:N \mathord }
2009       { \use:n }
2010       { \exp_not:V \l__siunitx_number_exponent_product_tl }
2011     \exp_not:n {#4}
2012   }
2013   { \exp_not:n {#4} }
2014   \exp_not:V \l__siunitx_number_exponent_base_tl
2015   ~
2016   { \__siunitx_number_output_exponent_auxiii:nn {#1} {#2} }
2017 }
2018 \cs_new:Npn \__siunitx_number_output_exponent_auxii:nnnn #1#2#3#4
2019 {

```

```

2020 \exp_not:n {#4}
2021 \exp_not:V \l__siunitx_number_output_exp_marker_tl
2022 \__siunitx_number_output_exponent_auxiii:nn {#1} {#2}
2023 }
2024 \cs_new:Npn \__siunitx_number_output_exponent_auxiii:nn #1#2
2025 {
2026 \tl_if_blank:nTF {#1}
2027 {
2028 \bool_lazy_and:nnT
2029 { \l__siunitx_number_implicit_plus_bool }
2030 { ! \str_if_eq_p:nn {#2} { 0 } }
2031 { \__siunitx_number_output_sign:N + }
2032 }
2033 { \__siunitx_number_output_sign:N #1 }
2034 \__siunitx_number_output_digits:nn { integer } {#2}
2035 }

```

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

```

2036 \cs_new:Npn \__siunitx_number_output_end: { }

```

(End definition for \siunitx_number_output:N and others. These functions are documented on page 39.)

2.7 Miscellaneous tools

\l__siunitx_number_valid_tl The list of valid tokens.

```

2037 \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.

```

2038 \prg_new_protected_conditional:Npnn \siunitx_if_number:n #1
2039 { T , F , TF }
2040 {
2041 \group_begin:
2042 \bool_set_true:N \l__siunitx_number_validate_bool
2043 \bool_set_true:N \l__siunitx_number_parse_bool
2044 \siunitx_number_parse:nN {#1} \l__siunitx_number_parsed_tl
2045 \tl_if_empty:NTF \l__siunitx_number_parsed_tl
2046 {
2047 \group_end:
2048 \prg_return_false:
2049 }
2050 {
2051 \group_end:
2052 \prg_return_true:
2053 }
2054 }

```

(End definition for \siunitx_if_number:nTF. This function is documented on page 40.)

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

```

2055 \prg_new_conditional:Npnn \siunitx_if_number_token:N #1
2056 { p , T , F , TF }
2057 {
2058   \__siunitx_number_token_auxi:NN #1
2059   \l_siunitx_number_input_decimal_tl
2060   \l__siunitx_number_input_uncert_close_tl
2061   \l_siunitx_number_input_comparator_tl
2062   \l__siunitx_number_input_digit_tl
2063   \l_siunitx_number_input_exponent_tl
2064   \l__siunitx_number_input_ignore_tl
2065   \l_siunitx_number_input_uncert_open_tl
2066   \l_siunitx_number_input_sign_tl
2067   \l__siunitx_number_input_uncert_sign_tl
2068   \q_recursion_tail
2069   \q_recursion_stop
2070 }
2071 \cs_new:Npn \__siunitx_number_token_auxi:NN #1#2
2072 {
2073   \quark_if_recursion_tail_stop_do:Nn #2 { \prg_return_false: }
2074   \__siunitx_number_token_auxii:NN #1 #2
2075   \__siunitx_number_token_auxi:NN #1
2076 }
2077 \cs_new:Npn \__siunitx_number_token_auxii:NN #1#2
2078 {
2079   \exp_after:wN \__siunitx_number_token_auxiii:NN \exp_after:wN #1
2080   #2 \q_recursion_tail \q_recursion_stop
2081 }
2082 \cs_new:Npn \__siunitx_number_token_auxiii:NN #1#2
2083 {
2084   \quark_if_recursion_tail_stop:N #2
2085   \str_if_eq:nnT {#1} {#2}
2086   {
2087     \use_i_delimit_by_q_recursion_stop:nw
2088     {
2089       \use_i_delimit_by_q_recursion_stop:nw
2090       { \prg_return_true: }
2091     }
2092   }
2093   \__siunitx_number_token_auxiii:NN #1
2094 }

```

(End definition for `\siunitx_if_number_token:NTF` and others. This function is documented on page [40](#).)

2.8 Messages

```

2095 \msg_new:nnnn { siunitx } { invalid-number }
2096 { Invalid-number~'#1'. }
2097 {
2098   The~input~'#1'~could~not~be~parsed~as~a~number~following~the~
2099   format~defined~in~module~documentation.
2100 }

```

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.

```

2101 \keys_set:nn { siunitx }
2102 {
2103   bracket-ambiguous-numbers = true ,
2104   bracket-negative-numbers = false ,
2105   drop-exponent = false ,
2106   drop-uncertainty = false ,
2107   drop-zero-decimal = false ,
2108   evaluate-expression = false ,
2109   exponent-base = 10 ,
2110   exponent-mode = input ,
2111   exponent-product = \times ,
2112   expression = #1 ,
2113   fixed-exponent = 0 ,
2114   group-digits = all ,
2115   group-minimum-digits = 5 ,
2116   group-separator = \ , , % (
2117   input-close-uncertainty = ) ,
2118   input-comparators = { <=>\approx\ge\geq\gg\le\leq\ll\sim } ,
2119   input-decimal-markers = { . , } ,
2120   input-digits = 0123456789 ,
2121   input-exponent-markers = dDeE ,
2122   input-ignore = \ , ,
2123   input-open-uncertainty = ( , % )
2124   input-signs = +-\mp\pm ,
2125   input-uncertainty-signs = \pm ,
2126   minimum-decimal-digits = 0 ,
2127   minimum-integer-digits = 0 ,
2128   negative-color = , % (
2129   output-close-uncertainty = ) ,
2130   output-decimal-marker = . ,
2131   output-open-uncertainty = ( , % )
2132   parse-numbers = true ,
2133   print-implicit-plus = false ,
2134   print-unity-mantissa = true ,
2135   print-zero-exponent = false ,
2136   retain-explicit-plus = false ,
2137   retain-zero-uncertainty = false ,
2138   round-half = up ,
2139   round-minimum = 0 ,
2140   round-mode = none ,
2141   round-pad = true ,
2142   round-precision = 2 ,
2143   tight-spacing = false ,
2144   uncertainty-mode = compact ,
2145   uncertainty-separator = =
2146 }
2147 \end{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`, `\textpm`, `\texttimes` and `\textcenteredperiod` for printing in text mode
- `amstext`: the `\text` command for printing in text mode

For detection of math mode fonts, as well as `\mathrm`, the existence of `\symoperators` is assumed; other math font commands are not *required* to exist.

```
\siunitx_print_number:n  
\siunitx_print_number:(V|x)  
\siunitx_print_unit:n  
\siunitx_print_unit:(V|x)
```

```
\siunitx_print_number:n {<material>}  
\siunitx_print_unit:n {<material>}
```

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

```
\siunitx_print_match:n  
\siunitx_print_math:n  
\siunitx_print_text:n
```

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

Prints the *<material>* 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.

<hr/> <hr/>	<code>color = <color></code>
	Color to apply to printed output: the latter should be a named color defined for use with <code>\textcolor</code> . The standard setting is empty (no color).
<hr/> <hr/>	<code>mode = 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> .
<hr/> <hr/>	<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).
<hr/> <hr/>	<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> .
<hr/> <hr/>	<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> .
<hr/> <hr/>	<code>reset-math-version = true false</code>
	Switch to determine whether the active <code>\mathversion</code> is reset to <code>normal</code> when printing in math mode. Note that math version is typically used to select <code>\boldmath</code> , though it is also be used by <i>e.g.</i> <code>sansmath</code> . The standard setting is <code>true</code> .
<hr/> <hr/>	<code>reset-text-family = 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> .
<hr/> <hr/>	<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> .
<hr/> <hr/>	<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> .

<hr/> <hr/> text-family-to-math	<p>text-family-to-math = true false</p> <p>Switch to determine if the family of the current text font should be applied (where possible) to printing in math mode. The standard setting is false.</p>
<hr/> <hr/> text-font-command	<p>text-font-command = $\langle cmd \rangle$</p> <p>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.</p>
<hr/> <hr/> text-series-to-math	<p>text-series-to-math = true false</p> <p>Switch to determine if the weight of the current text font should be applied (where possible) to printing in math mode. This is achieved by setting the <code>\mathversion</code>, and so will override <code>reset-math-version</code>. The mappings between text and math weight are set . The standard setting is false.</p>
<hr/> <hr/> unit-color	<p>unit-color = $\langle color \rangle$</p> <p>Color to apply to units in output: the latter should be a named color defined for use with <code>\textcolor</code>. The standard setting is empty (no color).</p>
<hr/> <hr/> unit-mode	<p>unit-mode = match math text</p> <p>Selects which mode (math or text) units are printed in: a choice from the options match, math or text. The option match matches the mode prevailing at the point <code>\siunitx-print...:n</code> is called. The math and text options choose the relevant \TeX mode for printing. The standard setting is math.</p>
<hr/> <hr/> series-version-mapping	<p>series-version-mapping / $\langle weight \rangle$ = $\langle version \rangle$</p> <p>Defines how <code>siunitx</code> maps from text font weight to math font version. The pre-defined weights are those used as-standard by <code>autoinst</code>:</p> <ul style="list-style-type: none"> • ul • el • l • sl • m • sb • b • eb • ub <p>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.</p>

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_tl
```

Scratch space.

```
6 \tl_new:N \l__siunitx_print_tmp_tl
```

(End definition for `\l__siunitx_print_tmp_tl`.)

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_series_bool

7 \tl_new:N \l__siunitx_print_number_mode_tl
8 \tl_new:N \l__siunitx_print_unit_mode_tl
9 \keys_define:nn { siunitx }
10 {
11   color .meta:n =
12     { number-color = #1 , unit-color = #1 } ,
13   mode .meta:n =
14     { number-mode = #1 , unit-mode = #1 } ,
15   number-color .tl_set:N =
16     \l__siunitx_print_number_color_tl ,
17   number-mode .choices:nn =
18     { match , math , text }
19     {
20       \tl_set_eq:NN
21       \l__siunitx_print_number_mode_tl \l_keys_choice_tl
22     } ,
23   propagate-math-font .bool_set:N =
24     \l__siunitx_print_math_font_bool ,
25   reset-math-version .bool_set:N =
26     \l__siunitx_print_math_version_bool ,
27   reset-text-family .bool_set:N =
28     \l__siunitx_print_text_family_bool ,
29   reset-text-series .bool_set:N =
```

```

30     \l__siunitx_print_text_series_bool ,
31     reset-text-shape .bool_set:N =
32     \l__siunitx_print_text_shape_bool ,
33     text-family-to-math .bool_set:N =
34     \l__siunitx_print_math_family_bool ,
35     text-font-command .tl_set:N =
36     \l__siunitx_print_text_font_tl ,
37     text-series-to-math .bool_set:N =
38     \l__siunitx_print_math_series_bool ,
39     unit-color .tl_set:N =
40     \l__siunitx_print_unit_color_tl ,
41     unit-mode .choices:nn =
42     { match , math , text }
43     {
44         \tl_set_eq:NN
45         \l__siunitx_print_unit_mode_tl \l_keys_choice_tl
46     }
47 }

```

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

`\l__siunitx_print_version_ul_tl`

One set of “focussed” options.

`\l__siunitx_print_version_el_tl`

```

48 \keys_define:nn { siunitx / series-version-mapping }
49 {
50     ul . tl_set:N = \l__siunitx_print_version_ul_tl ,
51     el . tl_set:N = \l__siunitx_print_version_el_tl ,
52     l . tl_set:N = \l__siunitx_print_version_l_tl ,
53     sl . tl_set:N = \l__siunitx_print_version_sl_tl ,
54     m . tl_set:N = \l__siunitx_print_version_m_tl ,
55     sb . tl_set:N = \l__siunitx_print_version_sb_tl ,
56     b . tl_set:N = \l__siunitx_print_version_b_tl ,
57     eb . tl_set:N = \l__siunitx_print_version_eb_tl ,
58     ub . tl_set:N = \l__siunitx_print_version_ub_tl
59 }

```

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

`\siunitx_print_number:n`

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

`\siunitx_print_number:V`

`\siunitx_print_number:x`

`\siunitx_print_unit:n`

`\siunitx_print_unit:V`

`\siunitx_print_unit:x`

`__siunitx_print_aux:nn`

```

60 \cs_new_protected:Npn \siunitx_print_number:n #1
61 { \__siunitx_print_aux:nn { number } {#1} }
62 \cs_generate_variant:Nn \siunitx_print_number:n { V , x }
63 \cs_new_protected:Npn \siunitx_print_unit:n #1
64 { \__siunitx_print_aux:nn { unit } {#1} }
65 \cs_generate_variant:Nn \siunitx_print_unit:n { V , x }
66 \cs_new_protected:Npn \__siunitx_print_aux:nn #1#2
67 {
68     \tl_if_empty:cTF { l__siunitx_print_ #1 _color_tl }
69     { \use:n }
70     { \exp_args:Nv \textcolor { l__siunitx_print_ #1 _color_tl } }
71     {
72         \use:c
73         {
74             siunitx_print_

```

```

75         \tl_use:c { l__siunitx_print_ #1 _mode_tl } :n
76     }
77     {#2}
78 }
79 }

```

(End definition for `\siunitx_print_number:n`, `\siunitx_print_unit:n`, and `__siunitx_print_aux:nn`. These functions are documented on page 93.)

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

```

80 \cs_new_protected:Npn \siunitx_print_match:n #1
81 {
82     \mode_if_math:TF
83     { \siunitx_print_math:n {#1} }
84     { \siunitx_print_text:n {#1} }
85 }

```

(End definition for `\siunitx_print_match:n`. This function is documented on page 93.)

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

```

86 \cs_new_protected:Npn \__siunitx_print_replace_font:N #1
87 {
88     \tl_if_empty:NF \l_siunitx_unit_font_tl
89     {
90         \tl_replace_all:NVn #1
91         \l_siunitx_unit_font_tl
92         { \use:n }
93     }
94 }

```

(End definition for `__siunitx_print_replace_font:N`.)

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

```

95 \clist_map_inline:nn { uc , ec , c , sc , sx , x , ex , ux }
96 { \tl_const:cn { c__siunitx_print_series_ #1 _tl } { m } }

```

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

`\c_siunitx_print_series_l_tl` Font widths with one letter.

```

97 \clist_map_inline:nn { l , m , b }
98 { \tl_const:cn { c__siunitx_print_series_ #1 _tl } { #1 } }

```

(End definition for `\c_siunitx_print_series_l_tl`, `\c_siunitx_print_series_m_tl`, and `\c_siunitx_print_series_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.

```

99 \cs_new_protected:Npn \siunitx_print_math:n #1
100 {
101     \bool_lazy_and:nnTF
102     { \l__siunitx_print_math_series_bool }
103     { \str_if_eq_p:Nn \math@version { normal } } }
104 {
105     \tl_set:Nx \l__siunitx_print_tmp_tl
\__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:N
\__siunitx_print_math_aux:w
\__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

```

```

106         { \exp_after:wN \__siunitx_print_extract_series:Nw \f@series ? \q_stop }
107         \tl_if_empty:NTF \l__siunitx_print_tmp_tl
108         { \__siunitx_print_math_auxi:n {#1} }
109         { \__siunitx_print_math_version:Vn \l__siunitx_print_tmp_tl {#1} }
110     }
111     { \__siunitx_print_math_auxi:n {#1} }
112 }

```

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.

```

113 \cs_new:Npn \__siunitx_print_extract_series:Nw #1#2 ? #3 \q_stop
114 {
115     \cs_if_exist:cTF { c__siunitx_print_series_ #1#2 _tl }
116     { \__siunitx_print_convert_series:v { c__siunitx_print_series_ #1#2 _tl } }
117     {
118         \cs_if_exist:cTF { c__siunitx_print_series_ #1 _tl }
119         { \__siunitx_print_convert_series:v { c__siunitx_print_series_ #1 _tl } }
120         { \__siunitx_print_convert_series:n {#1#2} }
121     }
122 }
123 \cs_new:Npn \__siunitx_print_convert_series:n #1
124 { \tl_use:c { l__siunitx_print_version_ #1 _tl } }
125 \cs_generate_variant:Nn \__siunitx_print_convert_series:n { v }
126 \cs_new_protected:Npn \__siunitx_print_math_auxi:n #1
127 {
128     \bool_if:NTF \l__siunitx_print_math_version_bool
129     { \__siunitx_print_math_version:nn { normal } {#1} }
130     { \__siunitx_print_math_auxii:n {#1} }
131 }

```

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.

```

132 \cs_new_protected:Npn \__siunitx_print_math_version:nn #1#2
133 {
134     \str_if_eq:VnTF \math@version { #1 }
135     { \__siunitx_print_math_auxii:n {#2} }
136     {
137         \mode_if_math:TF
138         { \text }
139         { \use:n }
140         {
141             \mathversion {#1}
142             \__siunitx_print_math_auxii:n {#2}
143         }
144     }
145 }
146 \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.

```

147 \cs_new_protected:Npn \__siunitx_print_math_auxii:n #1
148 { \ensuremath { \__siunitx_print_math_auxiii:n {#1} } }
149 \cs_new_protected:Npn \__siunitx_print_math_auxiii:n #1
150 {
151   \bool_if:NTF \l__siunitx_print_math_family_bool
152   {
153     \str_case_e:nnF { \f@family }
154     {
155       { \rmdefault } { \__siunitx_print_math_auxv:n }
156       { \sfdefault } { \__siunitx_print_math_aux:Nn \mathsf }
157       { \ttdefault } { \__siunitx_print_math_aux:Nn \mathtt }
158     }
159     { \__siunitx_print_math_auxiv:n }
160   }
161   { \__siunitx_print_math_auxiv:n }
162   {#1}
163 }

```

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.

```

164 \cs_new_protected:Npn \__siunitx_print_math_auxiv:n #1
165 {
166   \bool_if:NTF \l__siunitx_print_math_font_bool
167   {
168     \__siunitx_print_math_aux:N
169     \mathbf \mathit \mathsf \mathtt
170     \q_recursion_tail \q_recursion_stop
171   }
172   { \__siunitx_print_math_auxv:n }
173   {#1}
174 }
175 \cs_new_protected:Npn \__siunitx_print_math_auxv:n #1
176 {
177   \bool_lazy_or:nnTF
178   { \int_compare_p:nNn \fam = { -1 } }
179   { \int_compare_p:nNn \fam = \symoperators }
180   { \use:n }
181   { \mathrm }
182   {#1}
183 }
184 \cs_new_protected:Npn \__siunitx_print_math_aux:N #1
185 {
186   \quark_if_recursion_tail_stop_do:Nn #1 { \use:n }
187   \exp_after:wN \exp_after:wN \exp_after:wN \__siunitx_print_math_aux:w
188   \cs:w \cs_to_str:N #1 \c_space_tl \cs_end:
189   \use@mathgroup ? { -2 } \q_stop #1
190 }
191 \cs_new_protected:Npn \__siunitx_print_math_aux:w #1 \use@mathgroup #2#3 #4 \q_stop #5
192 {
193   \int_compare:nNnTF \fam = {#3}

```



```

194     { \use_i_delimit_by_q_recursion_stop:nw { \__siunitx_print_math_aux:Nn #5 } }
195     { \__siunitx_print_math_aux:N }
196   }

```

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

```

197 \cs_new_protected:Npx \__siunitx_print_math_aux:Nn #1#2
198 {
199   \group_begin:
200     \tl_set:Nn \exp_not:N \l__siunitx_print_tmp_tl {#2}
201     \__siunitx_print_replace_font:N \exp_not:N \l__siunitx_print_tmp_tl
202     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
203       { \char_generate:nn { '\_ } { 8 } }
204       { \exp_not:N \__siunitx_print_math_sub:n }
205     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
206       { ^ }
207       { \exp_not:N \__siunitx_print_math_super:n }
208     #1 { \exp_not:N \tl_use:N \exp_not:N \l__siunitx_print_tmp_tl }
209   \group_end:
210 }
211 \cs_generate_variant:Nn \__siunitx_print_math_aux:Nn { c }
212 \cs_new_protected:Npx \__siunitx_print_math_sub:n #1
213 {
214   \char_generate:nn { '\_ } { 8 }
215   { \exp_not:N \__siunitx_print_math_script:n {#1} }
216 }
217 \cs_new_protected:Npn \__siunitx_print_math_super:n #1
218 { ^ { \__siunitx_print_math_script:n {#1} } }
219 \cs_new_protected:Npn \__siunitx_print_math_script:n #1
220 {
221   \group_begin:
222     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
223     \__siunitx_print_replace_font:N \l__siunitx_print_tmp_tl
224     \tl_use:N \l__siunitx_print_tmp_tl
225   \group_end:
226 }

```

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

```

227 \AtBeginDocument
228 {
229   \@ifpackageloaded { tex4ht }
230   {
231     \cs_set_protected:Npn \__siunitx_print_math_auxii:n #1
232     {
233       \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
234       \exp_args:NNnx \tl_replace_all:Nnn \l__siunitx_print_tmp_tl
235         { ^ } { \token_to_str:N ^ }
236       \ensuremath { \exp_args:NV \__siunitx_print_math_auxiii:n \l__siunitx_print_tmp_tl }
237     }
238   }
239   { }
240 }

```

(End definition for `\siunitx_print_math:n` and others. This function is documented on page 93.)

```

\siunitx_print_text:n
  \_siunitx_print_text_replace:n
  \_siunitx_print_text_replace:N
  \_siunitx_print_text_replace:Nn
  \_siunitx_print_text_replace:Nnn
  \_siunitx_print_text_replace_frac:n
  \_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:n
  \_siunitx_print_text_scripts_two:n
  \_siunitx_print_text_fraction:Nnn
241 \cs_new_protected:Npn \siunitx_print_text:n #1
242 {
243   \text
244   {
245     \bool_if:NT \l__siunitx_print_text_family_bool
246     { \fontfamily { \familydefault } }
247     \bool_if:NT \l__siunitx_print_text_series_bool
248     { \fontseries { \seriesdefault } }
249     \bool_if:NT \l__siunitx_print_text_shape_bool
250     { \fontshape { \shapedefault } }
251     \bool_lazy_any:nT
252     {
253       { \l__siunitx_print_text_family_bool }
254       { \l__siunitx_print_text_series_bool }
255       { \l__siunitx_print_text_shape_bool }
256     }
257     { \selectfont }
258     \tl_use:N \l__siunitx_print_text_font_tl
259     \exp_args:NnV \tl_if_head_eq_meaning:nNTF {#1} \l_siunitx_unit_fraction_tl
260     { \_siunitx_print_text_fraction:Nnn #1 }
261     { \_siunitx_print_text_replace:n {#1} }
262   }
263 }

```

Typesetting in text mode is easy in font control terms but more tricky in the manipulation of the input. The easy part comes first.

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

```

264 \cs_new_protected:Npn \_siunitx_print_text_replace:n #1
265 {
266   \group_begin:
267   \tl_if_head_eq_meaning:nNTF {#1} \mathchoice
268   { \_siunitx_print_text_replace:Nnnn #1 }
269   {
270     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
271     \_siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
272     \tl_use:N \l__siunitx_print_tmp_tl
273   }
274   \group_end:
275 }
276 \cs_new_protected:Npx \_siunitx_print_text_replace:N #1
277 {
278   \_siunitx_print_replace_font:N #1
279   \exp_not:N \_siunitx_print_text_replace:Nnn #1
280   \exp_not:N \mathord { }
281   \exp_not:N \pm
282   { \exp_not:N \textpm }
283   \exp_not:N \mp
284   { \exp_not:n { \ensuremath { \mp } } } }
285 -
286   { \exp_not:N \textminus }
287   \exp_not:N \times
288   { \exp_not:N \texttimes }
289   \exp_not:N \cdot

```

```

290     { \exp_not:N \textperiodcentered }
291     \char_generate:nm { '\_ } { 8 }
292     { \exp_not:N \__siunitx_print_text_sub:n }
293     ~
294     { \exp_not:N \__siunitx_print_text_super:n }
295     \exp_not:N \q_recursion_tail
296     { ? }
297     \exp_not:N \q_recursion_stop
298   }
299   \cs_new_protected:Npn \__siunitx_print_text_replace:NNn #1#2#3
300   {
301     \quark_if_recursion_tail_stop:N #2
302     \tl_replace_all:Nnn #1 {#2} {#3}
303     \__siunitx_print_text_replace:NNn #1
304   }
305   \cs_new_protected:Npn \__siunitx_print_text_replace:Nnnnn #1#2#3#4#5
306   {
307     \ensuremath
308     {
309       \mathchoice
310       { \__siunitx_print_print_replace_frac:n {#2} }
311       { \__siunitx_print_print_replace_frac:n {#3} }
312       { \__siunitx_print_print_replace_frac:n {#4} }
313       { \__siunitx_print_print_replace_frac:n {#5} }
314     }
315   }

```

Almost the same as the lead-off but here we need to deal with re-inserting a text mode shift.

```

316   \cs_new_protected:Npn \__siunitx_print_print_replace_frac:n #1
317   {
318     \exp_args:NnV \tl_if_head_eq_meaning:nNTF {#1} \l_siunitx_unit_fraction_tl
319     { \__siunitx_print_text_fraction:Nnn #1 }
320     { \mbox { \__siunitx_print_text_replace:n {#1} } }
321   }

```

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

```

322   \sys_if_engine_xetex:T
323   {
324     \AtBeginDocument
325     {
326       \ifpackageloaded { bidi }
327       {
328         \cs_set_protected:Npn \__siunitx_print_text_replace:n #1
329         {
330           \group_begin:
331           \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
332           \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
333           \LRE { \tl_use:N \l__siunitx_print_tmp_tl }
334           \group_end:
335         }
336       }
337     }
338   }

```

339 }

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.

```

340 \cs_new_protected:Npn \__siunitx_print_text_sub:n #1
341 {
342   \__siunitx_print_text_scripts:NnN
343   \textsubscript {#1} \__siunitx_print_text_super:n
344 }
345 \cs_new_protected:Npn \__siunitx_print_text_super:n #1
346 {
347   \__siunitx_print_text_scripts:NnN
348   \textsuperscript {#1} \__siunitx_print_text_sub:n
349 }
350 \cs_new_protected:Npn \__siunitx_print_text_scripts:NnN #1#2#3
351 {
352   \cs_set_protected:Npn \__siunitx_print_text_scripts:
353   {
354     \if_meaning:w \l_peek_token #3
355     \exp_after:wN \__siunitx_print_text_scripts_two:NnNn
356   \else:
357     \exp_after:wN \__siunitx_print_text_scripts_one:Nn
358   \fi:
359     #1 {#2}
360   }
361   \peek_after:Nw \__siunitx_print_text_scripts:
362 }
363 \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.

```

364 \cs_new_protected:Npn \__siunitx_print_text_scripts_one:Nn #1#2
365 {
366   \group_begin:
367   \tl_set:Nn \l__siunitx_print_tmp_tl {#2}
368   \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
369   \exp_args:NNV \group_end:
370   #1 \l__siunitx_print_tmp_tl
371 }
```

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.

```

372 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:NnNn #1#2#3#4
373 {
374   \cs_if_eq:NNTF #1 \textsubscript
375   { \__siunitx_print_text_scripts_two:nn {#4} {#2} }
376   { \__siunitx_print_text_scripts_two:nn {#2} {#4} }
377 }
378 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:nn #1#2
379 {
380   \group_begin:
381   \exp_not:N \m@th
382   \exp_not:N \ensuremath
383   {
```

```

384         ^ { \exp_not:N \__siunitx_print_text_scripts_two:n {#1} }
385         \char_generate:nn { '\_ } { 8 }
386         { \exp_not:N \__siunitx_print_text_scripts_two:n {#2} }
387     }
388     \group_end:
389 }
390 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:n #1
391 {
392     \mbox
393     {
394         \fontsize \sf@size \z@ \selectfont
395         \__siunitx_print_text_scripts_one:Nn \use:n {#1}
396     }
397 }

```

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

```

398 \cs_new_protected:Npn \__siunitx_print_text_fraction:Nnn #1#2#3
399 {
400     \ensuremath
401     {
402         #1
403         { \mbox { \__siunitx_print_text_replace:n {#2} } }
404         { \mbox { \__siunitx_print_text_replace:n {#3} } }
405     }
406 }

```

(End definition for `\siunitx_print_text:n` and others. This function is documented on page [93](#).)

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.

```

407 \keys_set:nn { siunitx }
408 {
409     color                =      ,
410     mode                 = math ,
411     number-color         =      ,
412     number-mode          = math ,
413     propagate-math-font  = false ,
414     reset-math-version   = true  ,
415     reset-text-shape     = true  ,
416     reset-text-series    = true  ,
417     reset-text-family    = true  ,
418     text-family-to-math  = false ,
419     text-font-command    =      ,
420     text-series-to-math  = false ,
421     unit-color           =      ,
422     unit-mode            = math
423 }

```

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

```

424 \keys_set:nn { siunitx / series-version-mapping }
425 {
426     ul = light ,

```

```
427     el = light ,
428     l  = light ,
429     sl = light ,
430     m  = normal ,
431     sb = bold ,
432     b  = bold ,
433     eb = bold ,
434     ub = bold
435 }
436 \end{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
\__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           \tl_if_blank:nTF {#4}
122           { \__siunitx_quantity_parsed_aux:nnn {#1#2} {#3} { } }
123           { \__siunitx_quantity_parsed_aux:nnn {#1#2} {#3} { { } #4 } }
124         }
125         { \siunitx_quantity_print:nV {#1#2#3#4} \l__siunitx_quantity_unit_tl }
126       }
127     }
128   }

```

For the case of a separated uncertainty with repeated units, we print the two parts

independently. The third argument here is the exponent if there is one, with the spacing correct in either case as we only pass the empty group if one is required.

```

129 \cs_new_protected:Npn \__siunitx_quantity_parsed_aux:nnn #1#2#3
130 {
131   \siunitx_quantity_print:nV {#1#3} \l__siunitx_quantity_unit_tl
132   \tl_if_blank:nF {#2}
133     { \siunitx_quantity_print:nV { { } #2#3 } \l__siunitx_quantity_unit_tl }
134 }

```

(End definition for `\siunitx_quantity:nn` and others. This function is documented on page 107.)

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

```

135 \cs_new_protected:Npn \__siunitx_quantity_extract_exp:nNN #1#2#3
136 { \__siunitx_quantity_extract_exp:nnnnnnnNN #1 #2 #3 }
137 \cs_new_protected:Npn \__siunitx_quantity_extract_exp:nnnnnnnNN #1#2#3#4#5#6#7#8#9
138 {
139   \fp_set:Nn #8 {#6#7}
140   \tl_set:Nx #9
141     { {#1} {#2} {#3} {#4} {#5} { } { 0 } }
142 }

```

(End definition for `__siunitx_quantity_extract_exp:nNN` and `__siunitx_quantity_extract_exp:nnnnnnnNN`.)

`\siunitx_quanity_print:nn` 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. The lazy test here is looking for the case where a 1 has been inserted at the start of a format unit *and* we have some other number to print: the 1 is then removed and there is no space inserted.

```

143 \cs_new_protected:Npn \siunitx_quantity_print:nn #1#2
144 {
145   \siunitx_print_number:n {#1}
146   \tl_if_blank:nF {#2}
147   {
148     \bool_lazy_or:nnTF
149       { \tl_if_blank_p:n {#1} }
150       { ! \tl_if_head_eq_charcode_p:nN {#2} { 1 } }
151     {
152       \tl_use:N \l__siunitx_quantity_product_tl
153       \bool_if:NTF \l__siunitx_quantity_break_bool
154         { \penalty \binoppenalty }
155         { \nobreak }
156       \siunitx_print_unit:n {#2}
157     }
158     {
159       \exp_args:No \siunitx_print_unit:n { \use_none:n #2 }
160     }
161   }
162 }
163 \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.

```

164 \keys_set:nn { siunitx }
165 {
166   allow-quantity-breaks      = false ,
167   prefix-mode                 = input ,
168   quantity-product           = \, ,
169   separate-uncertainty-units = bracket
170 }

```

1.4 Adjustments to units

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

```

\__siunitx_quantity_non_latin:n
\__siunitx_quantity_non_latin:nnnn
171 \bool_lazy_or:nnTF
172 { \sys_if_engine luatex_p: }
173 { \sys_if_engine xetex_p: }
174 {
175   \cs_new:Npn \__siunitx_quantity_non_latin:n #1
176     { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
177 }
178 {
179   \cs_new:Npn \__siunitx_quantity_non_latin:n #1
180     {
181       \exp_last_unbraced:Nf \__siunitx_quantity_non_latin:nnnn
182       { \char_to_utfviii_bytes:n {#1} }
183     }
184   \cs_new:Npn \__siunitx_quantity_non_latin:nnnn #1#2#3#4
185     {
186       \exp_after:wN \exp_after:wN \exp_after:wN
187       \exp_not:N \char_generate:nn {#1} { 13 }
188       \exp_after:wN \exp_after:wN \exp_after:wN
189       \exp_not:N \char_generate:nn {#2} { 13 }
190     }
191 }

```

(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.

```

192 \siunitx_declare_unit:Nxn \degree
193 { \__siunitx_quantity_non_latin:n { "00B0 } }
194 { quantity-product = { } }

```

(End definition for `\degree`. This function is documented on page 146.)

```

195 \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     \_siunitx_symbol_deal_with_utf:
```

```
26     \exp_after:wN \exp_after:wN \exp_after:wN
```

```
27     \exp_not:N \char_generate:nn {#1} { 13 }
```

```
28     \char_generate:nn {#2} { 12 }
```

```
29   }
```

```
30 }
```

```
31 \cs_new:Npn \_siunitx_symbol_deal_with_utf: { }
```

(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. Active characters are set to `\scan_stop:` so that the code here gives exactly the tokens (bytes) we want: needed for encodings other than UTF-8.

```

32 \prg_new_protected_conditional:Npnn \_siunitx_symbol_if_replace:Nn #1#2 { T , TF }
33 {
34   \group_begin:
35   \protected@edef \l__siunitx_symbol_tmpa_tl
36     { \exp_not:V \l_siunitx_unit_font_tl { \_siunitx_symbol_non_latin:n {#2} } }
37   \int_step_inline:nnn { "80 } { "FF }
38     { \char_set_active_eq:nN {##1} \scan_stop: }
39   \keys_set:nn { siunitx } { parse-units = false }
40   \siunitx_unit_format:nN {#1} \l__siunitx_symbol_tmpb_tl
41   \str_if_eq:VVTFF \l__siunitx_symbol_tmpa_tl \l__siunitx_symbol_tmpb_tl
42     {
43       \group_end:
44       \prg_return_true:
45     }
46     {
47       \group_end:
48       \prg_return_false:
49     }
50 }

```

(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”.

```

51 \AtBeginDocument
52 {
53   \_siunitx_symbol_if_replace:NnT \arcminute { "02B9 }
54   {
55     \siunitx_declare_unit:Nn \arcminute
56     { \ensuremath { { } ' } }
57   }
58   \_siunitx_symbol_if_replace:NnT \arcsecond { "02BA }
59   {
60     \siunitx_declare_unit:Nn \arcsecond
61     { \ensuremath { { } '' } }
62   }

```

For `\degree`, direct input works in text mode so there is only a need to tidy up for math mode.

```

63   \_siunitx_symbol_if_replace:NnT \degree { "00B0 }
64   {
65     \siunitx_declare_unit:Nxn \degree
66     {
67       \exp_not:N \text
68       {
69         \@ifpackageloaded { inputenc }
70         { \exp_not:N \textdegree }
71         { \_siunitx_symbol_non_latin:n { "00B0 } }

```

```

72         }
73     }
74     { quantity-product = { } }
75 }

```

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

```

76 \group_begin:
77 \tl_set:Nx \l__siunitx_symbol_tmpa_tl { \__siunitx_symbol_non_latin:n { "00B0 } C }
78 \protected@edef \l__siunitx_symbol_tmpa_tl
79 { \exp_not:V \l_siunitx_unit_font_tl { \l__siunitx_symbol_tmpa_tl } }
80 \keys_set:nn { siunitx } { parse-units = false }
81 \siunitx_unit_format:nN { \degreeCelsius } \l__siunitx_symbol_tmpb_tl
82 \str_if_eq:VTF \l__siunitx_symbol_tmpa_tl \l__siunitx_symbol_tmpb_tl
83 {
84     \group_end:
85     \siunitx_declare_unit:Nx \degreeCelsius
86     {
87         \exp_not:N \text
88         {
89             \@ifpackageloaded { inputenc }
90             { \exp_not:N \textdegree }
91             { \__siunitx_symbol_non_latin:n { "00B0 } }
92         }
93         C
94     }
95 }
96 { \group_end: }

```

For `\ohm`, there is a math mode symbol we can use, so there has to be a mode-dependent definition. This doesn't work if the text mode symbol is bust: the `fourier` package puts us in that position.

```

97 \__siunitx_symbol_if_replace:NnT \ohm { "03A9 }
98 {
99     \tl_set:Nx \l__siunitx_symbol_tmp_tl
100     {
101         \cs_if_exist:NTF \upOmega
102         { \exp_not:N \upOmega }
103         { \exp_not:N \Omega }
104     }
105     \siunitx_declare_unit:Nx \ohm
106     {
107         \@ifpackageloaded { fourier }
108         {
109             \exp_not:N \ensuremath
110             { \exp_not:V \l__siunitx_symbol_tmp_tl }
111         }
112         {
113             \exp_not:N \ifmmode
114             \@ifpackageloaded { fontspec }
115             {
116                 \exp_not:N \text
117                 {
118                     \exp_not:N \ensuremath
119                     { \exp_not:V \l__siunitx_symbol_tmp_tl }

```

```

120         }
121     }
122     { \exp_not:V \l__siunitx_symbol_tmp_tl }
123 \exp_not:N \else
124 \exp_not:N \text
125 {
126     \bool_lazy_or:nnTF
127     { \sys_if_engine luatex_p: }
128     { \sys_if_engine xetex_p: }
129     { \__siunitx_symbol_non_latin:n { "03A9 } }
130     { \exp_not:N \textohm }
131 }
132 \exp_not:N \fi
133 }
134 }
135 }

```

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

```

136 \__siunitx_symbol_if_replace:NnT \micro { "03BC }
137 {
138     \siunitx_declare_prefix:Nnx \micro { -6 }
139     {
140         \exp_not:N \text
141         {
142             \bool_lazy_or:nnTF
143             { \sys_if_engine luatex_p: }
144             { \sys_if_engine xetex_p: }
145             { \__siunitx_symbol_non_latin:n { "00B5 } }
146             { \exp_not:N \textmu }
147         }
148     }
149 }
150 }

```

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 (X_YT_EX and LuaT_EX give different outcomes using for example `\textohm`).

```

151 \AtBeginDocument
152 {
153     \@ifpackageloaded { hyperref }
154     {
155         \exp_args:Nx \pdfstringdefDisableCommands
156         {
157             \cs_set_eq:NN \siunitx_print_text:n \exp_not:N \use:n
158             \siunitx_declare_unit:Nn \exp_not:N \ohm
159             { \__siunitx_symbol_non_latin:n { "03A9 } }
160         }
161     }
162     { }
163 }
164 \</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 an 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 none 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>. Notice the additional support for <code>none</code> here. 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
```

```
\siunitx_cell_end:
```

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

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

```
8 {
```

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

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

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

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

```
13 }
```

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

```
15 {
```

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

```
17   {
```

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

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

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

```
21   }
```

```
22 }
```

(End definition for `\siunitx_cell_begin:w` and `\siunitx_cell_end:`. These functions are documented on page [117](#).)

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_token_aux:N
_siunitx_table_collect_relax: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. This has to be an explicit token as we can get the same meaning from \protect.

```

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           { \__siunitx_table_collect_loop: }

```

```

55         \end                { \tabularnewline \end }
56         \relax              { \_siunitx_table_collect_relax:N #1 }
57         \tabularnewline     { \tabularnewline }
58         \siunitx_cell_end: { \siunitx_cell_end: }
59     }
60     { \_siunitx_table_collect_token_aux:N #1 }
61 }
62 \cs_new_protected:Npn \_siunitx_table_collect_token_aux:N #1
63 {
64     \tl_put_right:Nn \l__siunitx_table_collect_tl {#1}
65     \_siunitx_table_collect_loop:
66 }
67 \cs_new_protected:Npn \_siunitx_table_collect_relax:N #1
68 {
69     \str_if_eq:nnTF {#1} { \relax }
70     { \relax }
71     { \_siunitx_table_collect_token_aux:N #1 }
72 }
73 \AtBeginDocument
74 {
75     \@ifpackageloaded { mdwtab }
76     {
77         \cs_set_protected:Npn \_siunitx_table_collect_token:N #1
78         {
79             \_siunitx_table_collect_search:NnF #1
80             {
81                 \@maybe@unskip { \_siunitx_table_collect_loop: }
82                 \tab@setcr      { \_siunitx_table_collect_loop: }
83                 \unskip         { \_siunitx_table_collect_loop: }
84                 \end            { \tabularnewline \end }
85                 \relax          { \_siunitx_table_collect_relax:N #1 }
86                 \tabularnewline { \tabularnewline }
87                 \siunitx_cell_end: { \siunitx_cell_end: }
88             }
89             { \_siunitx_table_collect_token_aux:N #1 }
90         }
91     }
92     { }
93 }
94 \cs_new_protected:Npn \_siunitx_table_collect_search:NnF #1#2#3
95 {
96     \_siunitx_table_collect_search_aux:NNn #1
97     #2
98     #1 {#3}
99     \q_stop
100 }
101 \cs_new_protected:Npn \_siunitx_table_collect_search_aux:NNn #1#2#3
102 {
103     \token_if_eq_meaning:NNTF #1 #2
104     { \use_i_delimit_by_q_stop:nw {#3} }
105     { \_siunitx_table_collect_search_aux:NNn #1 }
106 }

```

(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 107 \tl_new:N \l__siunitx_table_before_tl
\l__siunitx_table_after_tl 108 \tl_new:N \l__siunitx_table_number_tl
109 \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 110 \cs_new_protected:Npn \_siunitx_table_collect_end:
\_siunitx_table_collect_end_aux:n 111 {
\_siunitx_table_collect_end:w 112 \exp_args:NNV \group_end:
113 \_siunitx_table_collect_end:n \l__siunitx_table_collect_tl
114 \exp_args:NV \_siunitx_table_split:nNNN
115 \l__siunitx_table_collect_tl
116 \l__siunitx_table_before_tl
117 \l__siunitx_table_number_tl
118 \l__siunitx_table_after_tl
119 \tl_if_empty:NTF \l__siunitx_table_number_tl
120 { \_siunitx_table_print_text:V \l__siunitx_table_before_tl }
121 {
122 \_siunitx_table_print:VVV
123 \l__siunitx_table_before_tl
124 \l__siunitx_table_number_tl
125 \l__siunitx_table_after_tl
126 }
127 }
```

To cover the use of REVTeX, we need to allow for the insertion of `\array@row@rst` into cell content: that explodes inside `\protected@edef`. We use the classical solution of making locally equal to `\scan_stop:`.

```
128 \cs_new_protected:Npn \_siunitx_table_collect_end:n #1
129 {
130 \str_if_eq:eeTF { \exp_not:n {#1} }
131 { { \_siunitx_table_collect_end_aux:n {#1} } }
132 { \tl_set:Nn }
133 {
134 \cs_if_exist:NT \array@row@rst
135 { \cs_set_eq:NN \array@row@rst \scan_stop: }
136 \protected@edef
137 {
138 \l__siunitx_table_collect_tl {#1}
139 }
140 \cs_new:Npn \_siunitx_table_collect_end_aux:n #1
141 { \exp_after:wN \_siunitx_table_collect_end:w #1 \q_stop }
142 \cs_new:Npn \_siunitx_table_collect_end:w #1 \q_stop
143 { \exp_not:n {#1} }
```

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

```

\_siunitx_table_split:nNNN
  \_siunitx_table_split_loop:NNN
  \_siunitx_table_split_group:NNNn
  \_siunitx_table_split_token:NNNN
144 \cs_new_protected:Npn \_siunitx_table_split:nNNN #1#2#3#4
145 {
146   \tl_clear:N #2
147   \tl_clear:N #3
148   \tl_clear:N #4
149   \_siunitx_table_split_loop:NNN #2#3#4 #1 \q_recursion_tail \q_recursion_stop
150   \_siunitx_table_split_tidy:N #2
151   \_siunitx_table_split_tidy:N #4
152 }
153 \cs_new_protected:Npn \_siunitx_table_split_loop:NNN #1#2#3
154 {
155   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
156   { \_siunitx_table_split_group:NNNn #1#2#3 }
157   { \_siunitx_table_split_token:NNNN #1#2#3 }
158 }
159 \cs_new_protected:Npn \_siunitx_table_split_group:NNNn #1#2#3#4
160 {
161   \tl_if_empty:NTF #2
162   { \tl_put_right:Nn #1 { {#4} } }
163   { \tl_put_right:Nn #3 { {#4} } }
164   \_siunitx_table_split_loop:NNN #1#2#3
165 }
166 \cs_new_protected:Npn \_siunitx_table_split_token:NNNN #1#2#3#4
167 {
168   \quark_if_recursion_tail_stop:N #4
169   \tl_if_empty:NTF \l_siunitx_table_after_tl
170   {
171     \siunitx_if_number_token:NTF #4
172     { \tl_put_right:Nn #2 {#4} }
173     {
174       \tl_if_empty:NTF #2
175       { \tl_put_right:Nn #1 {#4} }
176       { \tl_put_right:Nn #3 {#4} }
177     }
178   }
179   { \tl_put_right:Nn #3 {#4} }
180   \_siunitx_table_split_loop:NNN #1#2#3
181 }

```

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

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).

```

\_siunitx_table_split_tidy:N
\_siunitx_table_split_tidy:Nn
\_siunitx_table_split_tidy:NV
182 \cs_new_protected:Npn \_siunitx_table_split_tidy:N #1
183 {
184   \tl_if_empty:NF #1
185   { \_siunitx_table_split_tidy:NV #1 #1 }
186 }
187 \cs_new_protected:Npn \_siunitx_table_split_tidy:Nn #1#2

```

```

188 {
189   \str_if_eq:onT { \exp_after:wN { \use:n #2 } } {#2}
190   { \tl_set:No #1 { \use:n #2 } }
191 }
192 \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 0pt plus 0.5fill
\kern 0pt
#
\hskip 0pt plus 0.5fill

```

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

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

```

193 \cs_new_protected:Npn \__siunitx_table_skip:n #1
194 {
195   \skip_horizontal:n {#1}
196   \tex_kern:D \c_zero_skip
197 }
(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
198 \dim_new:N \l_siunitx_table_column_width_dim
199 \keys_define:nn { siunitx }
200 {
201   table-column-width .code:n =
202   {
203     \dim_set:Nn \l_siunitx_table_column_width_dim {#1}
204     \dim_compare:nNnT \l_siunitx_table_column_width_dim > \c_zero_dim
205       { \bool_set_true:N \l_siunitx_table_fixed_width_bool }
206   } ,
207   table-fixed-width .bool_set:N =
208   \l_siunitx_table_fixed_width_bool
209 }
(End definition for \l_siunitx_table_column_width_dim and \l_siunitx_table_fixed_width_bool.)

```


`_siunitx_table_align_center:n`
`_siunitx_table_align_left:n`
`_siunitx_table_align_right:n`
`_siunitx_table_align_none:n`
`_siunitx_table_align_auxi:nn`
`_siunitx_table_align_auxii:nn`

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.

```

210 \cs_new_protected:Npn \_siunitx_table_align_center:n #1
211 { \_siunitx_table_align_auxi:nn {#1} { Opt~plus~0.5fill } }
212 \cs_new_protected:Npn \_siunitx_table_align_left:n #1
213 { \_siunitx_table_align_auxi:nn {#1} { Opt } }
214 \cs_new_protected:Npn \_siunitx_table_align_right:n #1
215 { \_siunitx_table_align_auxi:nn {#1} { Opt~plus~1fill } }
216 \cs_new_protected:Npn \_siunitx_table_align_none:n #1
217 {
218   \bool_if:NTF \l__siunitx_table_fixed_width_bool
219     { \hbox_to_wd:nn \l__siunitx_table_column_width_dim }
220     { \use:n }
221     {#1}
222 }
223 \cs_new_protected:Npn \_siunitx_table_align_auxi:nn #1#2
224 {
225   \bool_if:NTF \l__siunitx_table_fixed_width_bool
226     { \hbox_to_wd:nn \l__siunitx_table_column_width_dim }
227     { \use:n }
228     {
229       \_siunitx_table_skip:n {#2}
230       #1
231       \_siunitx_table_skip:n { Opt~plus~1fill - #2 }
232     }
233 }
234 \AtBeginDocument
235 {
236   \@ifpackageloaded { colortbl }
237   {
238     \cs_new_eq:NN
239       \_siunitx_table_align_auxii:nn
240       \_siunitx_table_align_auxi:nn
241     \cs_set_protected:Npn \_siunitx_table_align_auxi:nn #1#2
242     {
243       \_siunitx_table_skip:n{ Opt~plus~-0.5fill }
244       \_siunitx_table_align_auxii:nn {#1} {#2}
245       \_siunitx_table_skip:n { Opt~plus~-0.5fill }
246     }
247   }
248   { }
249 }

```

(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.

```

250 \keys_define:nn { siunitx }
251 {
252   table-text-alignment .choices:nn =
253     { center , left , right , none }
254     { \tl_set:Nn \l__siunitx_table_align_text_tl {#1} } ,
255   }
256 \tl_new:N \l__siunitx_table_align_text_tl

```

(End definition for \l__siunitx_table_align_text_tl.)

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

```

\_siunitx_table_print_text:V
257 \cs_new_protected:Npn \_siunitx_table_print_text:n #1
258 {
259   \bool_set_true:N \l__siunitx_table_text_bool
260   \use:c { \_siunitx_table_align \l__siunitx_table_align_text_tl :n } {#1}
261 }
262 \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
263 \box_new:N \l__siunitx_table_integer_box
264 \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: Primitives renamed.

```

\_siunitx_table_fill:
265 \cs_new_eq:NN \_siunitx_table_fil: \tex_hfil:D
266 \cs_new_eq:NN \_siunitx_table_fill: \tex_hfill:D

```

(End definition for _siunitx_table_fil: and _siunitx_table_fill:.)

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

```

267 \cs_new:Npn \_siunitx_table_cleanup_decimal:w
268   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
269   { #1#2#3#4#5#6#7 }

```

(End definition for _siunitx_table_cleanup_decimal:w.)

_siunitx_table_color_check:N Handle the fact that splitting a number can leave a negative color dangling.

```

\_siunitx_table_color_check:w
\_siunitx_table_color_check:Nnw
270 \cs_new_protected:Npn \_siunitx_table_color_check:N #1
271 { \exp_after:wN \_siunitx_table_color_check:w #1 \q_stop }
272 \cs_new_protected:Npn \_siunitx_table_color_check:w #1 \q_nil #2 \q_stop
273 {
274   \tl_if_head_eq_meaning:nNT {#1} \color
275   { \_siunitx_table_color_check:Nnw #1 \q_stop }
276 }
277 \cs_new_protected:Npn \_siunitx_table_color_check:Nnw #1#2#3 \q_stop
278 { \keys_set:nn { siunitx } { number-color = #2 } }

```

(End definition for _siunitx_table_color_check:N, _siunitx_table_color_check:w, and _siunitx_table_color_check:Nnw.)

`_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. Notice that we need to allow for the width of the decimal marker itself. When not aligning non-numerical material, we put the extra space into the boxes around the number.

```

279 \cs_new_protected:Npn \_siunitx_table_center_marker:
280 {
281   \hbox_set:Nn \l__siunitx_table_tmp_box
282     { \ensuremath { \mathord { \l_siunitx_number_output_decimal_tl } } }
283   \dim_compare:nNnTF
284     { \box_wd:N \l__siunitx_table_integer_box }
285     >
286     {
287       \box_wd:N \l__siunitx_table_decimal_box
288       - \box_wd:N \l__siunitx_table_tmp_box
289     }
290   {
291     \bool_if:NTF \l__siunitx_table_align_after_bool
292     {
293       \_siunitx_table_center_marker_aux:Nnnn \l__siunitx_table_decimal_box
294       {
295         \box_wd:N \l__siunitx_table_integer_box
296         + \box_wd:N \l__siunitx_table_tmp_box
297       }
298     }
299     {
300       \_siunitx_table_center_marker_aux:Nnnn \l__siunitx_table_after_box
301       {
302         \box_wd:N \l__siunitx_table_after_box
303         + \box_wd:N \l__siunitx_table_integer_box
304         - \box_wd:N \l__siunitx_table_decimal_box
305         + \box_wd:N \l__siunitx_table_tmp_box
306       }
307     }
308     { } { \_siunitx_table_fil: }
309   }
310   {
311     \bool_if:NTF \l__siunitx_table_align_before_bool
312     {
313       \_siunitx_table_center_marker_aux:Nnnn \l__siunitx_table_integer_box
314       {
315         \box_wd:N \l__siunitx_table_decimal_box
316         - \box_wd:N \l__siunitx_table_tmp_box
317       }
318     }
319     {
320       \_siunitx_table_center_marker_aux:Nnnn \l__siunitx_table_before_box
321       {
322         \box_wd:N \l__siunitx_table_before_box
323         + \box_wd:N \l__siunitx_table_decimal_box
324         - \box_wd:N \l__siunitx_table_integer_box
325         - \box_wd:N \l__siunitx_table_tmp_box
326       }

```

```

327     }
328     { \_siunitx_table_fil: } { }
329   }
330 }
331 \cs_new_protected:Npn \_siunitx_table_center_marker_aux:Nnnn #1#2#3#4
332 {
333   \hbox_set_to_wd:Nnn #1 {#2}
334   {
335     #3
336     \hbox_unpack:N #1
337     #4
338   }
339 }

```

(End definition for _siunitx_table_center_marker:.)

\l_siunitx_table_auto_round_bool
 \l_siunitx_table_align_mode_tl
 \l_siunitx_table_align_number_tl

Options for tables with defined space.

```

340 \keys_define:nn { siunitx }
341 {
342   table-alignment .meta:n =
343     { table-number-alignment = #1 , table-text-alignment = #1 },
344   table-alignment-mode .choices:nn =
345     { none , format , marker }
346     { \tl_set_eq:NN \l_siunitx_table_align_mode_tl \l_keys_choice_tl } ,
347   table-auto-round .bool_set:N =
348     \l_siunitx_table_auto_round_bool ,
349   table-format .code:n =
350     {
351       \group_begin:
352       \protected@edef \l_siunitx_table_tmp_tl {#1}
353       \exp_args:NNV \group_end:
354       \_siunitx_table_split:nNNN \l_siunitx_table_tmp_tl
355       \l_siunitx_table_before_model_tl
356       \l_siunitx_table_model_tl
357       \l_siunitx_table_after_model_tl
358       \exp_args:NV \_siunitx_table_generate_model:n \l_siunitx_table_model_tl
359       \tl_set:Nn \l_siunitx_table_align_mode_tl { format }
360     } ,
361   table-number-alignment .choices:nn =
362     { center , left , right }
363     { \tl_set_eq:NN \l_siunitx_table_align_number_tl \l_keys_choice_tl }
364   }
365 \tl_new:N \l_siunitx_table_align_mode_tl
366 \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
 \l_siunitx_table_model_tl

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

```

367 \tl_new:N \l_siunitx_table_format_tl
368 \tl_new:N \l_siunitx_table_before_model_tl
369 \tl_new:N \l_siunitx_table_model_tl
370 \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
\__siunitx_table_generate_model:nnnnnnn
\__siunitx_table_generate_model_S:nnw
\__siunitx_table_generate_model_S:nnn

```

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.

```

371 \cs_new_protected:Npn \__siunitx_table_generate_model:n #1
372 {
373   \group_begin:
374     \bool_set_true:N \l_siunitx_number_parse_bool
375     \keys_set:nn { siunitx } { retain-explicit-plus = true }
376     \siunitx_number_parse:nN {#1} \l__siunitx_table_format_tl
377   \exp_args:NNNV \group_end:
378   \tl_set:Nn \l__siunitx_table_format_tl \l__siunitx_table_format_tl
379   \tl_if_empty:NF \l__siunitx_table_format_tl
380   {
381     \exp_after:wN \__siunitx_table_generate_model:nnnnnnn
382     \l__siunitx_table_format_tl
383   }
384 }
385 \cs_new_protected:Npn \__siunitx_table_generate_model:nnnnnnn #1#2#3#4#5#6#7
386 {
387   \tl_set:Nx \l__siunitx_table_model_tl
388   {
389     \exp_not:n { {#1} {#2} }
390     { \prg_replicate:nn {#3} { 8 } }
391     { \prg_replicate:nn { 0 #4 } { 8 } }
392     {
393       \tl_if_blank:NF {#5}
394       {
395         \use:c { __siunitx_table_generate_model_ \tl_head:n {#5} :nnw }
396         {#4} #5
397       }
398     }
399     \exp_not:n { {#6} }
400     {
401       \int_compare:nNnTF {#7} = 0
402       { 0 }
403       { \prg_replicate:nn {#7} { 8 } }
404     }
405   }
406 }
407 \cs_new:Npn \__siunitx_table_generate_model_S:nnw #1#2#3
408 {
409   { S }
410   {
411     \exp_args:Nff \__siunitx_table_generate_model_S:nnn
412     { \tl_count:n {#1} } { \tl_count:n {#3} }
413     {#3}
414   }
415 }
416 \cs_new:Npn \__siunitx_table_generate_model_S:nnn #1#2#3
417 {
418   \prg_replicate:nn

```

```

419 {
420   \int_compare:nNnTF {#2} > {#1}
421   {
422     \str_range:nnn {#3} { 1 } {#1}
423     +
424     \str_range:nnn {#3} { 1 + #1 } {#2}
425   }
426   {#3}
427 }
428 { 8 }
429 }

```

(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`.

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.

```

430 \cs_new_protected:Npn \__siunitx_table_direct_begin:
431 { \__siunitx_table_direct_begin:w }
432 \cs_new_protected:Npn \__siunitx_table_direct_begin:w #1 \ignorespaces
433 {
434   #1
435   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
436   { \__siunitx_table_print_text:n }
437   {
438     \m@th
439     \use:c { __siunitx_table_direct_ \l__siunitx_table_align_mode_tl : }
440   }
441 }
442 \cs_new_protected:Npn \__siunitx_table_direct_end:
443 { \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.

```

444 \cs_new_protected:Npn \__siunitx_table_direct_marker:
445 {
446   \hbox_set:Nn \l__siunitx_table_tmp_box
447   { \ensuremath { \mathord { \l__siunitx_number_output_decimal_tl } } }
448   \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
449   { \box_wd:N \l__siunitx_table_tmp_box }
450   { \__siunitx_table_fil: }
451   \hbox_set:Nw \l__siunitx_table_integer_box
452   \c_math_toggle_token
453   \tl_map_inline:Nn \l__siunitx_number_input_decimal_tl
454   {
455     \char_set_active_eq:NN ##1 \__siunitx_table_direct_marker_switch:
456     \char_set_mathcode:nn { '##1 } { "8000 }
457   }

```

```

458 }
459 \cs_new_protected:Npn \__siunitx_table_direct_marker_switch:
460 {
461     \c_math_toggle_token
462     \hbox_set_end:
463     \hbox_set:Nw \l__siunitx_table_decimal_box
464     \c_math_toggle_token
465     \l_siunitx_number_output_decimal_tl
466 }

```

We set the two alignment booleans here so that a single auxiliary can cover this case as well as the one for centering the marker when also parsing.

```

467 \cs_new_protected:Npn \__siunitx_table_direct_marker_end:
468 {
469     \c_math_toggle_token
470     \hbox_set_end:
471     \bool_set_true:N \l__siunitx_table_align_before_bool
472     \bool_set_true:N \l__siunitx_table_align_after_bool
473     \__siunitx_table_center_marker:
474     \use:c { __siunitx_table_align_ \l__siunitx_table_align_text_tl :n }
475     {
476         \box_use_drop:N \l__siunitx_table_integer_box
477         \box_use_drop:N \l__siunitx_table_decimal_box
478     }
479 }

```

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

```

480 \cs_new_protected:Npn \__siunitx_table_direct_format:
481 {
482     \tl_set:Nx \l__siunitx_table_tmp_tl
483     { \siunitx_number_output:NN \l__siunitx_table_model_tl \q_nil }
484     \exp_after:wN \__siunitx_table_direct_format_aux:w
485     \l__siunitx_table_tmp_tl \q_stop
486 }
487 \cs_new_protected:Npn \__siunitx_table_direct_format_aux:w
488 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_stop
489 {
490     \hbox_set:Nn \l__siunitx_table_tmp_box
491     { \ensuremath { \__siunitx_table_cleanup_decimal:w #4 } }
492     \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
493     { \box_wd:N \l__siunitx_table_tmp_box }
494     { \__siunitx_table_fil: }
495     \hbox_set:Nn \l__siunitx_table_tmp_box { \ensuremath { #1#2#3 } }
496     \hbox_set_to_wd:Nnw \l__siunitx_table_integer_box
497     { \box_wd:N \l__siunitx_table_tmp_box }
498     \c_math_toggle_token
499     \tl_map_inline:Nn \l_siunitx_number_input_decimal_tl
500     {
501         \char_set_active_eq:NN ##1 \__siunitx_table_direct_format_switch:
502         \char_set_mathcode:nn { '##1 } { "8000 }
503     }
504     \__siunitx_table_fill:
505 }
506 \cs_new_protected:Npn \__siunitx_table_direct_format_switch:

```

```

507 {
508   \c_math_toggle_token
509   \hbox_set_end:
510   \hbox_set_to_wd:Nnw \l__siunitx_table_decimal_box
511   { \box_wd:N \l__siunitx_table_decimal_box }
512   \c_math_toggle_token
513   \mathord { \l__siunitx_number_output_decimal_tl }
514 }
515 \cs_new_protected:Npn \__siunitx_table_direct_format_end:
516 {
517   \c_math_toggle_token
518   \__siunitx_table_fil:
519   \hbox_set_end:
520   \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
521   {
522     \box_use_drop:N \l__siunitx_table_integer_box
523     \box_use_drop:N \l__siunitx_table_decimal_box
524   }
525 }

```

No parsing and no alignment is easy.

```

526 \cs_new_protected:Npn \__siunitx_table_direct_none: { \c_math_toggle_token }
527 \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
528 \box_new:N \l__siunitx_table_before_box
529 \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_before_dim` Space reserved for any non-numerical text before the number: as we need to allow for this to be available after setting the integer part, we need to carry it along for a bit.

```

530 \dim_new:N \l__siunitx_table_before_dim

```

(End definition for `\l__siunitx_table_before_dim`.)

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

```

531 \dim_new:N \l__siunitx_table_carry_dim

```

(End definition for `\l__siunitx_table_carry_dim`.)

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

```

\l__siunitx_table_align_exponent_bool
\l__siunitx_table_align_after_bool
\l__siunitx_table_align_before_bool
\l__siunitx_table_align_uncertainty_bool
532 \keys_define:nn { siunitx }
533 {
534   table-align-comparator .bool_set:N =
535     \l__siunitx_table_align_comparator_bool ,
536   table-align-exponent .bool_set:N =
537     \l__siunitx_table_align_exponent_bool ,
538   table-align-text-after .bool_set:N =
539     \l__siunitx_table_align_after_bool ,

```



```

540 table-align-text-before .bool_set:N =
541   \l__siunitx_table_align_before_bool ,
542 table-align-uncertainty .bool_set:N =
543   \l__siunitx_table_align_uncertainty_bool
544 }

```

(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_format:nnnnn
  \__siunitx_table_print_format_auxi:w
  \__siunitx_table_print_format_auxii:w
  \__siunitx_table_print_format_auxiii:w
  \__siunitx_table_print_format_auxiv:w
  \__siunitx_table_print_format_auxv:w
  \__siunitx_table_print_format_auxvi:w
  \__siunitx_table_print_format_auxvii:w
  \__siunitx_table_print_format_box:Nn
  \__siunitx_table_print_format_after:N
  \__siunitx_table_print_none:nnn

```

```

545 \cs_new_protected:Npn \__siunitx_table_print:nnn #1#2#3
546 { \use:c { __siunitx_table_print_ \l__siunitx_table_align_mode_tl :nnn } {#1} {#2} {#3} }
547 \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.

```

548 \cs_new_protected:Npn \__siunitx_table_print_marker:nnn #1#2#3
549 {
550   \hbox_set:Nn \l__siunitx_table_before_box {#1}
551   \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { Opt }
552   {
553     \box_clear:N \l__siunitx_table_before_box
554     #1
555   }
556   \hbox_set:Nn \l__siunitx_table_after_box {#3}
557   \dim_compare:nNnTF
558     { \box_wd:N \l__siunitx_table_after_box }
559     > { \box_wd:N \l__siunitx_table_before_box }
560     {
561       \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
562       { \box_wd:N \l__siunitx_table_after_box }
563       {
564         \__siunitx_table_fil:
565         \hbox_unpack:N \l__siunitx_table_before_box
566       }
567     }
568     {
569       \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
570       { \box_wd:N \l__siunitx_table_before_box }
571       {
572         \hbox_unpack:N \l__siunitx_table_after_box
573         \__siunitx_table_fil:
574       }
575     }
576   \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
577   \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
578   \tl_set:Nx \l__siunitx_table_tmp_tl
579     { \siunitx_number_output:NN \l__siunitx_table_tmp_tl \q_nil }
580   \__siunitx_table_color_check:N \l__siunitx_table_tmp_tl
581   \exp_after:wN \__siunitx_table_print_marker:w

```

```

582 \l__siunitx_table_tmp_tl \q_stop
583 }
584 \cs_new_protected:Npn \__siunitx_table_print_marker:w
585 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_stop
586 {
587   \hbox_set:Nn \l__siunitx_table_integer_box
588   { \siunitx_print_number:n { #1#2#3 } }
589   \hbox_set:Nn \l__siunitx_table_decimal_box
590   {
591     \siunitx_print_number:x
592     { \__siunitx_table_print_marker_aux:w #4 }
593   }
594   \__siunitx_table_center_marker:
595   \use:c { __siunitx_table_align_ \l__siunitx_table_align_text_tl :n }
596   {
597     \box_use_drop:N \l__siunitx_table_before_box
598     \box_use_drop:N \l__siunitx_table_integer_box
599     \box_use_drop:N \l__siunitx_table_decimal_box
600     \box_use_drop:N \l__siunitx_table_after_box
601   }
602 }
603 \cs_new:Npn \__siunitx_table_print_marker_aux:w
604 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
605 {
606   \exp_not:n {#1#2#3#4#5}
607   \tl_if_blank:nT {#1#2#3#4#5} { { } }
608   \exp_not:n {#6#7}
609 }

```

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. We have to carry the amount of space for the non-numerical material before the cell forward: this may end up being enlarged by unused parts of the integer.

```

610 \cs_new_protected:Npn \__siunitx_table_print_format:nnn #1#2#3
611 {
612   \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_before_model_tl }
613   \hbox_set:Nn \l__siunitx_table_before_box {#1}
614   \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { 0pt }
615   {
616     \box_clear:N \l__siunitx_table_before_box
617     #1
618   }
619   \dim_set:Nn \l__siunitx_table_before_dim { \box_wd:N \l__siunitx_table_tmp_box }
620   \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
621   \group_begin:
622     \bool_if:NT \l__siunitx_table_auto_round_bool
623     {
624       \exp_args:Nx \keys_set:nn { siunitx }
625       {
626         round-mode      = places ,
627         round-pad       = true   ,
628         round-precision =

```

```

629         \exp_after:wN \_siunitx_table_print_format:nnnnnn
630         \l__siunitx_table_format_tl
631     }
632 }
633 \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
634 \exp_args:NNNV \group_end:
635 \tl_set:Nn \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
636 \tl_set:Nx \l__siunitx_table_tmp_tl
637 {
638     \siunitx_number_output:NN \l__siunitx_table_model_tl \q_nil
639     \exp_not:N \q_mark
640     \siunitx_number_output:NN \l__siunitx_table_tmp_tl \q_nil
641 }
642 \exp_after:wN \_siunitx_table_print_format_auxi:w
643 \l__siunitx_table_tmp_tl \q_stop
644 \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_after_model_tl }
645 \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
646 { \box_wd:N \l__siunitx_table_tmp_box + \l__siunitx_table_carry_dim }
647 {
648     \bool_if:NT \l__siunitx_table_align_after_bool
649     { \skip_horizontal:n { \l__siunitx_table_carry_dim } }
650     #3
651     \_siunitx_table_fil:
652 }
653 \use:c { __siunitx_table_align \l__siunitx_table_align_number_tl :n }
654 {
655     \box_use_drop:N \l__siunitx_table_before_box
656     \box_use_drop:N \l__siunitx_table_integer_box
657     \box_use_drop:N \l__siunitx_table_decimal_box
658     \box_use_drop:N \l__siunitx_table_after_box
659 }
660 }
661 \cs_new:Npn \_siunitx_table_print_format:nnnnnn #1#2#3#4#5#6#7
662 { 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).

```

663 \cs_new_protected:Npn \_siunitx_table_print_format_auxi:w
664 #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
665 {
666     \_siunitx_table_color_check:w #3 \q_nil \q_stop
667     \_siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1}
668     \bool_if:NTF \l__siunitx_table_align_before_bool
669     {
670         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
671         { \box_wd:N \l__siunitx_table_tmp_box }
672         {
673             \_siunitx_table_fil:
674             \tl_if_blank:nF {#3}
675             { \siunitx_print_number:n {#3} }
676         }
677     }
678 }

```

```

679     \_siunitx_table_print_format_box:Nn \l__siunitx_table_integer_box {#3}
680     \dim_add:Nn \l__siunitx_table_before_dim
681     {
682         \box_wd:N \l__siunitx_table_tmp_box
683         - \box_wd:N \l__siunitx_table_integer_box
684     }
685 }
686 \_siunitx_table_print_format_auxii:w #2 \q_mark #4 \q_stop
687 }

```

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`”.) As the integer part is completed here, we are able to finalise the width of the pre-numeral part, reboxing it to have the correct width and possibly to force a single overfull warning if appropriate.

```

688 \cs_new_protected:Npn \_siunitx_table_print_format_auxii:w
689 #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
690 {
691     \_siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
692     \bool_lazy_and:nnTF
693     { \l__siunitx_table_align_comparator_bool }
694     { \dim_compare_p:nNn { \box_wd:N \l__siunitx_table_integer_box } > { Opt } }
695     {
696         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
697         {
698             \box_wd:N \l__siunitx_table_integer_box
699             + \box_wd:N \l__siunitx_table_tmp_box
700         }
701         {
702             \hbox_unpack:N \l__siunitx_table_integer_box
703             \_siunitx_table_fil:
704             \siunitx_print_number:n {#4#5}
705         }
706     }
707     {
708         \bool_if:NTF \l__siunitx_table_align_before_bool
709         {
710             \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
711             {
712                 \box_wd:N \l__siunitx_table_integer_box
713                 + \box_wd:N \l__siunitx_table_tmp_box
714             }
715             {
716                 \_siunitx_table_fil:
717                 \hbox_unpack:N \l__siunitx_table_integer_box
718                 \siunitx_print_number:n {#4#5}
719             }
720         }
721         {
722             \dim_set:Nn \l__siunitx_table_tmp_dim
723             { \box_wd:N \l__siunitx_table_integer_box }

```

```

724         \hbox_set:Nn \l__siunitx_table_integer_box
725         {
726             \hbox_unpack:N \l__siunitx_table_integer_box
727             \siunitx_print_number:n {#4#5}
728         }
729         \dim_add:Nn \l__siunitx_table_before_dim
730         {
731             + \box_wd:N \l__siunitx_table_tmp_box
732             + \l__siunitx_table_tmp_dim
733             - \box_wd:N \l__siunitx_table_integer_box
734         }
735     }
736 }
737 \hbox_set_to_wd:Nnn \l__siunitx_table_before_box \l__siunitx_table_before_dim
738 {
739     \__siunitx_table_fil:
740     \hbox_unpack:N \l__siunitx_table_before_box
741 }
742 \__siunitx_table_print_format_auxiii:w #3 \q_mark #6 \q_stop
743 }

```

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.

```

744 \cs_new_protected:Npn \__siunitx_table_print_format_auxiii:w
745   #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
746   {
747       \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
748       \__siunitx_table_print_format_box:Nn \l__siunitx_table_decimal_box {#4#5}
749       \dim_set:Nn \l__siunitx_table_carry_dim
750       {
751           \box_wd:N \l__siunitx_table_tmp_box
752           - \box_wd:N \l__siunitx_table_decimal_box
753       }
754       \__siunitx_table_print_format_auxiv:w #3 \q_mark #6 \q_stop
755   }

```

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.

```

756 \cs_new_protected:Npn \__siunitx_table_print_format_auxiv:w
757   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
758   {
759       \tl_if_blank:nTF {#1}
760       { \__siunitx_table_print_format_auxv:w }
761       { \__siunitx_table_print_format_auxvi:w }
762       #1#2 \q_mark #3#4 \q_stop
763   }
764 \cs_new_protected:Npn \__siunitx_table_print_format_auxv:w
765   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
766   #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
767   { \__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.

```

768 \cs_new_protected:Npn \__siunitx_table_print_format_auxvi:w
769   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
770   #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
771   {
772     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2#3 }
773     \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
774     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #5#6#7 }
775     \__siunitx_table_print_format_after:N \l__siunitx_table_align_uncertainty_bool
776     \__siunitx_table_print_format_auxvii:w #4 \q_mark #8 \q_stop
777   }

```

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.

```

778 \cs_new_protected:Npn \__siunitx_table_print_format_auxvii:w
779   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
780   {
781     \tl_if_blank:nF {#2}
782     {
783       \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2 }
784       \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
785       \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #3#4 }
786       \__siunitx_table_print_format_after:N \l__siunitx_table_align_exponent_bool
787     }
788   }

```

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

```

789 \cs_new_protected:Npn \__siunitx_table_print_format_box:Nn #1#2
790   {
791     \hbox_set:Nn #1
792     {
793       \tl_if_blank:nF {#2}
794       { \siunitx_print_number:n {#2} }
795     }
796   }

```

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

```

797 \cs_new_protected:Npn \__siunitx_table_print_format_after:N #1
798   {
799     \bool_if:NTF #1
800     {
801       \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
802       {
803         \box_wd:N \l__siunitx_table_decimal_box
804         + \l__siunitx_table_carry_dim
805         + \box_wd:N \l__siunitx_table_tmp_box
806       }
807       {
808         \hbox_unpack:N \l__siunitx_table_decimal_box
809         \__siunitx_table_fil:
810         \hbox_unpack:N \l__siunitx_table_tmp_box
811       }
812       \dim_set:Nn \l__siunitx_table_carry_dim
813       {

```

```

814         \l__siunitx_table_tmp_dim
815         - \box_wd:N \l__siunitx_table_tmp_box
816     }
817 }
818 {
819     \hbox_set:Nn \l__siunitx_table_decimal_box
820     {
821         \hbox_unpack:N \l__siunitx_table_decimal_box
822         \hbox_unpack:N \l__siunitx_table_tmp_box
823     }
824     \dim_add:Nn \l__siunitx_table_carry_dim
825     {
826         \l__siunitx_table_tmp_dim
827         - \box_wd:N \l__siunitx_table_tmp_box
828     }
829 }
830 }

```

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

```

831 \cs_new_protected:Npn \__siunitx_table_print_none:nnn #1#2#3
832 {
833     \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
834     {
835         #1
836         \siunitx_number_format:nN {#2} \l__siunitx_table_tmp_tl
837         \siunitx_print_number:V \l__siunitx_table_tmp_tl
838         #3
839     }
840 }

```

(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.

```

841 \keys_set:nn { siunitx }
842 {
843     table-align-comparator = true ,
844     table-align-exponent   = true ,
845     table-align-text-after  = true ,
846     table-align-text-before = true ,
847     table-align-uncertainty = true ,
848     table-alignment         = center ,
849     table-auto-round        = false ,
850     table-column-width      = Opt ,
851     table-fixed-width       = false ,
852     table-format            = 2.2 ,
853     table-number-alignment  = center ,
854     table-text-alignment    = center ,
855     table-alignment-mode    = marker
856 }

```

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

857 </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 <unit(s)>`

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

4 Units in (PDF) strings

`\siunitx_unit_pdfstring_context:` `\group_begin:`
`\siunitx_unit_pdfstring_context:`
`<Expansion context> <units>`
`\group_end:`

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

5 Pre-defined symbolic unit components

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

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

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

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

<hr/>	$\langle prefix \rangle \langle unit \rangle \backslash squared$
$\backslash cubed$	$\langle prefix \rangle \langle unit \rangle \backslash cubed$
	Pre-defined unit powers which apply to the preceding $\langle prefix \rangle / \langle unit \rangle$ combination.
<hr/>	
$\backslash per$	$\backslash per \langle prefix \rangle \langle unit \rangle \langle power \rangle$
	Indicates that the next $\langle prefix \rangle / \langle unit \rangle / \langle power \rangle$ combination is reciprocal, <i>i.e.</i> raises it to the power -1 . This symbolic representation may be applied in addition to a $\backslash power$, and will work correctly if the $\backslash power$ itself is negative. In literal mode $\backslash per$ will print a slash (“/”).
<hr/>	
$\backslash cancel$	$\backslash cancel \langle prefix \rangle \langle unit \rangle \langle power \rangle$
	Indicates that the next $\langle prefix \rangle / \langle unit \rangle / \langle power \rangle$ combination should be “cancelled out”. In the parsed output, the entire unit combination will be given as the argument to a function $\backslash cancel$, which is assumed to be available at a higher level. In literal mode, the same higher-level $\backslash cancel$ will be applied to the next token. It is the responsibility of the calling code to provide an appropriate definition for $\backslash cancel$ outside of the scope of the unit parser.
<hr/>	
$\backslash highlight$	$\backslash highlight \{ \langle color \rangle \} \langle prefix \rangle \langle unit \rangle \langle power \rangle$
	Indicates that the next $\langle prefix \rangle / \langle unit \rangle / \langle power \rangle$ combination should be highlighted in the specified $\langle color \rangle$. In the parsed output, the entire unit combination will be given as the argument to a function $\backslash textcolor$, which is assumed to be available at a higher level. In literal mode, the same higher-level $\backslash textcolor$ will be applied to the next token. It is the responsibility of the calling code to provide an appropriate definition for $\backslash textcolor$ outside of the scope of the unit parser.
<hr/>	
$\backslash of$	$\langle prefix \rangle \langle unit \rangle \langle power \rangle \backslash of \{ \langle qualifier \rangle \}$
	Indicates that the $\langle qualifier \rangle$ applies to the current $\langle prefix \rangle / \langle unit \rangle / \langle power \rangle$ combination. In parsed mode, the display of the result will depend upon module options. In literal mode, the $\langle qualifier \rangle$ will be printed in parentheses following the preceding $\langle unit \rangle$ and a full-width space.
<hr/>	
$\backslash raiseto$ $\backslash tothe$	$\backslash raiseto \{ \langle power \rangle \} \langle prefix \rangle \langle unit \rangle$ $\langle prefix \rangle \langle unit \rangle \backslash tothe \{ \langle power \rangle \}$
	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>

sticky-per	<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`.

unit-font-command	<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 144.)

`\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 144.)

`__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 [143](#).)

`\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 143.)

`\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 143.)

`\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 143.)

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 144.)

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 147.)

\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 147.)

\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 147.)

\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 147.)

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 143.)

```
\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 141.)

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]andable 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.

```

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:n {#1}
264 \tl_set:Nn \l__siunitx_unit_separator_tl { / }

```

```

265     \__siunitx_unit_format_literal_auxiii:w
266   }
267 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxiv:n #1
268 {
269     \__siunitx_unit_format_literal_auxv:nw { }
270     #1 \q_recursion_tail \q_recursion_stop
271 }

```

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

```

272 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxv:nw
273   #1#2 \q_recursion_stop
274 {
275     \tl_if_head_is_N_type:nTF {#2}
276     { \__siunitx_unit_format_literal_auxvi:nN }
277     {
278         \tl_if_head_is_group:nTF {#2}
279         { \__siunitx_unit_format_literal_auxix:nn }
280         { \__siunitx_unit_format_literal_auxxx:nw }
281     }
282     {#1} #2 \q_recursion_stop
283 }
284 \cs_new_protected:Npx \__siunitx_unit_format_literal_auxvi:nN #1#2
285 {
286     \exp_not:N \quark_if_recursion_tail_stop_do:Nn #2
287     { \exp_not:N \__siunitx_unit_format_literal_add:n {#1} }
288     \exp_not:N \token_if_eq_meaning:NNTF #2 ^
289     { \exp_not:N \__siunitx_unit_format_literal_super:nn {#1} }
290     {
291         \exp_not:N \token_if_eq_meaning:NNTF
292         #2 \c__siunitx_unit_math_subscript_tl
293         { \exp_not:N \__siunitx_unit_format_literal_sub:nn {#1} }
294         { \exp_not:N \__siunitx_unit_format_literal_auxvii:nN {#1} #2 }
295     }
296 }

```

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

```

297 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxvii:nN #1#2
298 {
299     \str_if_eq:nnTF {#2} { \protect }
300     { \__siunitx_unit_format_literal_auxviii:nN {#1} }
301     { \__siunitx_unit_format_literal_auxv:nw {#1#2} }
302 }
303 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxviii:nN #1#2
304 { \__siunitx_unit_format_literal_auxv:nw { #1 \protect #2 } }
305 \cs_new_protected:Npn \__siunitx_unit_format_literal_super:nn #1#2
306 {
307     \quark_if_recursion_tail_stop:n {#2}
308     \__siunitx_unit_format_literal_add:n {#1}
309     \tl_put_right:Nn \l__siunitx_unit_formatted_tl { ^ {#2} }
310     \__siunitx_unit_format_literal_auxvi:nN { }
311 }

```

```

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

```

(End definition for __siunitx_unit_format_literal:n and others.)

6.7 (PDF) String creation

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

```

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

```

(End definition for \siunitx_unit_pdfstring_context:.. This function is documented on page 144.)

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`.

```

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

```

(End definition for `\l__siunitx_unit_sticky_per_bool`.)

`\l__siunitx_unit_parsed_prop`
`\l__siunitx_unit_per_bool`
`\l__siunitx_unit_position_int`

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

```

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

```

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

`__siunitx_unit_parse:n`

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

```

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

```

```

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

```

(End definition for \l__siunitx_unit_parse:n.)

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

```

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

```

(End definition for \l__siunitx_unit_parse_add:nnnn.)

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

\l__siunitx_unit_parse_power:nnN
\l__siunitx_unit_parse_qualifier:nn
\l__siunitx_unit_parse_special:n

```

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

```

```

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

```

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

```

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

```

(End definition for __siunitx_unit_parse_prefix:Nn and others.)

__siunitx_unit_parse_unit:Nn Parsing units is slightly more involved than the other cases: this is the one place where the tracking value is incremented. If the switch \l__siunitx_unit_per_bool is set true then the current unit is also reciprocal: this can only happen if \l__siunitx_unit_sticky_per_bool is also true, so only one test is required.

```

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

```

```

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

```

(End definition for `__siunitx_unit_parse_unit:Nn.`)

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

```

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

```

(End definition for `__siunitx_unit_parse_per:.`)

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

```

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

```

```

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

```

(End definition for _siunitx_unit_parse_finalise:n.)

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

```

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

```

(End definition for _siunitx_unit_parse_finalise:.)

6.9 Formatting parsed units

\l_siunitx_unit_fraction_tl Set up the options which apply to formatting.

```

\l_siunitx_unit_denominator_bracket_bool 510 \keys_define:nn { siunitx }
    \l_siunitx_unit_forbid_literal_bool 511 {
\l__siunitx_unit_parse_bool 512     bracket-unit-denominator .bool_set:N =
    \l_siunitx_unit_per_symbol_tl 513     \l__siunitx_unit_denominator_bracket_bool ,
    \l_siunitx_unit_qualifier_mode_tl 514     forbid-literal-units .bool_set:N =
    \l_siunitx_unit_qualifier_phrase_tl 515     \l__siunitx_unit_forbid_literal_bool ,
516     fraction-command .tl_set:N =
517     \l_siunitx_unit_fraction_tl ,
518     parse-units .bool_set:N =
519     \l__siunitx_unit_parse_bool ,
520     per-mode .choice: ,
521     per-mode / fraction .code:n =
522     {
523         \bool_set_false:N \l__siunitx_unit_autofrac_bool
524         \bool_set_false:N \l__siunitx_unit_per_symbol_bool
525         \bool_set_true:N \l__siunitx_unit_powers_positive_bool
526         \bool_set_true:N \l__siunitx_unit_two_part_bool
527     } ,
528     per-mode / power .code:n =
529     {
530         \bool_set_false:N \l__siunitx_unit_autofrac_bool
531         \bool_set_false:N \l__siunitx_unit_per_symbol_bool

```



```

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

```

(End definition for \l_siunitx_unit_fraction_tl and others. This variable is documented on page 144.)

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

```
571 \bool_new:N \l__siunitx_unit_bracket_bool
```

(End definition for \l_siunitx_unit_bracket_bool.)

\l_siunitx_unit_bracket_open_tl Abstracted out but currently purely internal.

```

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

```

(End definition for \l__siunitx_unit_bracket_open_tl and \l__siunitx_unit_bracket_close_tl.)

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

576 \bool_new:N \l__siunitx_unit_font_bool

(End definition for \l__siunitx_unit_font_bool.)

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

\l__siunitx_unit_powers_positive_bool

\l__siunitx_unit_per_symbol_bool

\l__siunitx_unit_two_part_bool

577 \bool_new:N \l__siunitx_unit_autofrac_bool

578 \bool_new:N \l__siunitx_unit_per_symbol_bool

579 \bool_new:N \l__siunitx_unit_powers_positive_bool

580 \bool_new:N \l__siunitx_unit_two_part_bool

(End definition for \l__siunitx_unit_autofrac_bool and others.)

\l__siunitx_unit_numerator_bool Indicates that the current unit should go into the numerator when splitting into two parts (fractions or other “sorted” styles).

581 \bool_new:N \l__siunitx_unit_numerator_bool

(End definition for \l__siunitx_unit_numerator_bool.)

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

582 \tl_new:N \l__siunitx_unit_qualifier_mode_tl

(End definition for \l__siunitx_unit_qualifier_mode_tl.)

\l__siunitx_unit_combine_exp_fp For combining an exponent with the first unit.

583 \fp_new:N \l__siunitx_unit_combine_exp_fp

(End definition for \l__siunitx_unit_combine_exp_fp.)

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

584 \bool_new:N \l__siunitx_unit_prefix_exp_bool

(End definition for \l__siunitx_unit_prefix_exp_bool.)

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

585 \fp_new:N \l__siunitx_unit_prefix_fp

(End definition for \l__siunitx_unit_prefix_fp.)

\l__siunitx_unit_multiple_fp For multiplying units.

586 \fp_new:N \l__siunitx_unit_multiple_fp

(End definition for \l__siunitx_unit_multiple_fp.)

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

\l__siunitx_unit_part_tl

587 \tl_new:N \l__siunitx_unit_current_tl

588 \tl_new:N \l__siunitx_unit_part_tl

(End definition for \l__siunitx_unit_current_tl and \l__siunitx_unit_part_tl.)

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

```

589 \tl_new:N \l__siunitx_unit_denominator_tl

(End definition for \l_siunitx_unit_denominator_tl.)

```

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

```

590 \int_new:N \l__siunitx_unit_total_int

(End definition for \l__siunitx_unit_total_int.)

```

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

```

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

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

```

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

```

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

```

(End definition for _siunitx_unit_format_combine_exp:.)

_siunitx_unit_format_multiply: A simple mapping.

```

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

```

(End definition for `_siunitx_unit_format_multiply:.`)

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

```

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

```

(End definition for `_siunitx_unit_format_mass_to_kilogram:.`)

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

```

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

```

(End definition for `_siunitx_unit_format_bracket:N`.)

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

`_siunitx_unit_format_power_aux:wTF`

`_siunitx_unit_format_power_positive:`

`_siunitx_unit_format_power_negative:`

`_siunitx_unit_format_power_negative_aux:w`

`_siunitx_unit_format_power_superscript:`

```

710 \cs_new_protected:Npn \_siunitx_unit_format_power:
711 {

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

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

```

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

```

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

```

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

```

(End definition for __siunitx_unit_format_prefix: and others.)

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

```

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

```



```

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

```

(End definition for _siunitx_unit_format_qualifier: and others.)

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

```

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

```

(End definition for _siunitx_unit_format_special:.)

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

```

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

```

(End definition for _siunitx_unit_format_unit:.)

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

```

876 \cs_new_protected:Npn \_siunitx_unit_format_output:
877 {
878   \_siunitx_unit_format_font:
879   \bool_set_false:N \l__siunitx_unit_bracket_bool
880   \use:c
881   {
882     \_siunitx_unit_format_output_
883     \bool_if:NTF \l__siunitx_unit_numerator_bool
884     { aux: }
885     { denominator: }

```

```

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

```

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

```

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

```

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

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

```

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

```

(End definition for `_siunitx_unit_format_font:`)

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

```

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

```

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

```

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

```

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

```

963 \cs_new_protected:Npn \_siunitx_unit_format_finalise_autofrac:

```

```

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

```

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

```

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

```

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

```

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

```

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

6.10 Non-Latin character support

`_siunitx_unit_non_latin:n` A small amount of code to make it convenient to include non-Latin characters in units without having to directly include them in the sources directly. We only make the first token active as some packages (e.g. kotex) do this.

```

1014 \bool_lazy_or:nnTF
1015   { \sys_if_engine luatex_p: }
1016   { \sys_if_engine xetex_p: }
1017   {
1018     \cs_new:Npn \_siunitx_unit_non_latin:n #1
1019       { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
1020   }
1021   {
1022     \cs_new:Npn \_siunitx_unit_non_latin:n #1
1023       {
1024         \exp_last_unbraced:Nf \_siunitx_unit_non_latin:nnnn
1025         { \char_to_utfviii_bytes:n {#1} }
1026       }
1027     \cs_new:Npn \_siunitx_unit_non_latin:nnnn #1#2#3#4
1028       {
1029         \exp_after:wN \exp_after:wN \exp_after:wN
1030         \exp_not:N \char_generate:nn {#1} { 13 }
1031         \char_generate:nn {#2} { 12 }
1032       }
1033   }

```

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

6.11 Pre-defined unit components

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

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

\metre `\meter`

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

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

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

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

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

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

`\siunitx_declare_unit:Nn \kelvin { K }`

`\siunitx_declare_unit:Nn \candela { cd }`

(End definition for `\kilogram` and others. These functions are documented on page 145.)

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

`\siunitx_declare_unit:Nn \gram { g }`

(End definition for `\gram`. This function is documented on page 145.)

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

```

1043 \siunitx_declare_prefix:Nnn \yocto { -24 } { y }
1044 \siunitx_declare_prefix:Nnn \zepto { -21 } { z }
1045 \siunitx_declare_prefix:Nnn \atto { -18 } { a }
1046 \siunitx_declare_prefix:Nnn \femto { -15 } { f }
1047 \siunitx_declare_prefix:Nnn \pico { -12 } { p }
1048 \siunitx_declare_prefix:Nnn \nano { -9 } { n }
1049 \siunitx_declare_prefix:Nnn \micro { -6 } { \_siunitx_unit_non_latin:n { "03BC } }
1050 \siunitx_declare_prefix:Nnn \milli { -3 } { m }
1051 \siunitx_declare_prefix:Nnn \centi { -2 } { c }
1052 \siunitx_declare_prefix:Nnn \deci { -1 } { d }

```

(End definition for `\yocto` and others. These functions are documented on page 145.)

`\deca` Now the large ones.

```

1053 \siunitx_declare_prefix:Nnn \deca { 1 } { da }
1054 \siunitx_declare_prefix:Nnn \hecto { 1 } { da }
1055 \siunitx_declare_prefix:Nnn \kilo { 2 } { h }
1056 \siunitx_declare_prefix:Nnn \mega { 3 } { k }
1057 \siunitx_declare_prefix:Nnn \giga { 6 } { M }
1058 \siunitx_declare_prefix:Nnn \tera { 9 } { G }
1059 \siunitx_declare_prefix:Nnn \peta { 12 } { T }
1060 \siunitx_declare_prefix:Nnn \peta { 15 } { P }
1061 \siunitx_declare_prefix:Nnn \exa { 18 } { E }
1062 \siunitx_declare_prefix:Nnn \zetta { 21 } { Z }
1063 \siunitx_declare_prefix:Nnn \yotta { 24 } { Y }

```

(End definition for `\deca` and others. These functions are documented on page 145.)

`\becquerel` Named derived units: first half of alphabet.

```

1064 \siunitx_declare_unit:Nn \becquerel { Bq }
1065 \siunitx_declare_unit:Nx \degreeCelsius { \_siunitx_unit_non_latin:n { "00B0 } C }
1066 \siunitx_declare_unit:Nn \coulomb { C }
1067 \siunitx_declare_unit:Nn \farad { F }
1068 \siunitx_declare_unit:Nn \gray { Gy }
1069 \siunitx_declare_unit:Nn \hertz { Hz }
1070 \siunitx_declare_unit:Nn \henry { H }
1071 \siunitx_declare_unit:Nn \joule { J }
1072 \siunitx_declare_unit:Nn \katal { kat }
1073 \siunitx_declare_unit:Nn \lumen { lm }
1074 \siunitx_declare_unit:Nn \lux { lx }

```

(End definition for `\becquerel` and others. These functions are documented on page 146.)

`\newton` Named derived units: second half of alphabet.

```

1075 \siunitx_declare_unit:Nn \newton { N }
1076 \siunitx_declare_unit:Nx \ohm { \_siunitx_unit_non_latin:n { "03A9 } }
1077 \siunitx_declare_unit:Nn \pascal { Pa }
1078 \siunitx_declare_unit:Nn \radian { rad }
1079 \siunitx_declare_unit:Nn \siemens { S }
1080 \siunitx_declare_unit:Nn \sievert { Sv }
1081 \siunitx_declare_unit:Nn \steradian { sr }
1082 \siunitx_declare_unit:Nn \tesla { T }
1083 \siunitx_declare_unit:Nn \volt { V }
1084 \siunitx_declare_unit:Nn \watt { W }
1085 \siunitx_declare_unit:Nn \weber { Wb }

```

(End definition for `\newton` and others. These functions are documented on page 146.)

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

```

1086 \siunitx_declare_unit:Nn \astronomicalunit { au }
1087 \siunitx_declare_unit:Nn \bel { B }
1088 \siunitx_declare_unit:Nn \decibel { \deci \bel }
1089 \siunitx_declare_unit:Nn \dalton { Da }
1090 \siunitx_declare_unit:Nn \day { d }
1091 \siunitx_declare_unit:Nn \electronvolt { eV }
1092 \siunitx_declare_unit:Nn \hectare { ha }
1093 \siunitx_declare_unit:Nn \hour { h }
1094 \siunitx_declare_unit:Nn \litre { L }
1095 \siunitx_declare_unit:Nn \liter { \litre }
1096 \siunitx_declare_unit:Nn \minute { min }
1097 \siunitx_declare_unit:Nn \neper { Np }
1098 \siunitx_declare_unit:Nn \tonne { t }

```

(End definition for `\astronomicalunit` and others. These functions are documented on page 146.)

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

```

1099 \siunitx_declare_unit:Nx \arcminute { \_siunitx_unit_non_latin:n { "02B9 } }
1100 \siunitx_declare_unit:Nx \arcsecond { \_siunitx_unit_non_latin:n { "02BA } }
1101 \siunitx_declare_unit:Nx \degree { \_siunitx_unit_non_latin:n { "00B0 } }

```

(End definition for `\arcminute`, `\arcsecond`, and `\degree`. These functions are documented on page 146.)

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

```
1102 \siunitx_declare_unit:Nx \percent { \cs_to_str:N \% }
```

(End definition for `\percent`. This function is documented on page 146.)

`\square` Basic powers.

```

1103 \siunitx_declare_power:NNn \square \squared { 2 }
1104 \siunitx_declare_power:NNn \cubic \cubed { 3 }

```

(End definition for `\square` and others. These functions are documented on page 146.)

6.12 Messages

```

1105 \msg_new:nnnn { siunitx } { unit / dangling-part }
1106 { Found~#1~part~with~no~unit. }
1107 {
1108   Each~#1~part~must~be~associated~with~a~unit:~a~#1~part~was~found~
1109   but~no~following~unit~was~given.
1110 }
1111 \msg_new:nnnn { siunitx } { unit / duplicate-part }
1112 { Duplicate~#1~part:~#2. }
1113 {
1114   Each~unit~may~have~only~one~#1:\\
1115   the~additional~#1~part~'~#2'~will~be~ignored.
1116 }
1117 \msg_new:nnnn { siunitx } { unit / duplicate-sticky-per }
1118 { Duplicate~\token_to_str:N \per. }

```

```

1119 {
1120   When-the-'sticky-per'-option-is-active,~only-one~
1121   \token_to_str:N \per \ may-appear-in-a-unit.
1122 }
1123 \msg_new:nnnn { siunitx } { unit / literal }
1124 { Literal-units-disabled. }
1125 {
1126   You-gave-the-literal-input-'#1'~
1127   but-literal-unit-output-is-disabled.
1128 }
1129 \msg_new:nnnn { siunitx } { unit / non-convertible-exponent }
1130 { Exponent~'#1'~cannot-be-converted-into-a-symbolic-prefix. }
1131 {
1132   The-exponent~'#1'~does-not-match-with-any-of-the-symbolic-prefixes~
1133   set-up.
1134 }
1135 \msg_new:nnnn { siunitx } { unit / non-numeric-exponent }
1136 { Prefix~'#1'~does-not-have-a-numerical-value. }
1137 {
1138   The-prefix~'#1'~needs-to-be-combined-with-a-number,~but-it-has-no
1139   numerical-value.
1140 }
1141 \msg_new:nnnn { siunitx } { unit / part-before-unit }
1142 { Found~#1~part-before-first-unit:~#2. }
1143 {
1144   The~#1~part~'#2'~must-follow-after-a-unit:~
1145   it~cannot~appear-before-any-units-and-will~therefore-be-ignored.
1146 }

```

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.

```

1147 \keys_set:nn { siunitx }
1148 {
1149   bracket-unit-denominator = true      ,
1150   forbid-literal-units     = false     ,
1151   fraction-command         = \frac     ,
1152   inter-unit-product       = \,        ,
1153   extract-mass-in-kilograms = true      ,
1154   parse-units              = true      ,
1155   per-mode                 = power      ,
1156   per-symbol               = /          ,
1157   qualifier-mode           = subscript ,
1158   qualifier-phrase         =           ,
1159   sticky-per               = false      ,
1160   unit-font-command        = \mathrm
1161 }

```

Cover the case where the default font is sanserif.

```

1162 \AtBeginDocument
1163 {
1164   \str_if_eq:eeT
1165   { \exp_not:o { \familydefault } }
1166   { \exp_not:n { \sfdefault } }

```



```

1167 { \keys_set:nn { siunitx } { unit-font-command = \mathsf } }
1168 }
1169 \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 182.)

```
\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 182.)

\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 183.)

\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 183.)

\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 183.)

\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 182.)

\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 183.)

\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 182.)

\K Temperatures.

```

68 \siunitx_declare_unit:Nn \K { \kelvin }

```

(End definition for \K. This function is documented on page 182.)

\dB

```

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

```

(End definition for \dB. This function is documented on page 184.)

\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 184.)

\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 183.)

`\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 184.)

`\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 184.)

`\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 182.)

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

```
\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 188.)

```
\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 188.)

```
12 \ranglepackage\rangle
```


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.

```

37 \cs_new_protected:Npn \siunitx_command_create:
38 {
39   \bool_if:NT \l__siunitx_command_create_bool
40   { \__siunitx_command_create: }

```

At the beginning of table cells and inside x-type expansion, all symbolic units need to have *some* definition.

```

41   \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
42   {
43     \cs_if_free:NT ##1
44     { \cs_set_protected:Npn ##1 { \ERROR } }
45   }

```

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.

```

46   \@ifpackageloaded { soulpos }
47   {
48     \@ifpackageloaded { soul }
49     { }
50     {
51       \cs_undefine:N \hl
52       \cs_undefine:N \ul
53     }
54   }
55   { }
56 }
57 \AtBeginDocument { \siunitx_command_create: }
58 \cs_new_protected:Npn \__siunitx_command_create:
59 {
60   \bool_if:NT \l__siunitx_command_xspace_bool
61   { \RequirePackage { xspace } }
62   \bool_if:NT \l__siunitx_command_overwrite_bool
63   {
64     \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
65     { \cs_undefine:N ##1 }
66   }
67   \cs_set_protected:Npx \__siunitx_command_create:N ##1
68   {
69     \ProvideDocumentCommand ##1 { \bool_if:NT \l__siunitx_command_optarg_bool { o } }
70     {
71       \mode_leave_vertical:
72       \group_begin:
73       \bool_if:NTF \l__siunitx_command_optarg_bool
74       { \exp_not:N \IfNoValueTF {###1} }
75       { \use_i:nn }
76       {
77         \siunitx_unit_options_apply:n {##1}
78         \siunitx_unit_format:nN {##1}

```

```

79         \exp_not:N \l__siunitx_command_tmp_tl
80         \bool_if:NTF \l__siunitx_command_prespace_bool
81         { \siunitx_quantity_print:nV { } }
82         { \siunitx_print_unit:V }
83         \exp_not:N \l__siunitx_command_tmp_tl
84     }
85     { \siunitx_quantity:nn {###1} {##1} }
86 \group_end:
87 \bool_if:NT \l__siunitx_command_xspace_bool { \exp_not:N \xspace }
88 }
89 }
90 \seq_map_function:NN \l_siunitx_unit_seq \__siunitx_command_create:N
91 }
92 \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 [189](#).)

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.

```

93 \keys_set:nn { siunitx }
94 {
95     free-standing-units    = false ,
96     overwrite-commands    = false ,
97     space-before-unit     = false ,
98     unit-optional-argument = false ,
99     use-xspace             = false
100 }
101 \</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
29 {
30     abbreviations      ,
31     binary-units       ,
32     load-configurations ,
33     version-1-compatibility
34 }
35 {
36     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
37 }
```

1.2 Angle options

All straight-forward emulation.

```
38 \keys_define:nn { siunitx }
39 {
40     add-arc-degree-zero .code:n =
41     {
42         \__siunitx_option_deprecated:nnV
43         { add-arc-degree-zero }
44         { fill-angle-degrees }
45         \l_keys_value_tl
46     } ,
47     add-arc-degree-zero .default:n = true ,
48     add-arc-minute-zero .code:n =
49     {
50         \__siunitx_option_deprecated:nnV
51         { add-arc-minute-zero }
52         { fill-angle-minutes }
53         \l_keys_value_tl
54     } ,
55     add-arc-minute-zero .default:n = true ,
56     add-arc-second-zero .code:n =
57     {
58         \__siunitx_option_deprecated:nnV
59         { add-arc-second-zero }
60         { fill-angle-seconds }
61         \l_keys_value_tl
62     } ,
63     add-arc-second-zero .default:n = true ,
64     arc-separator .code:n =
65     {
66         \__siunitx_option_deprecated:nnV
67         { arc-separator }
68         { angle-separator }
69         \l_keys_value_tl
70     }
71 }
```

1.3 Combination functions options

```
72 \keys_define:nn { siunitx }
73 {
```

```

74 list-units / brackets .code:n =
75 {
76   \_siunitx_option_deprecated:nn
77   { list-units~==brackets }
78   { list-units~==bracket }
79 } ,
80 range-units / brackets .code:n =
81 {
82   \_siunitx_option_deprecated:nn
83   { range-units~==brackets }
84   { range-units~==bracket }
85 } ,
86 product-units / brackets .code:n =
87 {
88   \_siunitx_option_deprecated:nn
89   { product-units~==brackets }
90   { product-units~==bracket }
91 }
92 }

```

1.4 Command options

```

93 \keys_define:nn { siunitx }
94 {
95   overwrite-functions .code:n =
96   {
97     \_siunitx_option_deprecated:nnV
98     { overwrite-functions }
99     { overwrite-commands }
100     \l_keys_value_tl
101   } ,
102   overwrite-functions .default:n = true
103 }

```

1.5 Print options

```

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

```

```

124         {
125             reset-text-family~==false , ~
126             text-family-to-math~==true
127         }
128     } ,
129     detect-mode .code:n =
130     {
131         \__siunitx_option_deprecated:nn
132         { detect-mode }
133         { mode~==match }
134     } ,
135     detect-none .code:n =
136     {
137         \__siunitx_option_deprecated:nn
138         { detect-none }
139         {
140             mode~==math , ~
141             propagate-math-font~==false , ~
142             reset-math-version~==true , ~
143             reset-text-family~==true , ~
144             reset-text-series~==true , ~
145             text-family-to-math~==false , ~
146             text-series-to-math~==false
147         }
148     } ,
149     detect-shape .code:n =
150     {
151         \__siunitx_option_deprecated:nn
152         { detect-shape }
153         { reset-text-shape~==false }
154     } ,
155     detect-weight .code:n =
156     {
157         \__siunitx_option_deprecated:nn
158         { detect-weight }
159         {
160             reset-text-series~==false , ~
161             text-series-to-math~==true
162         }
163     }
164 }
165 \clist_map_inline:nn
166 {
167     detect-display-math ,
168     detect-inline-family ,
169     detect-inline-weight
170 }
171 {
172     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
173 }

```

The old font insertion options.

```

174 \clist_map_inline:nn
175 {
176     math-rm

```



```

177     math-sf      ,
178     math-tt      ,
179     number-math-rm ,
180     number-math-sf ,
181     number-math-tt ,
182     number-text-rm ,
183     number-text-sf ,
184     number-text-tt ,
185     text-rm      ,
186     text-sf      ,
187     text-tt      ,
188     unit-math-rm  ,
189     unit-math-sf  ,
190     unit-math-tt  ,
191     unit-text-rm  ,
192     unit-text-sf  ,
193     unit-text-tt
194 }
195 {
196     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
197 }

```

1.6 Symbol options

```

198 \clist_map_inline:nn
199 {
200     math-angstrom ,
201     math-arcminute ,
202     math-arcsecond ,
203     math-celsius   ,
204     math-degree    ,
205     math-micro     ,
206     math-ohm       ,
207     text-angstrom  ,
208     text-arcminute ,
209     text-arcsecond ,
210     text-celsius   ,
211     text-degree    ,
212     text-micro     ,
213     text-ohm       ,
214 }
215 {
216     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
217 }

```

1.7 Number options

```

218 \keys_define:nn { siunitx }
219 {
220     group-digits / false .code:n =
221     {
222         \__siunitx_option_deprecated:nn
223         { group-digits ~ = ~ false }
224         { group-digits ~ = ~ none }
225     } ,
226     group-digits / true .code:n =

```

```

227     {
228         \__siunitx_option_deprecated:nn
229         { group-digits ~ = ~ true }
230         { group-digits ~ = ~ all }
231     } ,
232     input-symbols .code:n =
233     {
234         \msg_info:nnnn { siunitx } { option-deprecated }
235         { input-symbols } { input-digits }
236         \tl_put_right:Nn \l__siunitx_number_input_digit_tl {#1}
237     } ,
238     separate-uncertainty .choice: ,
239     separate-uncertainty / false .code:n =
240     {
241         \__siunitx_option_deprecated:nn
242         { separate-uncertainty }
243         { uncertainty-mode~==compact }
244     } ,
245     separate-uncertainty / true .code:n =
246     {
247         \__siunitx_option_deprecated:nn
248         { separate-uncertainty }
249         { uncertainty-mode~==separate }
250     } ,
251     separate-uncertainty .default:n = true
252 }

```

A small number of removed options.

```

253 \clist_map_inline:nn
254 {
255     input-protect-tokens ,
256     input-quotient ,
257     output-product ,
258     quotient-mode
259 }
260 {
261     \keys_define:nn { siunitx } { #1 .code:n = \__siunitx_option_removed:n {#1} }
262 }

```

Options for number processing: largely removals.

```

263 \keys_define:nn { siunitx }
264 {
265     add-decimal-zero .choice: ,
266     add-decimal-zero / false .code:n =
267     {
268         \__siunitx_option_deprecated:nn
269         { add-decimal-zero }
270         { minimum-decimal-digits~==0 }
271     } ,
272     add-decimal-zero / true .code:n =
273     {
274         \__siunitx_option_deprecated:nn
275         { add-decimal-zero }
276         { minimum-decimal-digits~==1 }
277     } ,

```

```

278 add-decimal-zero .default:n = true ,
279 add-integer-zero .code:n =
280   { \_siunitx_option_removed:V \l_keys_key_tl } ,
281 close-bracket .code:n =
282   { \_siunitx_option_removed:V \l_keys_key_tl } ,
283 bracket-numbers .choice: ,
284 bracket-numbers / false .code:n =
285   {
286     \_siunitx_option_deprecated:nn
287     { bracket-numbers }
288     { bracket-ambiguous-numbers~=-false }
289   } ,
290 bracket-numbers / true .code:n =
291   {
292     \_siunitx_option_deprecated:nn
293     { bracket-numbers }
294     { bracket-ambiguous-numbers~=-true }
295   } ,
296 bracket-numbers .default:n = true ,
297 explicit-sign .code:n =
298   {
299     \str_if_eq:nnTF {#1} { + }
300     {
301       \_siunitx_option_deprecated:nn
302       { explicit-sign }
303       { print-implicit-plus~=-true }
304     }
305     { \_siunitx_option_removed:V \l_keys_key_tl }
306   } ,
307 group-four-digits .choice: ,
308 group-four-digits / false .code:n =
309   {
310     \_siunitx_option_deprecated:nn
311     { group-four-digits~=-false }
312     { group-minimum-digits~=-5 }
313   } ,
314 group-four-digits / true .code:n =
315   {
316     \_siunitx_option_deprecated:nn
317     { group-four-digits~=-false }
318     { group-minimum-digits~=-4 }
319   } ,
320 bracket-numbers .default:n = true ,
321 omit-uncertainty .code:n =
322   {
323     \_siunitx_option_deprecated:nnV
324     { omit-uncertainty }
325     { drop-uncertainty }
326     \l_keys_value_tl
327   } ,
328 omit-uncertainty .default:n = true ,
329 open-bracket .code:n =
330   { \_siunitx_option_removed:V \l_keys_key_tl } ,
331 retain-unity-mantissa .code:n =

```

```

332     {
333         \_siunitx_option_deprecated:nnV
334         { retain-unity-mantissa }
335         { print-unity-mantissa }
336         \l_keys_value_tl
337     } ,
338     retain-unity-mantissa .default:n = true ,
339     retain-zero-exponent .code:n =
340     {
341         \_siunitx_option_deprecated:nnV
342         { retain-zero-exponent }
343         { print-zero-exponent }
344         \l_keys_value_tl
345     } ,
346     retain-zero-exponent .default:n = true ,
347     round-integer-to-decimal .code:n =
348     { \_siunitx_option_removed:V \l_keys_key_tl } ,
349     scientific-notation .choice: ,
350     scientific-notation / engineering .code:n =
351     {
352         \_siunitx_option_deprecated:nn
353         { scientific-notation~~engineering }
354         { exponent-mode~~engineering }
355     } ,
356     scientific-notation / fixed .code:n =
357     {
358         \_siunitx_option_deprecated:nn
359         { scientific-notation~~fixed }
360         { exponent-mode~~fixed }
361     } ,
362     scientific-notation / false .code:n =
363     {
364         \_siunitx_option_deprecated:nn
365         { scientific-notation~~false }
366         { exponent-mode~~input }
367     } ,
368     scientific-notation / true .code:n =
369     {
370         \_siunitx_option_deprecated:nn
371         { scientific-notation~~true }
372         { exponent-mode~~scientific }
373     } ,
374     scientific-notation .default:n = true ,
375     zero-decimal-to-integer .code:n =
376     {
377         \_siunitx_option_deprecated:nnV
378         { zero-decimal-to-integer }
379         { drop-zero-decimal }
380         \l_keys_value_tl
381     } ,
382     zero-decimal-to-integer .default:n = true
383 }

```

1.7.1 Table options

All straight-forward emulation.

```
384 \keys_define:nn { siunitx }
385 {
386   table-align-text-post .code:n =
387   {
388     \__siunitx_option_deprecated:nnV
389     { table-align-text-post }
390     { table-align-text-after }
391     \l_keys_value_tl
392   } ,
393   table-align-text-post .default:n = true ,
394   table-align-text-pre .code:n =
395   {
396     \__siunitx_option_deprecated:nnV
397     { table-align-text-pre }
398     { table-align-text-before }
399     \l_keys_value_tl
400   } ,
401   table-align-text-pre .default:n = true ,
402   table-number-alignment / center-decimal-marker .code:n =
403   {
404     \msg_info:nnnn { siunitx } { option-deprecated }
405     { table-number-alignment~==center-decimal-marker }
406     { table-alignment-mode~==marker }
407     \keys_set:nn
408     { siunitx }
409     { table-alignment-mode = marker }
410   } ,
411   table-omit-exponent .code:n =
412   {
413     \__siunitx_option_deprecated:nnV
414     { table-omit-exponent }
415     { drop-exponent }
416     \l_keys_value_tl
417   } ,
418   table-omit-exponent .default:n = true ,
419   table-parse-only .code:n =
420   {
421     \msg_info:nnnn { siunitx } { option-deprecated }
422     { table-parse-only }
423     { table-alignment-mode~==none }
424     \str_if_eq:VnTF \l_keys_value_tl { false }
425     {
426       \keys_set:nn
427       { siunitx }
428       { table-alignment-mode = marker }
429     }
430     {
431       \keys_set:nn
432       { siunitx }
433       { table-alignment-mode = none }
434     }
435   }
```

```

435     } ,
436     table-space-text-post .code:n =
437     {
438         \msg_info:nnnn { siunitx } { option-deprecated }
439         { table-space-text-post }
440         { table-format }
441         \tl_set:Nn \l__siunitx_table_after_model_tl {#1}
442     } ,
443     table-space-text-pre .code:n =
444     {
445         \msg_info:nnnn { siunitx } { option-deprecated }
446         { table-space-text-post }
447         { table-format }
448         \tl_set:Nn \l__siunitx_table_before_model_tl {#1}
449     }
450 }

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

451 \cs_new_protected:Npn \__siunitx_option_table_format:n #1
452 {
453     \msg_info:nnnn { siunitx } { option-deprecated }
454     { table- #1 }
455     { table-format }
456     \tl_set:Nx \l__siunitx_table_format_tl
457     {
458         \cs:w __siunitx_option_table_ #1 :nnnnnnnn
459         \exp_after:wN \exp_after:wN \exp_after:wN \cs_end:
460         \exp_after:wN \l__siunitx_table_format_tl
461         \exp_after:wN { \l_keys_value_tl }
462     }
463     \exp_after:wN \__siunitx_table_generate_model:nnnnnnn
464     \l__siunitx_table_format_tl
465 }
466 \cs_new:Npn \__siunitx_option_table_comparator:nnnnnnnn #1#2#3#4#5#6#7#8
467 { \exp_not:n { {#8} {#2} {#3} {#4} {#5} {#6} {#7} } }
468 \cs_new:cpn { __siunitx_option_table_figures-decimal:nnnnnnnn }
469 #1#2#3#4#5#6#7#8
470 { \exp_not:n { {#1} {#2} {#3} {#8} {#5} {#6} {#7} } }
471 \cs_new:cpn { __siunitx_option_table_figures-exponent:nnnnnnnn }
472 #1#2#3#4#5#6#7#8
473 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#8} } }
474 \cs_new:cpn { __siunitx_option_table_figures-integer:nnnnnnnn }
475 #1#2#3#4#5#6#7#8
476 { \exp_not:n { {#1} {#2} {#8} {#4} {#5} {#6} {#7} } }
477 \cs_new:cpn { __siunitx_option_table_figures-uncertainty:nnnnnnnn }
478 #1#2#3#4#5#6#7#8
479 { \exp_not:n { {#1} {#2} {#3} {#4} { { S } {#8} } {#6} {#7} } }
480 \cs_new:cpn { __siunitx_option_table_sign-exponent:nnnnnnnn }
481 #1#2#3#4#5#6#7#8
482 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#8} {#7} } }
483 \cs_new:cpn { __siunitx_option_table_sign-mantissa:nnnnnnnn }
484 #1#2#3#4#5#6#7#8
485 { \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.

```

486 \keys_define:nn { siunitx }
487 {
488   table-comparator .code:n =
489     { \__siunitx_option_table_format:n { comparator } } ,
490   table-figures-decimal .code:n =
491     { \__siunitx_option_table_format:n { figures-decimal } } ,
492   table-figures-exponent .code:n =
493     { \__siunitx_option_table_format:n { figures-exponent } } ,
494   table-figures-integer .code:n =
495     { \__siunitx_option_table_format:n { figures-integer } } ,
496   table-figures-uncertainty .code:n =
497     { \__siunitx_option_table_format:n { figures-uncertainty } } ,
498   table-sign-exponent .code:n =
499     { \__siunitx_option_table_format:n { sign-exponent } } ,
500   table-sign-mantissa .code:n =
501     { \__siunitx_option_table_format:n { sign-mantissa } }
502 }

```

1.8 Unit options

```

503 \keys_define:nn { siunitx }
504 {
505   fraction-function .code:n =
506   {
507     \__siunitx_option_deprecated:nnV
508     { fraction-function }
509     { fraction-command }
510     \l_keys_value_tl
511   } ,
512   literal-superscript-as-power .code:n =
513   { \__siunitx_option_removed:V \l_keys_key_tl } ,
514   per-mode / reciprocal .code:n =
515   {
516     \__siunitx_option_deprecated:nn
517     { per-mode~~reciprocal }
518     { per-mode~~power }
519   } ,
520   per-mode / reciprocal-positive-first .code:n =
521   {
522     \__siunitx_option_deprecated:nn
523     { per-mode~~reciprocal-positive-first }
524     { per-mode~~power-positive-first }
525   } ,
526   power-font .code:n =
527   { \__siunitx_option_removed:V \l_keys_key_tl } ,
528   qualifier-mode / brackets .code:n =
529   {
530     \__siunitx_option_deprecated:nn
531     { qualifier-mode~~brackets }
532     { qualifier-mode~~bracket }
533   } ,
534   qualifier-mode / space .code:n =
535   {

```

```

536     \msg_info:nnnn { siunitx } { option-deprecated }
537     { qualifier-mode~~space }
538     { qualifier-mode~~phrase"~plus~"qualifier-phrase=\ }
539     \keys_set:nn
540     { siunitx }
541     { qualifier-mode = phrase, qualifier-phrase = \ }
542   } ,
543   qualifier-mode / text .code:n =
544   {
545     \__siunitx_option_deprecated:nn
546     { qualifier-mode~~text }
547     { qualifier-mode~~combine }
548   }
549 }

```

1.9 Quantity units

```

550 \keys_define:nn { siunitx }
551 {
552   allow-number-unit-breaks .code:n =
553   {
554     \__siunitx_option_deprecated:nnV
555     { allow-number-unit-breaks }
556     { allow-quantity-breaks }
557     \l_keys_value_tl
558   } ,
559   allow-number-unit-breaks .default:n = true ,
560   exponent-to-prefix .choice: ,
561   exponent-to-prefix / false .code:n =
562   {
563     \__siunitx_option_deprecated:nn
564     { exponent-to-prefix~~false }
565     { prefix-mode~~input }
566   } ,
567   exponent-to-prefix / true .code:n =
568   {
569     \__siunitx_option_deprecated:nn
570     { exponent-to-prefix~~true }
571     { prefix-mode~~combine-exponent }
572   } ,
573   exponent-to-prefix .default:n = true ,
574   multi-part-units .choice: ,
575   multi-part-units / brackets . code:n =
576   {
577     \__siunitx_option_deprecated:nn
578     { multi-part-units~~brackets }
579     { separate-uncertainty-units~~bracket }
580   } ,
581   multi-part-units / repeat . code:n =
582   {
583     \__siunitx_option_deprecated:nn
584     { multi-part-units~~repeat }
585     { separate-uncertainty-units~~repeat }
586   } ,
587   multi-part-units / single . code:n =

```



```

588     {
589         \_siunitx_option_deprecated:nn
590         { multi-part-units~==single }
591         { separate-uncertainty-units~==single }
592     } ,
593     number-unit-product .code:n =
594     {
595         \_siunitx_option_deprecated:nnV
596         { number-unit-product }
597         { quantity-product }
598         \l_keys_value_tl
599     } ,
600     number-unit-separator .code:n =
601     {
602         \_siunitx_option_deprecated:nnV
603         { number-unit-separator }
604         { quantity-product }
605         \l_keys_value_tl
606     } ,
607     prefixes-as-symbols .choice: ,
608     prefixes-as-symbols / false . code:n =
609     {
610         \_siunitx_option_deprecated:nn
611         { prefixes-as-symbols~==false }
612         { prefix-mode~==extract-exponent }
613     } ,
614     prefixes-as-symbols / true . code:n =
615     {
616         \_siunitx_option_deprecated:nn
617         { prefixes-as-symbols~==true }
618         { prefix-mode~==input }
619     } ,
620     prefixes-as-symbols .default:n = true
621 }
622 </options>

```

1.10 Preamble commands

623 <*interfaces>

\DeclareBinaryPrefix We simply drop #3.

```

624 \NewDocumentCommand \DeclareBinaryPrefix { +m m m }
625 {
626     \siunitx_declare_prefix:Nn #1 {#2}
627 }

```

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

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

```

628 \NewDocumentCommand \DeclareSIPrePower { +m m }
629 {
630     \siunitx_declare_power:NNn #1 \_siunitx_tmp:w {#2}
631     \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
632 }

```

```

633 \NewDocumentCommand \DeclareSIPostPower { +m m }
634 {
635   \siunitx_declare_power:NNn \_siunitx_tmp:w #1 {#2}
636   \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
637 }

```

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

1.11 Document commands

\si A straight copy of \unit.

```

638 \NewDocumentCommand \si { O { } m }
639 {
640   \mode_leave_vertical:
641   \group_begin:
642     \keys_set:nn { siunitx } {#1}
643     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
644     \siunitx_print_unit:V \l__siunitx_tmp_tl
645   \group_end:
646 }

```

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

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

```

647 \NewDocumentCommand \SI { O { } m o m }
648 {
649   \mode_leave_vertical:
650   \group_begin:
651     \keys_set:nn { siunitx } {#1}
652     \IfNoValueF {#3}
653     {
654       \siunitx_unit_format:nN {#3} \l__siunitx_tmp_tl
655       \siunitx_print_unit:V \l__siunitx_tmp_tl
656       \nobreak
657     }
658     \siunitx_quantity:nn {#2} {#4}
659   \group_end:
660 }

```

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

\SIlist Straight copies.

```

\SIrange 661 \NewDocumentCommand \SIlist
662 { O { } > { \SplitList { ; } } m > { \TrimSpaces } m }
663 {
664   \mode_leave_vertical:
665   \group_begin:
666     \siunitx_unit_options_apply:n {#3}
667     \keys_set:nn { siunitx } {#1}
668     \siunitx_quantity_list:nn {#2} {#3}
669   \group_end:
670 }
671 \NewDocumentCommand \SIrange { O { } m m > { \TrimSpaces } m }

```

```

672 {
673   \mode_leave_vertical:
674   \group_begin:
675     \siunitx_unit_options_apply:n {#4}
676     \keys_set:nn { siunitx } {#1}
677     \siunitx_quantity_range:nnn {#2} {#3} {#4}
678   \group_end:
679 }

```

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

1.12 Symbol commands

```

680 <@@=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.

```

681 \bool_lazy_or:nnTF
682 { \sys_if_engine luatex_p: }
683 { \sys_if_engine xetex_p: }
684 {
685   \cs_new:Npn \_siunitx_emulation_non_latin:n #1
686     { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
687 }
688 {
689   \cs_new:Npn \_siunitx_emulation_non_latin:n #1
690     {
691       \exp_last_unbraced:Nf \_siunitx_emulation_non_latin:nnnn
692       { \char_to_utfviii_bytes:n {#1} }
693     }
694   \cs_new:Npn \_siunitx_emulation_non_latin:nnnn #1#2#3#4
695     {
696       \exp_after:wN \exp_after:wN \exp_after:wN
697       \exp_not:N \char_generate:nn {#1} { 13 }
698       \exp_after:wN \exp_after:wN \exp_after:wN
699       \exp_not:N \char_generate:nn {#2} { 13 }
700     }
701 }

```

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

\SIUnitSymbolAngstrom The same setup as elsewhere but localised to the emulation module

\SIUnitSymbolArcminute

```

702 \AtBeginDocument

```

\SIUnitSymbolArcsecond

```

703 {

```

\SIUnitSymbolCelsius

```

704   \cs_new_protected:Npn \SIUnitSymbolArcminute

```

\SIUnitSymbolDegree

```

705     { \ensuremath { { } ^{\circ} } }

```

\SIUnitSymbolMicro

```

706   \cs_new_protected:Npn \SIUnitSymbolArcsecond

```

\SIUnitSymbolOhm

```

707     { \ensuremath { { } ^{\mu} } }

```

```

708   \@ifpackageloaded { fontspec }

```

```

709   {

```

```

710     \cs_new_protected:Npx \SIUnitSymbolAngstrom

```

```

711       { \_siunitx_emulation_non_latin:n { "00C5 } }

```

```

712     \cs_new_protected:Npx \SIUnitSymbolDegree

```

```

713       { \_siunitx_emulation_non_latin:n { "00B0 } }

```

```

714     \cs_new_protected:Npx \SIUnitSymbolCelsius

```

```

715       { \_siunitx_emulation_non_latin:n { "00B0 } C }

```

```

716     }
717     {
718       \cs_new_protected:Npx \SIUnitSymbolAngstrom
719       {
720         \siunitx_print_text:n
721         { \_siunitx_emulation_non_latin:n { "00C5 } }
722       }
723       \cs_new_protected:Npx \SIUnitSymbolCelsius
724       {
725         \siunitx_print_text:n
726         { \_siunitx_emulation_non_latin:n { "00B0 } } C
727       }
728       \cs_new_protected:Npx \SIUnitSymbolDegree
729       {
730         \siunitx_print_text:n
731         { \_siunitx_emulation_non_latin:n { "00B0 } }
732       }
733     }
734     \cs_new_protected:Npx \SIUnitSymbolMicro
735     {
736       \siunitx_print_text:n
737       {
738         \bool_lazy_or:nnTF
739         { \sys_if_engine luatex_p: }
740         { \sys_if_engine xetex_p: }
741         { \_siunitx_emulation_non_latin:n { "00B5 } }
742         { \exp_not:N \textmu }
743       }
744     }
745     \cs_new_protected:Npx \SIUnitSymbolOhm
746     {
747       \exp_not:N \ifmmode
748       \cs_if_exist:NTF \upOmega
749       { \exp_not:N \upOmega }
750       { \exp_not:N \Omega }
751       \exp_not:N \else
752       \siunitx_print_text:n
753       {
754         \bool_lazy_or:nnTF
755         { \sys_if_engine luatex_p: }
756         { \sys_if_engine xetex_p: }
757         { \_siunitx_emulation_non_latin:n { "03A9 } }
758         { \exp_not:N \textohm }
759       }
760       \exp_not:N \fi
761     }
762   }

```

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

1.13 Unit commands

\celsius Deprecated but should work.

```

763 \siunitx_declare_unit:Nn \celsius { \degreeCelsius }

```

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

Units that have been removed: to avoid issues, we mark them as deprecated.

```

764 \msg_new:nnn { siunitx } { unit-deprecated }
765 {
766   Unit~macro~#1~has~been~deprecated~in~this~release. \\\
767   The~BIPM~have~removed~this~unit~from~the~SI~Brochure.~
768   You~should~define~it~yourself~using~\token_to_str:N \DeclareSIUnit\ %
769   in~your~source.~The~current~definition~is\\\
770   \token_to_str:N \DeclareSIUnit #1 \{ #2 \}
771 }
772 \cs_gset_protected:Npn \__siunitx_emulation_tmp:w #1#2
773 {
774   \quark_if_recursion_tail_stop:N #1
775   \bool_new:c { g__siunitx_emulation_unit_warning_ \token_to_str:N #1 _bool }
776   \siunitx_declare_unit:Nx #1
777   {
778     \exp_not:N \bool_if:NF
779     \exp_not:c { g__siunitx_emulation_unit_warning_ \token_to_str:N #1 _bool }
780     {
781       \exp_not:N \bool_gset_true:N
782       \exp_not:c { g__siunitx_emulation_unit_warning_ \token_to_str:N #1 _bool }
783       \msg_warning:nnnn { siunitx } { unit-deprecated }
784       { \token_to_str:N #1 } { #2 }
785     }
786     #2
787   }
788   \__siunitx_emulation_tmp:w
789 }
790 \__siunitx_emulation_tmp:w
791 \atomicmassunit { u }
792 \bar { bar }
793 \barn { b }
794 \bohr
795 {
796   \exp_not:N \text
797   { \exp_not:N \ensuremath { a } } \char_generate:nn { '\_ } { 8 } { 0 }
798 }
799 \clight
800 {
801   \exp_not:N \text
802   { \exp_not:N \ensuremath { c } } \char_generate:nn { '\_ } { 8 } { 0 }
803 }
804 \electronmass
805 {
806   \exp_not:N \text { \exp_not:N \ensuremath { m } }
807   \char_generate:nn { '\_ } { 8 } { \exp_not:N \mathrm { e } }
808 }
809 \elementarycharge { \text { \ensuremath { e } } }
810 \hartree
811 {
812   \exp_not:N \text { \exp_not:N \ensuremath { E } }
813   \char_generate:nn { '\_ } { 8 } { \exp_not:N \mathrm { h } }
814 }
815 \knot { kn }

```

```

816 \mmHg          { mmHg }
817 \nauticalmile  { M }
818 \planckbar
819   { \exp_not:N \text { \exp_not:N \ensuremath { \exp_not:N \hbar } } }
820 \q_recursion_tail { }
821 \q_recursion_stop
822 \ifpackageloaded { fontspec }
823 {
824   \__siunitx_emulation_tmp:w \angstrom { \__siunitx_emulation_non_latin:n { "00C5 } }
825 }
826 {
827   \__siunitx_emulation_tmp:w \angstrom
828   { \exp_not:N \text { \__siunitx_emulation_non_latin:n { "00C5 } } }
829 }
830 \q_recursion_tail { }
831 \q_recursion_stop

```

1.14 Communication with pgf

```

832 <@=siunitx_number>

```

\SendSettingsToPgf

```

833 \NewDocumentCommand \SendSettingsToPgf { }
834 {
835   \use:x
836   {
837     \exp_not:N \pgfqkeys { /pgf/number~format }
838     {
839       \str_if_eq:VnT \l__siunitx_number_round_mode_tl { figures }
840       {
841         fixed ,
842         fixed~zerofill = true ,
843       }
844       precision = \int_use:N \l__siunitx_number_round_precision_int ,
845       set~decimal~separator =
846       \str_if_eq:VnTF \l_siunitx_number_output_decimal_tl { , }
847       { \exp_not:N \mathord }
848       { \use:n }
849       { \exp_not:N \l_siunitx_number_output_decimal_tl } ,
850       set~thousands~separator =
851       set~decimal~separator =
852       \str_if_eq:VnTF \l__siunitx_number_group_separator_tl { , }
853       { \exp_not:N \mathord }
854       { \use:n }
855       { \exp_not:N \l__siunitx_number_group_separator_tl } ,
856       min~exponent~for~1000~sep =
857       \int_eval:n { \l__siunitx_number_group_minimum_int - 1 } ,
858       \bool_lazy_or:nnF
859       { \l__siunitx_number_group_decimal_bool }
860       { \l__siunitx_number_group_integer_bool }
861       { min~exponent~for~1000~sep = 999 , }
862       showpos =
863       \bool_if:NTF \l__siunitx_number_implicit_plus_bool
864       { true }

```

```

865         { false }
866     }
867 }
868 }

```

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

```

869 </interfaces>

```

```

870 </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			
<code>\%</code>	1102	<code>\boldmath</code>	94
<code>\,</code>	107, 141, 9, 15, 21, 27, 168, 1152, 2116, 2122	bool commands:	
<code>\-</code>	102	<code>\bool_gset_true:N</code>	781
<code>\%</code>	6, 84, 121, 766, 769, 1114	<code>\bool_if:NTF</code>	10, 16, 18, 18, 32, 34, 39, 47, 50, 60, 61, 62, 69, 73, 80, 87, 91, 96, 106, 108, 111, 116, 119, 127, 128, 144, 151, 153, 164, 166, 166, 182, 183, 186, 192, 218, 225, 234, 245, 247, 249, 253, 255, 256, 263, 267, 291, 311, 378, 381, 407, 408, 416, 438, 448, 457, 476, 477, 482, 495, 583, 622, 648, 668, 702, 708, 725, 776, 778, 785, 799, 863, 863, 883, 895, 912, 932, 946, 955, 958, 1003, 1013, 1316, 1596, 1770, 1776, 1792, 1834, 2007
<code>\%</code>	8, 203, 214, 291, 385, 797, 802, 807, 813	<code>\bool_lazy_all:nTF</code>	1733
<code>\%</code>	212	<code>\bool_lazy_all_p:n</code>	1108, 1126
<code>\%</code>	50, 52, 56, 58, 62, 64, 114, 538, 541, 542, 544, 548, 550, 553, 559, 561, 565, 567, 570, 576, 578, 585, 587, 768, 1121	<code>\bool_lazy_and:nnTF</code>	101, 111, 145, 218, 270, 285, 432, 516, 602, 682, 692, 706, 897, 905, 1322, 1420, 1521, 1755, 2028
A		<code>\bool_lazy_and_p:nn</code>	1524, 1811
<code>\A</code>	182, 2	<code>\bool_lazy_any:nTF</code>	251, 1806
allow-quantity-breaks	107	<code>\bool_lazy_or:nnTF</code>	10, 126, 142, 148, 171, 177, 257, 318, 391, 397, 444, 544, 681, 738, 754, 858, 1014, 1105, 1123, 1451, 1974, 1990, 2003
<code>\ampere</code>	145, 2, 3, 4, 5, 6, 7, 1034	<code>\bool_lazy_or_p:nn</code>	1739
<code>\ang</code>	6, 100, 291, 327	<code>\bool_new:N</code>	3, 6, 9, 10, 10, 11, 12, 18, 19, 20, 21, 22, 23, 43, 44, 77, 88, 358, 448, 571, 576, 577, 578, 579, 580, 581, 584, 669, 775, 1636, 1698, 1699
angle-mode	12	<code>\bool_set_false:N</code>	9, 12, 16, 21, 22, 25, 29, 29, 34, 35, 39, 40, 50, 57, 58, 63, 64, 68, 70, 74, 75, 80, 81, 82, 88, 134, 148, 155, 171, 209, 216, 256, 260, 349, 364, 365, 459, 523, 524, 530, 531, 532, 533, 537, 538, 539, 544, 547, 551, 609, 611, 658, 685, 724, 772, 807, 879, 1657, 1661, 1666, 1667
angle-symbol-degree	12	<code>\bool_set_true:N</code>	11, 14, 17, 24, 30, 30, 34, 35, 52, 56, 62, 63, 69, 76, 102, 141, 163, 205, 208, 209, 234, 254, 259, 261, 363, 374, 457, 459, 471, 472, 525, 526,
angle-symbol-minute	12		
angle-symbol-over-decimal	12		
angle-symbol-second	12		
<code>\angstrom</code>	824, 827		
<code>\approx</code>	2118		
arc-separator	12		
<code>\arcminute</code>	146, 53, 55, 286, 367, 1099		
<code>\arcsecond</code>	146, 58, 60, 288, 372, 1099		
<code>\as</code>	182, 89		
<code>\astronomicalunit</code>	146, 1086		
<code>\AtBeginDocument</code>	5, 17, 37, 51, 57, 73, 151, 227, 234, 258, 324, 332, 702, 1162		
<code>\atomicmassunit</code>	791		
<code>\atto</code>	145, 90, 1043		
B			
<code>\bar</code>	792		
<code>\barn</code>	793		
<code>\becquerel</code>	146, 1064		
<code>\begin</code>	203		
<code>\bel</code>	146, 69, 1086		
<code>\bfseries</code>	93		
<code>\binoppenalty</code>	154		
<code>\bit</code>	188, 10		
<code>\bohr</code>	794		

540, 545, 546, 552, 553, 554, 558, 559, 560, 561, 612, 656, 869, 900, 939, 1651, 1652, 1656, 1662, 2042, 2043	\color 38, 274, 1758
\c_false_bool 47, 121, 388, 450, 454, 459, 467, 491, 497, 535, 550, 592, 615	color 94
\c_true_bool 44, 118, 373, 388, 448, 464, 491, 497, 548, 619, 629	\complexnum 192
box commands:	\complexqty 192
\box_clear:N 553, 616	compound-exponents 21
\box_new:N 3, 170, 171, 263, 264, 528, 529	compound-final-separator 21
\box_use:N 249	compound-pair-separator 21
\box_use_drop:N 244, 246, 476, 477, 522, 523, 597, 598, 599, 600, 655, 656, 657, 658	compound-separator 21
\box_wd:N 228, 229, 263, 284, 287, 288, 295, 296, 302, 303, 304, 305, 315, 316, 322, 323, 324, 325, 449, 493, 497, 511, 551, 558, 559, 562, 570, 614, 619, 646, 671, 682, 683, 694, 698, 699, 712, 713, 723, 731, 733, 751, 752, 773, 784, 803, 805, 815, 827	compound-separator-mode 21
bracket-ambiguous-numbers 40	compound-units 21
bracket-negative-numbers 40	\coulomb 146, 1064
bracket-unit-denominator 147	\cr 120, 29
\byte 188, 10	cs commands:
	\cs:w 117, 188, 458, 722, 1027
	\cs_end: 117, 188, 459, 725, 1030
	\cs_generate_variant:Nn 2, 3, 3, 4, 5, 5, 6, 7, 21, 27, 61, 62, 65, 72, 87, 115, 125, 146, 163, 181, 192, 194, 199, 208, 211, 262, 304, 375, 382, 382, 499, 547, 796, 851, 879, 929, 1040, 1227, 1230, 1241, 1831, 1949, 1964
	\cs_gset_protected:Npn 772
	\cs_if_eq:NNTF 374
	\cs_if_exist:NTF 101, 115, 118, 134, 234, 264, 347, 748
	\cs_if_free:NTF 7, 43
	\cs_new:Npn 14, 18, 23, 31, 68, 113, 123, 140, 142, 168, 175, 179, 184, 209, 210, 230, 241, 246, 247, 267, 358, 360, 364, 375, 383, 383, 385, 389, 398, 400, 403, 407, 416, 466, 468, 471, 474, 477, 480, 483, 603, 661, 678, 685, 689, 694, 700, 718, 733, 737, 739, 746, 748, 754, 774, 786, 797, 803, 809, 815, 827, 834, 852, 874, 880, 881, 888, 908, 913, 927, 937, 950, 958, 971, 985, 991, 996, 1009, 1018, 1021, 1022, 1027, 1033, 1035, 1041, 1050, 1066, 1080, 1098, 1119, 1137, 1167, 1194, 1201, 1208, 1213, 1223, 1226, 1228, 1231, 1242, 1251, 1258, 1265, 1266, 1267, 1273, 1278, 1284, 1289, 1303, 1311, 1320, 1331, 1346, 1354, 1372, 1388, 1389, 1407, 1416, 1418, 1440, 1449, 1466, 1468, 1477, 1483, 1502, 1509, 1515, 1517, 1548, 1555, 1566, 1571, 1576, 1581, 1589, 1602, 1612, 1614, 1619, 1624, 1626, 1701, 1703, 1705, 1707, 1709, 1714, 1719, 1731, 1746, 1753, 1760, 1766, 1784, 1790, 1796, 1804, 1818, 1832, 1843, 1852, 1854, 1856, 1858, 1864, 1874,
C	
\cancel 141, 147, 153, 107	
\candela 145, 1034	
\cdot 8, 33, 289	
\celsius 763	
\centi 145, 65, 1043	
char commands:	
\char_generate:nn 8, 15, 27, 28, 176, 187, 189, 203, 214, 291, 385, 686, 697, 699, 797, 802, 807, 813, 1019, 1030, 1031	
\char_set_active_eq:NN 222, 224, 226, 455, 501	
\char_set_active_eq:nN 38	
\char_set_catcode_active:N 102	
\char_set_catcode_active:n 212	
\char_set_mathcode:nn 456, 502	
\char_to_utfviii_bytes:n 21, 182, 692, 1025	
\char_value_catcode:n 15, 176, 686, 1019	
\clight 799	
clist commands:	
\clist_map_break: 119	
\clist_map_function:nN 34, 39	
\clist_map_inline:nn 19, 28, 95, 97, 107, 123, 165, 174, 198, 253, 502, 614	
\cm 182, 60	

1879, 1899, 1908, 1910, 1929, 1950, 1965, 1970, 1972, 1981, 1987, 2001, 2018, 2024, 2036, 2071, 2077, 2082	241, 243, 255, 262, 278, 328, 352, 460
<code>\cs_new:Npx</code> 245, 405, 414	<code>\cs_set_protected:Npx</code> 67, 198
<code>\cs_new_eq:NN</code> 238, 265, 266, 772, 1587	<code>\cs_to_str:N</code> 188, 287, 341, 1102
<code>\cs_new_protected:Npn</code> 7, 11, 14, 15, 16, 22, 23, 24, 25, 27, 28, 37, 37, 39, 40, 40, 45, 45, 49, 50, 55, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68, 70, 70, 73, 78, 78, 80, 85, 86, 89, 89, 91, 92, 93, 94, 94, 98, 99, 101, 101, 105, 109, 110, 116, 126, 128, 129, 131, 132, 132, 135, 137, 138, 139, 140, 141, 143, 144, 144, 146, 147, 149, 152, 153, 153, 156, 159, 160, 162, 164, 166, 168, 172, 174, 175, 176, 176, 180, 180, 182, 182, 184, 187, 187, 191, 193, 195, 204, 207, 208, 210, 210, 212, 214, 216, 217, 219, 221, 223, 228, 232, 236, 241, 241, 244, 246, 248, 248, 249, 250, 254, 257, 260, 260, 261, 264, 267, 270, 272, 272, 272, 275, 277, 279, 283, 284, 294, 295, 297, 299, 299, 303, 305, 305, 305, 308, 316, 316, 327, 330, 331, 338, 338, 340, 340, 341, 342, 345, 347, 348, 350, 351, 360, 363, 364, 369, 371, 372, 373, 373, 376, 377, 385, 387, 390, 393, 398, 399, 416, 417, 424, 430, 430, 430, 431, 432, 436, 442, 442, 444, 451, 452, 455, 457, 459, 462, 466, 467, 469, 469, 473, 480, 480, 481, 486, 487, 500, 500, 506, 506, 511, 513, 515, 520, 526, 527, 527, 527, 538, 545, 548, 554, 567, 576, 584, 591, 595, 597, 604, 610, 620, 626, 628, 632, 660, 663, 673, 676, 688, 692, 704, 704, 706, 710, 718, 720, 722, 739, 744, 744, 756, 764, 768, 774, 778, 780, 789, 797, 802, 812, 814, 823, 831, 834, 835, 845, 857, 871, 876, 888, 893, 919, 930, 942, 951, 963, 966, 979, 988, 997, 1001, 1011, 1023, 1594	<code>\cs_undefine:N</code> 51, 52, 65, 236, 266
<code>\cs_set:Npn</code> 30, 35	<code>\cubed</code> 147, 1103
<code>\cs_set_eq:NN</code> 123, 135, 157, 180, 334, 340, 351	<code>\cubic</code> 146, 1103
<code>\cs_set_protected:Npn</code> 29, 44, 77, 86, 231, 237,	
	D
	<code>\dalton</code> 146, 1086
	<code>\day</code> 146, 1086
	<code>\dB</code> 184, 69
	<code>\deca</code> 145, 1053
	<code>\deci</code> 145, 66, 69, 1043, 1088
	<code>\decibel</code> 146, 1086
	<code>\DeclareBinaryPrefix</code> 624
	<code>\DeclareCurrentRelease</code> 5, 8
	<code>\DeclareExpandableDocumentCommand</code> . 286
	<code>\DeclareRelease</code> 4, 6, 7
	<code>\DeclareSIPostPower</code> 628
	<code>\DeclareSIPower</code> 62
	<code>\DeclareSIPrefix</code> 62
	<code>\DeclareSIPrePower</code> 628
	<code>\DeclareSIQualifier</code> 62
	<code>\DeclareSIUnit</code> 62, 768, 770
	<code>\DeclareTranslation</code> . 40, 41, 42, 43, 44, 45
	<code>\def</code> 347
	<code>\degree</code> 112, 146, 63, 65, 192, 285, 363, 1099
	<code>\degreeCelsius</code> 146, 81, 85, 763, 1064
	<code>\deka</code> 145, 1053
	dim commands:
	<code>\dim_add:Nn</code> 680, 729, 824
	<code>\dim_compare:nNnTF</code> 204, 228, 283, 551, 557, 614
	<code>\dim_compare_p:nNn</code> 694
	<code>\dim_new:N</code> 4, 4, 198, 530, 531
	<code>\dim_set:Nn</code> 203, 263, 619, 722, 749, 773, 784, 812
	<code>\c_zero_dim</code> 204
	<code>\dm</code> 182, 60
	<code>drop-exponent</code> 40
	<code>drop-uncertainty</code> 40
	<code>drop-zero-decimal</code> 40
	E
	<code>\edef</code> 348
	<code>\electronmass</code> 804
	<code>\electronvolt</code> 146, 52, 53, 54, 55, 56, 57, 1086
	<code>\elementarycharge</code> 809
	<code>\else</code> 50, 52, 56, 58, 62, 64, 123, 542, 544, 548, 550, 553, 559, 561, 565, 567, 570, 576, 578, 585, 587, 751
	else commands:
	<code>\else:</code> 356

<code>\end</code>	55, 84, 200	316, 320, 321, 322, 325, 342, 350,
<code>\endinput</code>	21	351, 352, 381, 382, 384, 386, 407,
<code>\ensuremath</code>	39, 93, 25, 53, 56, 61,	410, 417, 418, 639, 697, 699, 742,
	109, 118, 148, 178, 236, 257, 282,	747, 749, 750, 751, 758, 760, 778,
	284, 307, 382, 400, 447, 491, 495,	779, 781, 782, 796, 797, 801, 802,
	705, 707, 797, 802, 806, 809, 812, 819	806, 807, 812, 813, 819, 828, 837,
<code>\ERROR</code>	44	847, 853, 1030, 1758, 1824, 1867, 2008
<code>\eV</code>	183, 42	<code>\exp_not:n</code>
<code>evaluate-expression</code>	41	112, 113, 114, 114, 119, 122, 130,
<code>\exa</code>	145, 1053	143, 169, 169, 201, 216, 216, 222,
<code>\exbi</code>	188, 2	228, 243, 280, 281, 284, 288, 289,
<code>exp</code> commands:		290, 290, 292, 293, 297, 319, 320,
<code>\exp_after:wN</code>	26,	322, 326, 332, 334, 349, 350, 356,
	68, 81, 106, 115, 135, 141, 148,	357, 358, 363, 364, 366, 367, 369,
	149, 152, 153, 156, 176, 178, 186,	380, 389, 394, 396, 397, 398, 399,
	187, 188, 189, 205, 235, 239, 245,	399, 411, 416, 437, 437, 439, 441,
	246, 271, 355, 357, 359, 372, 373,	467, 470, 473, 475, 476, 479, 482,
	381, 384, 459, 460, 461, 463, 483,	483, 484, 485, 491, 492, 493, 532,
	484, 508, 581, 596, 623, 629, 642,	590, 606, 608, 611, 686, 689, 704,
	676, 696, 698, 698, 713, 724, 729,	705, 706, 708, 710, 738, 741, 744,
	730, 741, 969, 1006, 1016, 1029,	747, 751, 756, 759, 760, 764, 782,
	1029, 1298, 1434, 1599, 1712, 2079	806, 828, 829, 830, 831, 837, 840,
<code>\exp_args:Nc</code>	129, 286	841, 842, 842, 849, 852, 853, 855,
<code>\exp_args:Ne</code>		861, 865, 866, 880, 890, 910, 917,
	48, 63, 1205, 1376, 1511, 1622	921, 922, 923, 923, 925, 926, 931,
<code>\exp_args:Nf</code>	416, 735,	936, 937, 941, 945, 946, 947, 962,
	750, 1075, 1093, 1369, 1470, 1487, 1529	973, 973, 974, 975, 976, 976, 979,
<code>\exp_args:Nff</code>	411	983, 983, 984, 985, 992, 993, 998,
<code>\exp_args:NNc</code>	246	1008, 1009, 1010, 1010, 1022, 1034,
<code>\exp_args:Nnno</code>	1459	1165, 1166, 1275, 1326, 1328, 1337,
<code>\exp_args:NNNV</code>		1340, 1344, 1378, 1386, 1388, 1424,
	26, 138, 239, 377, 634, 687, 809, 967	1429, 1430, 1431, 1438, 1447, 1454,
<code>\exp_args:NNnx</code>	234	1458, 1463, 1467, 1604, 1606, 1609,
<code>\exp_args:NNV</code>	16, 112, 353, 369	1610, 1621, 1629, 1630, 1748, 1750,
<code>\exp_args:NnV</code>	259, 318	1758, 1763, 1764, 1782, 1788, 1793,
<code>\exp_args:No</code>	159	1794, 1799, 1801, 1820, 1826, 1828,
<code>\exp_args:NV</code>		1838, 1841, 1860, 1869, 1870, 1883,
	73, 114, 211, 236, 358, 1254, 1617	1884, 1888, 1892, 1909, 1914, 1916,
<code>\exp_args:Nv</code>	70	1922, 1923, 1925, 1971, 1989, 1999,
<code>\exp_args:Nx</code>	155, 301, 624	2010, 2011, 2013, 2014, 2020, 2021
<code>\exp_args:Nxx</code>	98	<code>\expandafter</code>
<code>\exp_last_unbraced:Nf</code>		<code>\ExplFileDate</code>
	20, 181, 691, 1024, 1175, 1479	1, 12
<code>\exp_not:N</code> 27, 67, 70, 74, 79, 83, 87,		<code>exponent-base</code>
	87, 90, 102, 103, 106, 109, 113, 116,	41
	118, 123, 124, 130, 132, 140, 146,	<code>exponent-mode</code>
	157, 158, 187, 189, 193, 195, 200,	41
	200, 201, 201, 202, 202, 203, 204,	<code>exponent-product</code>
	205, 207, 208, 215, 217, 217, 218,	41
	218, 220, 222, 223, 224, 225, 226,	<code>expression</code>
	227, 279, 279, 280, 281, 282, 282,	41
	283, 286, 286, 287, 287, 288, 288,	<code>extract-mass-in-kilograms</code>
	289, 289, 289, 290, 290, 291, 292,	148
	293, 294, 294, 295, 297, 314, 315,	

F

<code>\F</code>	184, 70
<code>\fam</code>	178, 179, 193
<code>\familydefault</code>	93, 246, 1165
<code>\farad</code>	146, 70, 71, 72, 73, 74, 1064
<code>\femto</code>	145, 15, 35, 71, 91, 1043

\if_meaning:w	354	\kA	182, 2
\IfFormatAtLeastTF	42, 59	\katal	146, 1064
\ifmmode	50, 52, 56, 58, 62, 64, 113, 542, 544, 548, 550, 553, 559, 561, 565, 567, 570, 576, 578, 585, 587, 747	\kelvin	145, 68, 1034
\IfNoValueF	652	\keV	183, 42
\IfNoValueTF	74, 76	keys commands:	
\ignorespaces	117, 37, 215, 432	\l_keys_choice_tl	18, 21, 45, 346, 363, 409, 415, 454, 464, 507, 511, 567, 647, 663, 1694
input-close-uncertainty	41	\keys_define:nn	3, 4, 6, 9, 11, 12, 24, 28, 30, 36, 38, 46, 48, 72, 93, 104, 122, 172, 196, 199, 216, 218, 250, 261, 263, 340, 353, 384, 405, 450, 486, 503, 503, 510, 532, 550, 637, 1638
input-comparators	41	\l_keys_key_tl	280, 282, 305, 330, 348, 513, 527
input-decimal-markers	41	\keys_set:nn	2, 14, 19, 39, 46, 51, 80, 93, 94, 96, 97, 104, 112, 133, 144, 152, 155, 162, 164, 164, 171, 180, 188, 196, 205, 213, 220, 278, 278, 282, 375, 407, 407, 424, 426, 431, 433, 488, 529, 537, 539, 624, 642, 651, 667, 676, 841, 1147, 1167, 2101
input-digits	41	\l_keys_value_tl	45, 53, 61, 69, 100, 326, 336, 344, 380, 391, 399, 416, 424, 461, 510, 557, 598, 605
input-exponent-markers	41	\kg	182, 35
input-open-uncertainty	41	\kHz	182, 8
input-signs	41	\kibi	188, 2
input-uncertainty-signs	41	\kilo	145, 160, 7, 10, 20, 26, 41, 45, 51, 54, 58, 67, 80, 83, 87, 686, 808, 1034, 1053
int commands:		\kilogram	145, 1034
\int_abs:n	191, 843, 846, 886, 1943	\kJ	183, 42
\int_case:nnTF	307, 343, 385, 896	\km	182, 60
\int_compare:nNnTF	193, 198, 401, 420, 599, 607, 757, 788, 791, 805, 836, 883, 894, 987, 1073, 1084, 1091, 1152, 1155, 1186, 1216, 1244, 1261, 1296, 1307, 1335, 1350, 1358, 1379, 1391, 1836, 1931, 1954	\kmol	183, 14
\int_compare_p:nNn	113, 178, 179, 1106, 1124, 1453, 1522, 1976	\kN	183, 78
\int_do_while:nNnn	606	\knot	815
\int_eval:n	184, 406, 407, 433, 465, 505, 736, 751, 805, 820, 830, 857, 1076, 1094, 1162, 1190, 1361, 1370, 1401, 1411, 1472, 1512, 1848, 1918, 1934, 1958, 1984	\kohm	184, 86
\int_if_odd_p:n	1111, 1129	\kPa	184, 82
\int_incr:N	438, 613	\kV	183, 21
\int_max:nn	1491	\kW	183, 42
\int_min:nn	1534	\kWh	184, 42
\int_mod:nn	896, 903, 1848		
\int_new:N	5, 11, 359, 590	L	
\int_set:Nn	331, 389	\L	183, 27
\int_set_eq:NN	593	\l	183, 27
\int_step_inline:nn	369, 662, 678	\le	38, 97, 2118
\int_step_inline:nnn	37	\leq	2118
\int_use:N	391, 396, 419, 423, 442, 446, 451, 623, 744, 788, 793, 844	list-exponents	21
\int_zero:N	366, 597	list-final-separator	21
inter-unit-product	148	list-pair-separator	21
		list-separator	21
J		list-units	21
\J	183, 42	\liter	146, 31, 32, 33, 34, 1086
\joule	146, 48, 49, 50, 51, 1064		
K			
\K	182, 68		

<code>\litre</code>	146, 27, 28, 29, 30, 1086	<code>\Mohm</code>	184, 86
<code>\ll</code>	38, 96, 2118	<code>\mohm</code>	184, 86
<code>locale</code>	36	<code>\mol</code>	183, 14
<code>\LRE</code>	333	<code>\mole</code> ..	145, 14, 15, 16, 17, 18, 19, 20, 1034
<code>\lumen</code>	146, 1064	<code>\mp</code>	38, 94, 283, 284, 2124
<code>\lux</code>	146, 1064	<code>\MPa</code>	184, 82
M		<code>\ms</code>	182, 89
<code>\m</code>	182, 60	msg commands:	
<code>\mA</code>	182, 2	<code>\msg_error:nn</code>	461
<code>\mathbf</code>	169	<code>\msg_error:nnn</code>	31, 168, 193, 507, 640, 655
<code>\mathchoice</code>	141, 175, 267, 309, 972	<code>\msg_error:nnnn</code>	379, 412, 426
<code>\mathit</code>	169	<code>\msg_info:nnnn</code>	13,
<code>\mathord</code>	280, 282, 447, 513, 847,		18, 234, 404, 421, 438, 445, 453, 536
	853, 1763, 1788, 1793, 1824, 1867, 2008	<code>\msg_new:nnn</code>	4, 9, 35, 82, 119, 256, 764
<code>\mathrm</code>	93, 141, 181, 807, 813, 1160	<code>\msg_new:nnnn</code>	25, 1105, 1111,
<code>\mathsf</code>	100, 156, 169, 1167		1117, 1123, 1129, 1135, 1141, 2095
<code>\mathtt</code>	100, 157, 169	<code>\msg_warning:nn</code>	87, 124
<code>\mathversion</code>	93–95, 141	<code>\msg_warning:nnn</code>	24, 31, 240, 267
<code>\mbox</code>	93, 141, 320, 392, 403, 404	<code>\msg_warning:nnnn</code>	783
<code>\mdseries</code>	94	<code>\mV</code>	183, 21
<code>\mebi</code>	188, 2	<code>\MW</code>	183, 42
<code>\mega</code>	145, 11, 46, 55, 81, 84, 88, 1053	<code>\mW</code>	183, 42
<code>\MessageBreak</code>	18	N	
<code>\meter</code>	145, 177, 1034	<code>\N</code>	183, 78
<code>\metre</code>	145, 146, 160,	<code>\nA</code>	182, 2
	177, 60, 61, 62, 63, 64, 65, 66, 67, 1034	<code>\nano</code> ...	145, 4, 17, 23, 37, 62, 73, 93, 1043
<code>\MeV</code>	183, 42	<code>\nauticalmile</code>	817
<code>\meV</code>	183, 42	<code>negative-color</code>	42
<code>\mg</code>	182, 35	<code>\neper</code>	146, 1086
<code>\mH</code>	75	<code>\newcolumn</code>	242, 269
<code>\MHz</code>	182, 8	<code>\NewDocumentCommand</code>	62, 66, 70,
<code>\mHz</code>	182, 8		74, 91, 100, 108, 128, 138, 148, 156,
<code>\micro</code>	145, 5, 18, 24, 30, 34,		166, 175, 184, 192, 200, 209, 219,
	38, 43, 49, 63, 74, 77, 94, 136, 138, 1043		624, 628, 633, 638, 647, 661, 671, 833
<code>\milli</code>	145, 6, 9, 19, 25, 29, 33,	<code>\newton</code>	146, 78, 79, 80, 81, 1075
	39, 44, 50, 53, 64, 76, 79, 86, 95, 1043	<code>\nF</code>	184, 70
<code>minimum-decimal-digits</code>	42	<code>\ng</code>	182, 35
<code>minimum-integer-digits</code>	42	<code>\nm</code>	182, 60
<code>\minute</code>	146, 1086	<code>\nmol</code>	183, 14
<code>\mJ</code>	183, 42	<code>\nobreak</code>	155, 194, 277, 656
<code>\mL</code>	183, 27	<code>\ns</code>	182, 89
<code>\ml</code>	183, 27	<code>\num</code>	139, 100, 292, 327
<code>\mm</code>	182, 60	<code>number-angle-product</code>	13
<code>\mmHg</code>	816	<code>number-color</code>	94
<code>\mmol</code>	183, 14	<code>number-mode</code>	94
<code>\MN</code>	183, 78	<code>\numlist</code>	138, 294, 328
<code>\mN</code>	183, 78	<code>\numproduct</code>	138, 308
<code>mode</code>	94	<code>\numrange</code>	184, 310, 329
mode commands:		<code>\nV</code>	183, 21
<code>\mode_if_math:TF</code>	82, 137, 207	O	
<code>\mode_leave_vertical:</code>	71, 93, 102,	<code>\of</code>	147, 113
	110, 130, 141, 150, 159, 169, 177,		
	186, 194, 202, 211, 640, 649, 664, 673		

\ohm	146, 86, 87, 88, 97, 105, 158, 1075	print-zero-exponent	42
\Omega	103, 750	\ProcessKeysOptions	56
output-close-uncertainty	42	product-exponents	21
output-decimal-marker	42	product-mode	21
output-open-uncertainty	42	product-phrase	21
\over	149	product-symbol	21
overwrite-commands	189	prop commands:	
P			
\Pa	184, 82	\prop_clear:N	362
\pA	182, 2	\prop_count:N	662
\PackageError	15	\prop_get:NnN	639
parse-numbers	42	\prop_get:NnNTF	477, 624,
parse-units	148	630, 634, 636, 651, 664, 691, 782, 794	
\pascal	146, 82, 83, 84, 85, 1075	\prop_if_empty:NTF	188
\pdfstringdefDisableCommands		\prop_if_in:NnTF	
155, 336, 344		376, 420, 472, 506, 680, 682, 789	
\pebi	188, 2	\prop_if_in_p:Nn	400
peek commands:		\prop_new:N	62, 63, 357
\peek_after:Nw	361	\prop_put:Nnn	
\peek_catcode_ignore_spaces:NTF	41, 155, 435	58, 59, 383, 490, 495, 653, 671, 689	
\l_peek_token	354	\prop_remove:N	669
\penalty	154	\prop_remove:Nn	474, 486, 649
\per	147–149, 153, 160,	propagate-math-font	94
163, 166, 104, 452, 460, 466, 1118, 1121		\protect	299, 304
per-mode	148	\providecommand	4, 5, 42
per-symbol	148	\ProvideDocumentCommand	69
\percent	146, 1102	\ProvidesExplPackage	23
\peta	145, 1053	\ps	182, 89
\pF	184, 70	\pV	183, 21
\pg	182, 35	Q	
\pgfqkeys	837	\qty	206, 80, 290, 327
\pico	145, 3, 16, 22, 36, 61, 72, 92, 1043	\qtylist	138, 301, 328
\planckbar	818	\qtyproduct	138, 309
\pm	38,	\qtyrange	138, 312, 329
150, 182, 60, 95, 281, 1915, 2124, 2125		qualifier-mode	148
\pmol	183, 14	qualifier-phrase	148
\power	147	quantity-product	107
prefix-mode	107	quark commands:	
prg commands:		\q_mark	393, 394, 401,
\prg_new_conditional:Npnn	1294, 2055	402, 639, 664, 686, 689, 742, 745,	
\prg_new_protected_conditional:Npnn	11, 32, 2038	754, 757, 762, 765, 767, 769, 776, 779	
\prg_replicate:nn		\q_nil 52, 67, 95, 96, 99, 119, 130, 134,	
191, 390, 391, 403, 418, 612,		139, 140, 145, 169, 205, 268, 272,	
687, 710, 843, 868, 989, 1317, 1326,		273, 289, 386, 387, 389, 483, 488,	
1424, 1489, 1531, 1584, 1607, 1943		579, 585, 604, 638, 640, 664, 666,	
\prg_return_false:		689, 745, 757, 765, 766, 769, 770,	
19, 48, 1301, 1309, 2048, 2073		779, 1853, 1855, 1857, 1877, 1937, 1965	
\prg_return_true:		\quark_if_nil:NTF	
18, 44, 1306, 2052, 2090		1861, 1871, 1881, 1885, 1889, 1952	
print-implicit-plus	42	\quark_if_nil:nTF	297
print-unity-mantissa	42	\quark_if_recursion_tail_stop:N	
		88, 168, 301, 340, 634, 774, 2084	
		\quark_if_recursion_tail_stop:n	
		250, 262, 307, 314, 416	

<code>\quark_if_recursion_tail_stop_-</code>		<code>round-precision</code>	42
<code>do:Nn</code>	186, 232, 286, 332, 379, 401, 485, 529, 540, 1043, 1052, 1068, 1082, 1100, 1121, 1139, 1169, 1215, 1260, 1291, 1305, 1348, 1356, 2073		
<code>\q_recursion_stop</code>	84, 149, 170, 226, 237, 242, 258, 270, 273, 282, 297, 327, 349, 375, 383, 411, 486, 501, 530, 541, 605, 624, 821, 831, 1038, 1046, 1057, 1061, 1071, 1087, 1103, 1149, 1195, 1202, 1211, 1224, 1256, 1287, 1299, 1339, 2069, 2080		
<code>\q_recursion_tail</code>	83, 149, 170, 226, 237, 242, 258, 270, 295, 327, 375, 411, 605, 624, 820, 830, 1038, 1046, 1057, 1061, 1071, 1087, 1103, 1149, 1195, 1202, 1211, 1224, 1256, 1287, 1299, 1339, 2068, 2080		
<code>\q_stop</code>	68, 96, 97, 99, 99, 106, 113, 136, 140, 141, 142, 142, 145, 187, 189, 189, 191, 205, 237, 239, 247, 249, 271, 272, 273, 275, 277, 282, 290, 303, 308, 359, 360, 360, 364, 384, 387, 388, 389, 395, 401, 402, 403, 485, 488, 509, 511, 582, 585, 643, 664, 666, 686, 689, 714, 718, 730, 737, 742, 742, 744, 745, 754, 757, 762, 762, 766, 767, 770, 774, 776, 779, 779, 789, 797, 801, 807, 809, 825, 827, 921, 934, 937, 945, 1625, 1626		
R			
<code>\radian</code>	146, 1075		
<code>\raiseto</code>	147, 116		
<code>range-exponents</code>	21		
<code>range-phrase</code>	22		
<code>range-units</code>	22		
<code>\relax</code>	56, 69, 70, 85		
<code>\renewcommand</code>	246		
<code>\RequirePackage</code>	3, 4, 8, 10, 39, 55, 61, 61, 231		
<code>reset-math-version</code>	94		
<code>reset-text-family</code>	94		
<code>reset-text-series</code>	94		
<code>reset-text-shape</code>	94		
<code>retain-explicit-plus</code>	42		
<code>retain-zero-uncertainty</code>	42		
<code>\rmdefault</code>	155		
<code>\rmfamily</code>	94		
<code>round-half</code>	42		
<code>round-minimum</code>	42		
<code>round-mode</code>	42		
<code>round-pad</code>	42		
S			
<code>\s</code>	182, 89		
<code>scan commands:</code>			
<code>\scan_stop:</code>	114, 122, 38, 123, 135		
<code>\scriptspace</code>	202, 225, 250		
<code>\second</code>	145, 146, 89, 90, 91, 92, 93, 94, 95, 1034		
<code>\selectfont</code>	93, 257, 394		
<code>\SendSettingsToPgf</code>	833		
<code>separate-uncertainty-units</code>	107		
<code>seq commands:</code>			
<code>\seq_clear:N</code>	95		
<code>\seq_const_from_clist:Nn</code>	325		
<code>\seq_count:N</code>	302		
<code>\seq_item:Nn</code>	313, 320, 325		
<code>\seq_map_function:NN</code>	90		
<code>\seq_map_indexed_function:NN</code>	335		
<code>\seq_map_inline:Nn</code>	41, 64, 179, 338, 350		
<code>\seq_new:N</code>	5, 21, 22		
<code>\seq_put_right:Nn</code>	29, 75, 127, 147, 174, 178		
<code>\seq_remove_all:Nn</code>	631, 636		
<code>\seq_use:Nnnn</code>	28		
<code>series-version-mapping</code>	95		
<code>\seriesdefault</code>	93, 248		
<code>\sfdefault</code>	156, 1166		
<code>\shapedefault</code>	93, 250		
<code>\SI</code>	315, 330, 647		
<code>\si</code>	314, 330, 638		
<code>\siemens</code>	146, 1075		
<code>\sievert</code>	146, 1075		
<code>\SIlist</code>	316, 330, 661		
<code>\sim</code>	2118		
<code>\SIrange</code>	323, 330, 661		
<code>\sisetup</code>	219		
<code>\SIUnitSymbolAngstrom</code>	702		
<code>\SIUnitSymbolArcminute</code>	702		
<code>\SIUnitSymbolArcsecond</code>	702		
<code>\SIUnitSymbolCelsius</code>	702		
<code>\SIUnitSymbolDegree</code>	702		
<code>\SIUnitSymbolMicro</code>	702		
<code>\SIUnitSymbolOhm</code>	702		
<code>siunitx commands:</code>			
<code>\siunitx_angle:n</code>	12, 45, 224		
<code>\siunitx_angle:nnn</code>	12, 45, 227, 228		
<code>\l_siunitx_bracket_ambiguous_-</code>			
<code>bool</code>	40		
<code>\siunitx_cell_begin:w</code>	117, 7, 214, 279		
<code>\siunitx_cell_end:</code>	117, 7, 58, 87, 216, 281		
<code>\siunitx_command_create:</code>	189, 37		
<code>\siunitx_complex_number:n</code>	197		

`\siunitx_complex_quantity:nn` .. 206
`\siunitx_compound_number:n`
 20, [85](#), 421, 470, 517
`\siunitx_compound_quantity:nn` ...
 20, [250](#), 428, 477, 524
`\siunitx_declare_power:NNn`
 143, [40](#), 64, 630, 635, 1103, 1104
`\siunitx_declare_prefix:Nn`
 143, 2, 3, 4, 5, 6, 7, 8, 9, [49](#), 626
`\siunitx_declare_prefix:Nnn`
 143, [49](#), 68, 138, 1043, 1044, 1045,
 1046, 1047, 1048, 1049, 1050, 1051,
 1052, 1053, 1054, 1055, 1056, 1057,
 1058, 1059, 1060, 1061, 1062, 1063
`\siunitx_declare_qualifier:Nn` ...
 143, [64](#), 72
`\siunitx_declare_unit:Nn` ... 143,
 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, 54, 55, 55,
 56, 57, 60, 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, 85, 86, 87, 88, 89, 90, 91,
 92, 93, 94, 95, 105, 158, 763, 776,
 1034, 1035, 1036, 1037, 1038, 1039,
 1040, 1041, 1042, 1064, 1065, 1066,
 1067, 1068, 1069, 1070, 1071, 1072,
 1073, 1074, 1075, 1076, 1077, 1078,
 1079, 1080, 1081, 1082, 1083, 1084,
 1085, 1086, 1087, 1088, 1089, 1090,
 1091, 1092, 1093, 1094, 1095, 1096,
 1097, 1098, 1099, 1100, 1101, 1102
`\siunitx_declare_unit:Nnn`
 143, 144, 58, 65, [70](#), 78, 192
`\siunitx_format_number:nN` 12
`\siunitx_if_number:nTF` 40, [2038](#)
`\siunitx_if_number_p:n` 40
`\siunitx_if_number_token:NTF` ...
 40, 171, [2055](#)
`\siunitx_if_number_token_p:N` . [2055](#)
`\l_siunitx_list_separator_final_-`
 tl 20, 299, 306, 321, [403](#), 437
`\l_siunitx_list_separator_pair_-`
 tl 20, 297, 304, 319, [403](#), 439
`\l_siunitx_list_separator_tl` ...
 20, 298, 305, 320, [403](#), 441
`\siunitx_number_adjust_exponent:Nn`
 39, 84, 232, [1612](#)
`\siunitx_number_adjust_exponent:nn`
 39, [1612](#)
`\l_siunitx_number_bracket_-`
 ambiguous_bool
 63, 260, [1636](#), 1641, 1736
`\l_siunitx_number_comparator_tl` . 40
`\l_siunitx_number_exponent_tl` ... 40
`\siunitx_number_format:nN`
 39, [15](#), 113, 173, 836
`\l_siunitx_number_input_comparator_-`
 tl 30, 251, 2061
`\l_siunitx_number_input_decimal_-`
 tl 40,
 29, 41, 391, 406, 453, 499, 562, 2059
`\l_siunitx_number_input_exponent_-`
 tl 30, 262, 270, 271, 2063
`\l_siunitx_number_input_sign_tl` .
 30, 310, [418](#), 513, 2066
`\siunitx_number_list:nn` . 20, 153, [417](#)
`\siunitx_number_normalize_-`
 symbols:N 40, [78](#), 132
`\siunitx_number_output:N`
 39, 23, 125, 769, [1701](#)
`\siunitx_number_output:n` ... 39, [1701](#)
`\siunitx_number_output:NN`
 38, 39, 52, 67, 119,
 129, 134, 169, 483, 579, 638, 640, [1701](#)
`\siunitx_number_output:nN` .. 39, [1701](#)
`\l_siunitx_number_output_-`
 decimal_tl .. 40, 282, 447, 465,
 513, 846, 849, [1637](#), 1679, 1823, 1826
`\siunitx_number_parse:nN`
 38, 39, [107](#), [117](#), 20, 50, 64, 107,
 [109](#), 152, 166, 376, 576, 620, 675, 2044
`\l_siunitx_number_parse_bool` ...
 39, [40](#), [10](#), [10](#), [11](#),
 18, 18, 50, 62, 96, 111, 263, 374, 2043
`\siunitx_number_process:N` 39
`\siunitx_number_process:NN`
 39, 21, 72, 87, 91, 110,
 167, 210, 233, 238, 243, 577, 633, [692](#)
`\siunitx_number_product:n` 20, [172](#), [466](#)
`\siunitx_number_range:nn` 20, 189, [513](#)
`\l_siunitx_number_sign_tl` 40
`\siunitx_prin_number:n` 94
`\siunitx_print....:n` 28, [93–95](#)
`\siunitx_print_match:n` 93, [80](#)
`\siunitx_print_math:n` 93, 83, [99](#)
`\siunitx_print_number:n`
 93, [60](#), 90, 114, 145, 210, 217,
 251, 274, 274, 275, 276, 278, 380,
 588, 591, 675, 704, 718, 727, 794, 837
`\siunitx_print_text:n` 93,
 84, 157, [241](#), 720, 725, 730, 736, 752
`\siunitx_print_unit:n` 93, [107](#), 47,
 [60](#), 82, 135, 156, 159, 224, 280, 644, 655

\siunitx_quantity_print:nn	143	__siunitx_angle_arc_print_-		
\siunitx_quantity:nn		auxii:w	172
	107 , 40 , 85 , 97 , 658	__siunitx_angle_arc_print_-		
\siunitx_quantity_list:nn		auxiii:n	172
	20 , 145 , 417 , 668	__siunitx_angle_arc_print_-		
\l_siunitx_quantity_prefix_mode_-			auxiv:NN	172
tl	9 , 65 , 184 , 188 , 265	__siunitx_angle_arc_print_-		
\siunitx_quantity_print:nn		auxv:w	172
	107 , 55 , 81 , 104 , 108 , 125 , 131 , 133 , 143 , 163 , 374	__siunitx_angle_arc_print_-		
\siunitx_quantity_product:nn	...		auxvi:n	172
	20 , 163 , 466	__siunitx_angle_arc_sign:nn	...	91
\siunitx_quantity_range:nnn	...		__siunitx_angle_arc_sign:nnn	66 , 91	
	20 , 181 , 513 , 677	\l__siunitx_angle_astronomy_bool		
\l_siunitx_range_phrase_tl	6 , 186
	21 , 311 , 313 , 324 , 501 , 532	\l__siunitx_angle_degrees_tl	...	
\l_siunitx_unit_font_tl	50 , 51 , 52 , 54 , 87 , 135
	143 , 36 , 79 , 88 , 91 , 122 , 321 , 334 , 829 , 841 , 852 , 865 , 936	__siunitx_angle_extract_-		
\siunitx_unit_format:nN	...	107 , 141 , 142 , 154 , 40 , 46 , 54 , 78 , 81 , 92 , 132 , 134 , 223 , 239 , 264 , 279 , 643 , 654	sign:nnnnnnnn	91
\siunitx_unit_format_combine_-			\l__siunitx_angle_fill_degrees_-		
exponent:nnN	...	142 , 75 , 132 , 195	bool	6
\siunitx_unit_format_extract_-			\l__siunitx_angle_fill_minutes_-		
prefixes:nN	...	142 , 80 , 132 , 218	bool	6
\siunitx_unit_format_multiply:nnN			\l__siunitx_angle_fill_seconds_-		
	142 , 132 , 244	bool	6
\siunitx_unit_format_multiply_-			\l__siunitx_angle_force_arc_bool		
combine_exponent:nnnN	142 , 132 , 203			6 , 47
\siunitx_unit_format_multiply_-			\l__siunitx_angle_force_decimal_-		
extract_prefixes:nnN	142 , 132 , 224		bool	6 , 61
\l_siunitx_unit_fraction_tl	...		\l__siunitx_angle_marker_box	...	
	144 , 259 , 318 , 510 , 983		170 , 214 , 228 , 232 , 238 , 240 , 244 , 249	
\siunitx_unit_options_apply:n	...		\l__siunitx_angle_minutes_tl	87 , 136	
	143 , 144 , 43 , 77 , 89 , 95 , 132 , 143 , 161 , 179 , 204 , 367 , 666 , 675	\l__siunitx_angle_product_tl	6 , 278	
\siunitx_unit_pdfstring_context:			\l__siunitx_angle_seconds_tl	87 , 137	
	144 , 346 , 347	\l__siunitx_angle_separator_tl	6 , 195	
\siunitx_unit_power_set:NnN	...	153	__siunitx_angle_sign:nnnnnnn	...	91
\l_siunitx_unit_seq	..	144 , 22 , 75 , 90	\l__siunitx_angle_sign_tl	
\l_siunitx_unit_symbolic_seq	90 , 101 , 105 , 114 , 163
	144 , 21 , 29 , 41 , 64 , 179 , 350 , 631 , 636	\l__siunitx_angle_symbol_degree_-		
siunitx internal commands:			tl	6 , 175
__siunitx_angle:n	291 , 358	\l__siunitx_angle_symbol_minute_-		
__siunitx_angle:nnn	105 , 221	tl	6 , 177
__siunitx_angle:w	358	\l__siunitx_angle_symbol_second_-		
__siunitx_angle_arc_convert:n	..	45	tl	6 , 179
__siunitx_angle_arc_print:nnn	..		\l__siunitx_angle_tmp_bool	
	53 , 134 , 172		3 , 208 , 209 , 256
__siunitx_angle_arc_print_-			\l__siunitx_angle_tmp_dim	
auxi:nnn	172		3 , 241 , 263 , 265
__siunitx_angle_arc_print_-			\l__siunitx_angle_tmp_tl	
auxii:nw	187 , 204	..	3 , 152 , 153 , 159 , 223 , 224 , 279 , 280	
			\l__siunitx_angle_unit_box	
				170 , 219 , 229 , 233 , 237 , 246
			__siunitx_bookmark_cmd:Nn	...	284

__siunitx_bookmark_cmd:Nnn	__siunitx_compound_format_-
. 284, 290, 291, 292, 293, 294, 301,	combine-exponent_aux:n 180
308, 309, 310, 312, 314, 315, 316, 323	__siunitx_compound_format_-
\c_siunitx_bookmark_seq . . . 325, 338	extract-exponent:n 180
\l_siunitx_column_type_tl . . 46, 274	__siunitx_compound_format_-
__siunitx_command_create: 37	extract-exponent:nn 180
__siunitx_command_create:N 37	__siunitx_compound_format_-
\l_siunitx_command_create_bool .	extract-exponent_aux:n 180
. 4, 39	__siunitx_compound_format_-
\l_siunitx_command_optarg_bool .	input:n 180
. 4, 69, 73	__siunitx_compound_format_-
\l_siunitx_command_overwrite_-	input:nn 241
bool 4, 62	__siunitx_compound_format_-
\l_siunitx_command_prespace_-	units:nn 109, 180
bool 4, 80	__siunitx_compound_parsed:n 129, 162
\l_siunitx_command_tmp_tl . 3, 79, 83	__siunitx_compound_print:N
\l_siunitx_command_xspace_bool 90, 268, 275, 278, 283
. 4, 60, 87	__siunitx_compound_print:nnN . . 283
\l_siunitx_compound_bracket_-	__siunitx_compound_print:nnnN . 283
close_tl 14, 276, 292, 398	__siunitx_compound_print_aux:n 283
\l_siunitx_compound_bracket_-	__siunitx_compound_print_aux:nn 283
open_tl 14, 274, 290, 396	__siunitx_compound_print_-
\l_siunitx_compound_count_int . .	quantity:n 268, 279, 283
. 11, 331, 354, 359	__siunitx_compound_print_-
\l_siunitx_compound_end_tl	separator:n 283
. 9, 333, 364	\l_siunitx_compound_separator_-
\l_siunitx_compound_exp_-	final_tl 18, 357
bracket_bool 18, 254, 286	\l_siunitx_compound_separator_-
\l_siunitx_compound_exp_-	pair_tl 18, 322
combine_bool . 18, 112, 209, 234, 256	\l_siunitx_compound_separator_-
\l_siunitx_compound_exp_tl	text_bool 18, 378
. 8, 148, 266, 272, 287, 293, 297	\l_siunitx_compound_separator_-
__siunitx_compound_extract_-	tl 18, 352, 370
exp:nN 180	\l_siunitx_compound_start_tl . . .
__siunitx_compound_extract_- 9, 332, 349
exp:nnnnnnN 180	\l_siunitx_compound_tmp_fp
__siunitx_compound_extract_- 4, 212, 213, 230, 232
exponents: 114, 131	\l_siunitx_compound_tmp_seq . . .
__siunitx_compound_extract_- 4, 95, 127,
exponents_auxi:w 131	147, 174, 178, 302, 313, 320, 325, 336
__siunitx_compound_extract_-	\l_siunitx_compound_tmp_tl
exponents_auxii:nw 131 4, 107,
__siunitx_compound_extract_-	110, 118, 120, 121, 124, 127, 133,
exponents_auxiii:nnnnnn 131	136, 166, 167, 168, 169, 170, 171,
\l_siunitx_compound_first_tl . . .	173, 174, 210, 231, 232, 233, 238, 243
. 7, 110, 119,	__siunitx_compound_uncert_-
125, 134, 150, 210, 212, 233, 238, 243	bracket:N 121, 171, 383
__siunitx_compound_format:n . . . 85	__siunitx_compound_uncert_-
__siunitx_compound_format:nn 85, 262	bracket:nnw 383
__siunitx_compound_format:nnn . . 85	__siunitx_compound_uncert_-
__siunitx_compound_format_-	bracket:w 383
combine-exponent:n 180	\l_siunitx_compound_unit_bool . .
__siunitx_compound_format_- 12, 88, 108, 116, 164, 261
combine-exponent:nn 180	

\l__siunitx_compound_unit_-	
bracket_bool	18 , 253 , 258 , 271
\l__siunitx_compound_unit_power_-	
bool	22 , 57 , 63 , 69 , 75 , 81 , 182
\l__siunitx_compound_unit_-	
repeat_bool	18 , 255 , 259 , 267
\l__siunitx_compound_unit_tl . . .	
.	12 , 213 , 230 , 239 , 244 , 264 , 374
__siunitx_compound_unparsed:n . .	
.	103 , 162
__siunitx_declare_column:Nnn . .	232
__siunitx_emulation_non_latin:n	
.	681 , 711 , 713 , 715 , 721 , 726 , 731 , 741 , 757 , 824 , 828
__siunitx_emulation_non_-	
latin:nnnn	681
__siunitx_emulation_tmp:w	
.	772 , 788 , 790 , 824 , 827
__siunitx_list_aux:	417
__siunitx_list_count:n	375
__siunitx_list_count:w	375
\l__siunitx_list_exp_tl	403 , 435
\l__siunitx_list_units_tl . .	403 , 443
__siunitx_list_use:nnnnn	
.	296 , 303 , 318 , 375
__siunitx_list_use_aux:nnnnn . .	375
__siunitx_list_use_auxi:nw	
.	388 , 389 , 398
__siunitx_list_use_auxii:w	375
__siunitx_list_use_auxiii:nnw . .	375
__siunitx_list_use_auxiiii:nnw .	375
__siunitx_load_check:	25
__siunitx_load_check:n	28 , 36 , 40
__siunitx_number_adjust_exp:nn	1612
__siunitx_number_adjust_-	
exp:nnnnnnnn	1612
__siunitx_number_adjust_exp:nW	
.	1612
\l__siunitx_number_arg_tl	
.	50 , 53 , 70 , 125 , 131 , 132 , 134 , 135 , 247 , 254 , 257 , 273 , 288 , 300 , 374 , 509 , 515 , 523
\l__siunitx_number_bracket_-	
close_tl	1632 , 1750 , 1801
\l__siunitx_number_bracket_-	
negative_bool	1638 , 1776
\l__siunitx_number_bracket_open_-	
tl	1632 , 1748 , 1799
\l__siunitx_number_comparator_tl	
.	71 , 253 , 256 , 357
__siunitx_number_digits:NN	701 , 966
__siunitx_number_digits:Nn . . .	966
__siunitx_number_digits:nn . . .	966
__siunitx_number_digits:nnnnnnn	966
__siunitx_number_digits_S:n . .	966
__siunitx_number_digits_-	
uncert:nn	982 , 991
__siunitx_number_digits_uncert_-	
S:n	996
__siunitx_number_drop_exponent:NN	
.	699 , 1001
__siunitx_number_drop_exponent:nnnnnnn	
.	1001
\l__siunitx_number_drop_exponent_-	
bool	637 , 1003
__siunitx_number_drop_uncertainty:NN	
.	697 , 1011
__siunitx_number_drop_uncertainty:nnnnnnn	
.	1011
\l__siunitx_number_drop_uncertainty_-	
bool	637 , 1013
\l__siunitx_number_drop_zero_-	
decimal_bool	637 , 1596
\l__siunitx_number_explicit_-	
plus_bool	30 , 319 , 518
__siunitx_number_exponent:NN . . .	
.	714 , 718
\l__siunitx_number_exponent_-	
base_tl	1638 , 2014
__siunitx_number_exponent_-	
engineering:nnnnnnn	718
__siunitx_number_exponent_-	
engineering:nnNw	718
__siunitx_number_exponent_-	
engineering_0:nnnn	718
__siunitx_number_exponent_-	
engineering_1:nnnn	718
__siunitx_number_exponent_-	
engineering_2:nnnn	718
__siunitx_number_exponent_-	
engineering_aux:nnnnnnn	718
__siunitx_number_exponent_-	
engineering_uncert:nn	718
__siunitx_number_exponent_-	
engineering_uncert_S:nnn	718
__siunitx_number_exponent_-	
finalise:n	
.	718 , 1276 , 1569 , 1574 , 1579 , 1592
__siunitx_number_exponent_-	
fixed:nnnnnnn	718
__siunitx_number_exponent_-	
fixed:nnnnnnnn	718
\l__siunitx_number_exponent_-	
fixed_int	637 , 736 , 744
__siunitx_number_exponent_-	
input:nnnnnnn	718

\l_siunitx_number_exponent_- mode_tl 637 , 723 , 727 , 1236 , 1525 , 1543 , 1551 , 1560	__siunitx_number_if_token_- auxiii:NN 2055
\l_siunitx_number_exponent_- product_tl 1638 , 2010	\l_siunitx_number_implicit_- plus_bool ... 863 , 1638 , 1770 , 2029
__siunitx_number_exponent_- scientific:nnnnnn 718	\l_siunitx_number_input_digit_- tl 30 , 236 , 341 , 385 , 403 , 487 , 542 , 2062
__siunitx_number_exponent_- scientific:nnnnnnnn 718	\l_siunitx_number_input_ignore_- tl 30 , 120 , 133 , 2064
__siunitx_number_exponent_- scientific:nnnw 718	\l_siunitx_number_input_tl 75 , 131 , 169
__siunitx_number_exponent_- shift:nnn 718 , 1269	\l_siunitx_number_input_uncert_- close_tl ... 30 , 531 , 556 , 569 , 2060
__siunitx_number_exponent_- shift_down:nnn 718	\l_siunitx_number_input_uncert_- open_tl 30 , 413 , 2065
__siunitx_number_exponent_- shift_down:nnnw 718	\l_siunitx_number_input_uncert_- sign_tl 30 , 157 , 2067
__siunitx_number_exponent_- shift_down:nw 718	\l_siunitx_number_min_decimal_- int 637 , 980 , 999
__siunitx_number_exponent_- shift_uncert:nw 718	\l_siunitx_number_min_integer_- int 637 , 975
__siunitx_number_exponent_- shift_uncert_S:nnnn 718	\l_siunitx_number_negative_- color_tl 1638 , 1757 , 1758
__siunitx_number_exponent_- shift_up:nnn 718	__siunitx_number_normalize_- aux:nN 78
__siunitx_number_exponent_- shift_up:nnw 718	__siunitx_number_normalize_- aux:NnN 81 , 86 , 90
__siunitx_number_exponent_- shift_up_aux:nnn 718	__siunitx_number_normalize_- minus:N 80 , 103
\l_siunitx_number_exponent_tl 72 , 264 , 298 , 313 , 321 , 322 , 333 , 343 , 360	__siunitx_number_normalize_- sign:N 78
__siunitx_number_exponent_- uncert:n 718	\c_siunitx_number_normalize_tl . 78
__siunitx_number_expression:n 30 , 128	__siunitx_number_output:Nn .. 1701
\l_siunitx_number_expression_- bool 30 , 127	__siunitx_number_output:nn .. 1701
\l_siunitx_number_flex_tl .. 48 , 73 , 143 , 152 , 156 , 178 , 465 , 467 , 503	__siunitx_number_output:nnnnnn 1701
\l_siunitx_number_group_- decimal_bool 859 , 1638	__siunitx_number_output_- bracket:nn 1701
\l_siunitx_number_group_- integer_bool 860 , 1638	__siunitx_number_output_- bracket:w 1701
\l_siunitx_number_group_- minimum_int 857 , 1638 , 1837	__siunitx_number_output_color:n 1721 , 1753
\l_siunitx_number_group_- separator_tl 852 , 855 , 1638 , 1866 , 1869 , 1894	__siunitx_number_output_- comparator:nn 1701
__siunitx_number_if_token_- auxi:NN 2055	__siunitx_number_output_- decimal:nn 1701
__siunitx_number_if_token_- auxii:NN 2055	__siunitx_number_output_- decimal_aux:n 1701
	__siunitx_number_output_- decimal_loop:NNNN 1701
	__siunitx_number_output_- digits:nn 1701
	__siunitx_number_output_end: . 1701

\l__siunitx_number_output_exp_- marker_tl	1638, 1994, 2021	__siunitx_number_output_- uncertainty:nnn	1701
__siunitx_number_output_- exponent:nnnn	1701	__siunitx_number_output_- uncertainty_unaligned:n . . .	1701
__siunitx_number_output_- exponent_auxi:nnnn	1701	\l__siunitx_number_outputted_tl	9, 22, 25, 27
__siunitx_number_output_- exponent_auxii:nnnn	1701	__siunitx_number_parse:nN . . .	109
__siunitx_number_output_- exponent_auxiii:nn	1701	__siunitx_number_parse_check:	137, 141
__siunitx_number_output_- integer:nnn	1701	__siunitx_number_parse_combine_- uncert:	159, 174
__siunitx_number_output_- integer_aux:n	1701	__siunitx_number_parse_combine_- uncert_auxi:nnnnnnnn	174
__siunitx_number_output_- integer_aux_0:n	1701	__siunitx_number_parse_combine_- uncert_auxii:nnnnnn	174
__siunitx_number_output_- integer_aux_1:n	1701	__siunitx_number_parse_combine_- uncert_auxiii:nnnnnn	174
__siunitx_number_output_- integer_aux_2:n	1701	__siunitx_number_parse_combine_- uncert_auxiv:nnnn	174
__siunitx_number_output_- integer_first:nnNN	1701	__siunitx_number_parse_combine_- uncert_auxv:w	174
__siunitx_number_output_- integer_loop:NNNN	1701	__siunitx_number_parse_combine_- uncert_auxvi:w	174
__siunitx_number_output_sign:N	1701	__siunitx_number_parse_comparator:	136, 244
__siunitx_number_output_sign:nN	1701	__siunitx_number_parse_comparator_- aux:Nw	244
__siunitx_number_output_- sign:nnn	1701	__siunitx_number_parse_exponent:	260, 525
__siunitx_number_output_sign_- brackets:w	1701	__siunitx_number_parse_exponent_- auxi:w	260
__siunitx_number_output_sign_- color:w	1701	__siunitx_number_parse_exponent_- auxii:nn	260
\l__siunitx_number_output_- uncert_close_tl	1638, 1925	__siunitx_number_parse_exponent_- auxiii:Nw	260
\l__siunitx_number_output_- uncert_open_tl	1638, 1923	__siunitx_number_parse_exponent_- auxiv:nn	260
__siunitx_number_output_uncert_- S:nnnw	1701	__siunitx_number_parse_exponent_- check:N	260
__siunitx_number_output_uncert_- S:nnw	1701	__siunitx_number_parse_exponent_- cleanup:N	260
__siunitx_number_output_uncert_- S_aux:nnn	1701	__siunitx_number_parse_exponent_- cleanup:wN	346, 348
__siunitx_number_output_uncert_- S_aux:nnnw	1933, 1950, 1957, 1964	__siunitx_number_parse_exponent_- zero_test:N	260
__siunitx_number_output_uncert_- S_aux:nnw	1955, 1965	__siunitx_number_parse_finalise:	172, 351
__siunitx_number_output_uncert_- S_compact-marker:nn	1701	__siunitx_number_parse_finalise:nw	351
__siunitx_number_output_uncert_- S_compact:nn	1701	__siunitx_number_parse_loop:	266, 306, 369
__siunitx_number_output_uncert_- S_full:nn	1701	__siunitx_number_parse_loop_- after_decimal:NNN	369

__siunitx_number_parse_loop_- break:wN 369 , 530, 532, 541, 564, 574, 601, 635	\l__siunitx_number_partial_tl 76 , 371 , 433 , 434, 437, 447, 471, 472, 475, 479, 489, 547, 578, 589, 590, 600, 611, 612
__siunitx_number_parse_loop_- first:N 369	__siunitx_number_process:nnnnnnnNN 692
__siunitx_number_parse_loop_- first:NNN 372 , 377 , 466	__siunitx_number_round:NN 712 , 715 , 1023
__siunitx_number_parse_loop_- main:NNNN 369	__siunitx_number_round:nnn 1035 , 1360 , 1399 , 1409 , 1480 , 1504 , 1512
__siunitx_number_parse_loop_- main_decimal:NN 369	__siunitx_number_round_auxi:nnnN 1035
__siunitx_number_parse_loop_- main_digit:NNNN 369	__siunitx_number_round_auxii:nnnN 1035
__siunitx_number_parse_loop_- main_end:NN 369	__siunitx_number_round_auxiii:nnnN 1035
__siunitx_number_parse_loop_- main_sign:NNN 369	__siunitx_number_round_auxiv:nnN 1035
__siunitx_number_parse_loop_- main_store:NNN 369 , 591	__siunitx_number_round_auxiv:nnnN 1056 , 1070 , 1080 , 1093
__siunitx_number_parse_loop_- main_uncert:NNN 369	__siunitx_number_round_auxv:nnN 1035
__siunitx_number_parse_loop_- root_swap:NNwNN 369	__siunitx_number_round_auxvi:nnN 1035
__siunitx_number_parse_sign: 258 , 506	__siunitx_number_round_auxvi:nnN 1086 , 1096 , 1102 , 1119
__siunitx_number_parse_sign_- aux:Nw 506	__siunitx_number_round_auxvii:nnN 1035
__siunitx_number_parse_uncert:NN 460 , 527	__siunitx_number_round_auxviii:nnN 1035
__siunitx_number_parse_uncert:NNNN 527	__siunitx_number_round_engineering:nn 1035
__siunitx_number_parse_uncert_- after:N 527	__siunitx_number_round_engineering:nnN 1035
__siunitx_number_parse_uncert_- auxi:NN 527	__siunitx_number_round_engineering:NNNNn 1035
__siunitx_number_parse_uncert_- auxii:N 527	__siunitx_number_round_figures:nnnnnnn 1320
__siunitx_number_parse_uncert_- auxii:NN 527	__siunitx_number_round_figures_- aux:nnnnnnn 1320
__siunitx_number_parse_uncert_- auxiii:N 558 , 571 , 576	__siunitx_number_round_figures_- count:nnN 1320
__siunitx_number_parse_uncert_- extend:nnnN 527	__siunitx_number_round_figures_- count:nnnN 1320
__siunitx_number_parse_uncert_- marker:N 527	__siunitx_number_round_final:nn 1035
__siunitx_number_parse_uncert_- marker:nnnN 596 , 597	__siunitx_number_round_final_- decimal:nnw 1035
__siunitx_number_parse_uncert_- marker:nNw 602 , 604	__siunitx_number_round_final_- integer:nnw 1035
\l__siunitx_number_parsed_tl 49 , 20 , 21 , 23 , 74 , 119 , 139 , 148 , 160 , 162 , 164 , 178 , 213 , 215 , 265 , 302 , 305 , 314 , 324 , 350 , 353 , 355 , 358 , 373 , 504 , 519 , 520 , 522 , 524 , 2044 , 2045	__siunitx_number_round_final_- output:nn 1035
	__siunitx_number_round_final_- shift:nn 1035

_siunitx_number_round_final_- shift:Nw	1035	_siunitx_number_round_scientific:nn	1267
_siunitx_number_round_final_- significant:N	1035	_siunitx_number_round_scientific:nn	1035
_siunitx_number_round_final_- significant:n	1035	_siunitx_number_round_truncate:n	1035
_siunitx_number_round_final_- significant:w	1035	_siunitx_number_round_truncate:nnN	1035
_siunitx_number_round_final_- significant:N ...	1210 , 1213 , 1217	_siunitx_number_round_truncate_- direct:n	1035
_siunitx_number_round_final_- significant:n	1205 , 1208	_siunitx_number_round_uncertainty:nnn	1449
_siunitx_number_round_final_- significant:w	1220 , 1223	_siunitx_number_round_uncertainty:nnnn	1449
_siunitx_number_round_fixed:nn	1035	_siunitx_number_round_uncertainty:nnnnn	1449
\l_siunitx_number_round_half_- even_bool	637 , 1110 , 1128	_siunitx_number_round_uncertainty:nnnnnnn	1449
_siunitx_number_round_if_- half:N	1294	_siunitx_number_round_uncertainty_- end:nn	1449
_siunitx_number_round_if_- half:n	1294	_siunitx_number_round_uncertainty_- engineering:nnn	1449
_siunitx_number_round_if_half_- p:n	1112 , 1130 , 1294	_siunitx_number_round_uncertainty_- engineering_2:n	1449
_siunitx_number_round_input:nn	1035	_siunitx_number_round_uncertainty_- engineering_3:n	1449
\l_siunitx_number_round_min_tl	672 , 680 , 1427 , 1435	_siunitx_number_round_uncertainty_- engineering_4:n	1449
\l_siunitx_number_round_mode_tl	637 , 839 , 1028 , 1233 , 1280	_siunitx_number_round_uncertainty_- fixed:nnn	1449
_siunitx_number_round_none:nnnnnnn	1023	_siunitx_number_round_uncertainty_- input:nnn	1449
_siunitx_number_round_pad:nnn	1311 , 1365 , 1394	_siunitx_number_round_uncertainty_- scientific:nnn	1449
\l_siunitx_number_round_pad_- bool	637 , 1316	_siunitx_number_round_uncertainty_- shift:nn	1449
_siunitx_number_round_places:nnnnnnn	1372	_siunitx_number_round_uncertainty_- shift:nnnnn	1499 , 1509
_siunitx_number_round_places_- decimal:nn	1372	_siunitx_number_round_uncertainty_- shift:nnnnnn	1449
_siunitx_number_round_places_- end:nnn	1273 , 1372 , 1553	_siunitx_number_round_uncertainty_- shift:nnnw	1449
_siunitx_number_round_places_- finalise:n	1372	_siunitx_number_round_uncertainty_- shift_aux:nnnnn	1449
_siunitx_number_round_places_- finalise:nnnnn	1372	_siunitx_number_round_uncertainty_- simple:nnnnn	1487 , 1502
_siunitx_number_round_places_- finalise:nnnnnnn	1372	_siunitx_number_round_uncertainty_- simple:nnnnnnn	1449
_siunitx_number_round_places_- integer:nn	1372	_siunitx_number_set_round_- min:n	660 , 673
\l_siunitx_number_round_- precision_int	637 , 844 , 1255 , 1335 , 1358 , 1361 , 1366 , 1379 , 1392 , 1395 , 1402 , 1412 , 1453 , 1473	_siunitx_number_set_round_- min:nnnnnnn	673
		\l_siunitx_number_tight_bool	1638 , 1792 , 2007

\l__siunitx_number_tmp_tl	__siunitx_option_table_sign-exponent:nnnnnnnn
..... 8, 155, 158, 451
269, 274, 280, 281, 289, 290, 675, 676	__siunitx_option_table_sign-mantissa:nnnnnnnn
__siunitx_number_token_auxi:NN 451
..... 2058, 2071, 2075	__siunitx_print_aux:nn
__siunitx_number_token_auxii:NN 60
..... 2074, 2077	__siunitx_print_convert_-
__siunitx_number_token_auxiii:NN	series:n
..... 2079, 2082, 2093 99
\l__siunitx_number_uncert_mode_-	__siunitx_print_extract_-
tl	series:Nw
1638, 1735, 1912, 1924 99
\l__siunitx_number_uncert_-	__siunitx_print_math_aux:N
separator_tl 99
1638, 1922	__siunitx_print_math_aux:Nn ...
\l__siunitx_number_unity_- 99
mantissa_bool ...	__siunitx_print_math_aux:w
1638, 1808, 2004 99
\l__siunitx_number_valid_tl ..	__siunitx_print_math_auxi:n ...
2037 99
\l__siunitx_number_validate_bool	__siunitx_print_math_auxii:n ...
..... 77, 166, 2042 99
__siunitx_number_zero_decimal:NN	__siunitx_print_math_auxiii:n ..
..... 700, 1594 99
__siunitx_number_zero_decimal:nnnnnnn	__siunitx_print_math_auxiv:n ...
..... 1594 99
\l__siunitx_number_zero_exponent_-	__siunitx_print_math_auxv:n ...
bool 99
1638, 1740, 1813, 1991	\l__siunitx_print_math_family_-
\l__siunitx_number_zero_uncert_-	bool
bool 7, 151
30, 234, 583	\l__siunitx_print_math_font_bool
__siunitx_option_deprecated:nn 7, 166
..... 11, 76, 82, 88, 108, 122, 131,	__siunitx_print_math_script:n ..
137, 151, 157, 222, 228, 241, 247, 99
268, 274, 286, 292, 301, 310, 316,	\l__siunitx_print_math_series_-
352, 358, 364, 370, 516, 522, 530,	bool
545, 563, 569, 577, 583, 589, 610, 616 7, 102
__siunitx_option_deprecated:nnn	__siunitx_print_math_sub:n
..... 11, 99
42, 50, 58, 66, 97, 323, 333, 341,	__siunitx_print_math_super:n ...
377, 388, 396, 413, 507, 554, 595, 602 99
__siunitx_option_removed:n	__siunitx_print_math_text:n ...
..... 22, 36, 172, 196, 216, 99
261, 280, 282, 305, 330, 348, 513, 527	__siunitx_print_math_version:nn
__siunitx_option_table_comparator:nnnnnnn 99
..... 451	\l__siunitx_print_math_version_-
__siunitx_option_table_comparator:nnnnnnnn	bool
..... 466 7, 128
__siunitx_option_table_figures-decimal:nnnnnnnn	\l__siunitx_print_number_color_-
..... 451	tl
__siunitx_option_table_figures-exponent:nnnnnnnn 7
..... 451	\l__siunitx_print_number_mode_tl .
__siunitx_option_table_figures-integer:nnnnnnnn 7
..... 451	__siunitx_print_print_replace_-
__siunitx_option_table_figures-uncertainty:nnnnnnnn	frac:n
..... 451	310, 311, 312, 313, 316
__siunitx_option_table_format:n	__siunitx_print_replace_font:N .
451, 489, 491, 493, 495, 497, 499, 501 86, 201, 223, 278
	\c__siunitx_print_series_b_tl ...
 97
	\c__siunitx_print_series_c_tl ...
 95
	\c__siunitx_print_series_ecl_tl .
 95
	\c__siunitx_print_series_ex_tl ..
 95
	\c__siunitx_print_series_l_tl ...
 97
	\c__siunitx_print_series_m_tl ...
 97
	\c__siunitx_print_series_sc_tl ..
 95
	\c__siunitx_print_series_sx_tl ..
 95
	\c__siunitx_print_series_uc_tl ..
 95
	\c__siunitx_print_series_ux_tl ..
 95
	\c__siunitx_print_series_x_tl ...
 95
	__siunitx_print_text_family_-
	bool
 28, 245, 253
	__siunitx_print_text_font_tl ..
 7, 258
	__siunitx_print_text_fraction:Nnn
 241
	__siunitx_print_text_replace:N
 241

__siunitx_print_text_replace:n	241	\l__siunitx_quantity_bracket_-	
__siunitx_print_text_replace:NnN		close_tl	5 , 113
.....	241	\l__siunitx_quantity_bracket_-	
__siunitx_print_text_replace:Nnnn		open_tl	5 , 111
.....	241	\l__siunitx_quantity_break_bool	
__siunitx_print_text_replace:Nnnnn		9 , 153
.....	268 , 305	__siunitx_quantity_extract_-	
__siunitx_print_text_replace_-		exp:nNN	73 , 135
frac:n	241	__siunitx_quantity_extract_-	
__siunitx_print_text_scripts:	241	exp:nnnnnnnNN	135
__siunitx_print_text_scripts:NnN		__siunitx_quantity_non_latin:n	
.....	241	171 , 193
__siunitx_print_text_scripts_-		__siunitx_quantity_non_latin:nnnn	
one:Nn	357 , 364 , 395	171
__siunitx_print_text_scripts_-		\l__siunitx_quantity_number_tl	
one:NnN	241	38 , 53 , 56 , 64 , 66 , 67 , 68 , 72 , 74 , 82 , 85 , 87 , 91
__siunitx_print_text_scripts_-		__siunitx_quantity_parsed:nn	40
two:n	241	__siunitx_quantity_parsed_-	
__siunitx_print_text_scripts_-		aux:nnn	40
two:nn	241	__siunitx_quantity_parsed_-	
__siunitx_print_text_scripts_-		aux:nnnn	40
two:NnNn	241	__siunitx_quantity_parsed_-	
\l__siunitx_print_text_series_-		aux:nnnw	40
bool	30 , 247 , 254	__siunitx_quantity_parsed_aux:w	40
\l__siunitx_print_text_shape_-		__siunitx_quantity_parsed_-	
bool	32 , 249 , 255	combine-exponent:n	40
__siunitx_print_text_sub:n	241	__siunitx_quantity_parsed_-	
__siunitx_print_text_super:n	241	input:n	40
\l__siunitx_print_tmp_tl	6 , 105 , 107 , 109 , 200 , 201 , 202 , 205 , 208 , 222 , 223 , 224 , 233 , 234 , 236 , 270 , 271 , 272 , 331 , 332 , 333 , 367 , 368 , 370	\l__siunitx_quantity_product_tl	
\l__siunitx_print_unit_color_tl	7	9 , 152
\l__siunitx_print_unit_mode_tl	7	\l__siunitx_quantity_tmp_fp	
\l__siunitx_print_version_b_tl	48	3 , 74 , 76 , 81 , 85
\l__siunitx_print_version_eb_tl	48	\l__siunitx_quantity_tmp_tl	3
\l__siunitx_print_version_el_tl	48	\l__siunitx_quantity_uncert_-	
\l__siunitx_print_version_l_tl	48	bracket_bool	9 , 106
\l__siunitx_print_version_m_tl	48	\l__siunitx_quantity_uncert_-	
\l__siunitx_print_version_sb_tl	48	repeat_bool	9 , 119
\l__siunitx_print_version_sl_tl	48	\l__siunitx_quantity_unit_tl	
\l__siunitx_print_version_ub_tl	48	38 , 46 , 47 , 54 , 56 , 76 , 81 , 92 , 104 , 116 , 125 , 131 , 133
\l__siunitx_print_version_ul_tl	48	__siunitx_range_aux:	513
__siunitx_product_aux:	466	\l__siunitx_range_exp_tl	501 , 531
__siunitx_product_aux:n	466	\l__siunitx_range_units_tl	501 , 534
\l__siunitx_product_exp_tl	447 , 490	__siunitx_symbol_deal_with_utf:	
\l__siunitx_product_phrase_bool		25 , 31
.....	447 , 482 , 495	__siunitx_symbol_if_replace:Nn	32
\l__siunitx_product_phrase_tl		__siunitx_symbol_if_replace:NnTF	
.....	447 , 483	32 , 53 , 58 , 63 , 97 , 136
\l__siunitx_product_symbol_tl		__siunitx_symbol_non_latin:n	
.....	447 , 484	10 , 36 , 71 , 77 , 91 , 129 , 145 , 159
\l__siunitx_product_units_tl	447 , 496	__siunitx_symbol_non_latin:nnnn	10
		\l__siunitx_symbol_tmp_tl	
		99 , 110 , 119 , 122

<code>\l__siunitx_symbol_tmpa_tl</code>	<code>__siunitx_table_collect_begin:</code> ..
..... 3 , 35 , 41 , 77 , 78 , 79 , 82 11 , 24
<code>\l__siunitx_symbol_tmpb_tl</code>	<code>__siunitx_table_collect_begin:N</code> 130
..... 3 , 40 , 41 , 81 , 82	<code>__siunitx_table_collect_begin:w</code> 24
<code>\l__siunitx_table_after_box</code>	<code>__siunitx_table_collect_end:</code> 19 , 110
..... 300 , 302 , 528 ,	<code>__siunitx_table_collect_end:n</code> . 110
556 , 558 , 562 , 569 , 572 , 600 , 645 , 658	<code>__siunitx_table_collect_end:w</code> . 110
<code>\l__siunitx_table_after_model_tl</code>	<code>__siunitx_table_collect_end-</code>
..... 357 , 370 , 441 , 644	aux:n 110
<code>\l__siunitx_table_after_tl</code>	<code>__siunitx_table_collect_group:n</code> 39
..... 107 , 118 , 125 , 169	<code>__siunitx_table_collect_loop:</code> ..
<code>\l__siunitx_table_align_after-</code> 31 , 38 , 39
bool 291 , 472 , 532 , 648	<code>__siunitx_table_collect_relax:N</code> 39
<code>__siunitx_table_align_auxi:nn</code> . 210	<code>__siunitx_table_collect-</code>
<code>__siunitx_table_align_auxii:nn</code> 210	search:NnTF 39
<code>\l__siunitx_table_align_before-</code>	<code>__siunitx_table_collect_search-</code>
bool 311 , 471 , 532 , 668 , 708	aux:Nnn 39
<code>__siunitx_table_align_center:n</code> 210	<code>\l__siunitx_table_collect_tl</code> ...
<code>\l__siunitx_table_align_comparator-</code> 23 , 27 , 47 , 64 , 113 , 115 , 138
bool 532 , 693	<code>__siunitx_table_collect_token:N</code> 39
<code>\l__siunitx_table_align_exponent-</code>	<code>__siunitx_table_collect_token-</code>
bool 532 , 786	aux:N 39
<code>__siunitx_table_align_left:n</code> .. 210	<code>__siunitx_table_color_check:N</code> ..
<code>\l__siunitx_table_align_mode_tl</code> 270 , 580
..... 340 , 439 , 443 , 546	<code>__siunitx_table_color_check:Nnw</code> 270
<code>__siunitx_table_align_none:n</code> .. 210	<code>__siunitx_table_color_check:w</code> ..
<code>\l__siunitx_table_align_number-</code> 270 , 666
tl 340 , 520 , 653 , 833	<code>\l__siunitx_table_column_width-</code>
<code>__siunitx_table_align_right:n</code> . 210	dim 198 , 219 , 226
<code>\l__siunitx_table_align_text_tl</code> .	<code>\l__siunitx_table_decimal_box</code> ...
..... 250 , 260 , 474 , 595 263 ,
<code>\l__siunitx_table_align_uncertainty-</code>	287 , 293 , 304 , 315 , 323 , 448 , 463 ,
bool 532 , 775	477 , 492 , 510 , 511 , 523 , 589 , 599 ,
<code>\l__siunitx_table_auto_round-</code>	657 , 748 , 752 , 801 , 803 , 808 , 819 , 821
bool 340 , 622	<code>__siunitx_table_direct_begin:</code> ..
<code>\l__siunitx_table_before_box</code> 12 , 430
..... 320 , 322 ,	<code>__siunitx_table_direct_begin:w</code> 430
528 , 550 , 551 , 553 , 559 , 561 , 565 ,	<code>__siunitx_table_direct_end:</code> 20 , 430
570 , 597 , 613 , 614 , 616 , 655 , 737 , 740	<code>__siunitx_table_direct_format:</code> 430
<code>\l__siunitx_table_before_dim</code> ...	<code>__siunitx_table_direct_format:nnnnnnn</code>
..... 530 , 619 , 680 , 729 , 737 430
<code>\l__siunitx_table_before_model-</code>	<code>__siunitx_table_direct_format:w</code> 430
tl 355 , 368 , 448 , 612	<code>__siunitx_table_direct_format-</code>
<code>\l__siunitx_table_before_tl</code>	aux:w 484 , 487
..... 107 , 116 , 120 , 123	<code>__siunitx_table_direct_format-</code>
<code>\l__siunitx_table_carry_dim</code>	end: 430
.... 531 , 646 , 649 , 749 , 804 , 812 , 824	<code>__siunitx_table_direct_format-</code>
<code>__siunitx_table_center_marker:</code> .	switch: 430
..... 279 , 473 , 594	<code>__siunitx_table_direct_marker:</code> 430
<code>__siunitx_table_center_marker-</code>	<code>__siunitx_table_direct_marker-</code>
aux:Nnn 293 , 300 , 313 , 320 , 331	end: 430
<code>__siunitx_table_cleanup-</code>	<code>__siunitx_table_direct_marker-</code>
decimal:w 267 , 491	switch: 430
	<code>__siunitx_table_direct_none:</code> .. 430

__siunitx_table_direct_none_-	__siunitx_table_print_marker_-
end: 430	aux:w 545
__siunitx_table_fill:	__siunitx_table_print_none:nnn 545
.... 265 , 308 , 328 , 450 , 494 , 518 ,	__siunitx_table_print_text:n ...
564 , 573 , 651 , 673 , 703 , 716 , 739 , 809 120 , 257 , 436
__siunitx_table_fill: 265 , 504	__siunitx_table_skip:n
\l__siunitx_table_fixed_width_- 193 , 229 , 231 , 243 , 245
bool 198 , 218 , 225	__siunitx_table_split:nNNN ...
\l__siunitx_table_format_tl 367 , 114 , 144 , 354
376 , 378 , 379 , 382 , 456 , 460 , 464 , 630	__siunitx_table_split_group:NNNn
__siunitx_table_generate_- 144
model:n 358 , 371	__siunitx_table_split_loop:NNN 144
__siunitx_table_generate_-	__siunitx_table_split_tidy:N ...
model:nnnnnnn 371 , 463 150 , 151 , 182
__siunitx_table_generate_model_-	__siunitx_table_split_tidy:Nn . 182
S:nnn 371	__siunitx_table_split_token:NNNN
__siunitx_table_generate_model_- 144
S:nnw 371	\l__siunitx_table_text_bool ...
\l__siunitx_table_integer_box 6 , 9 , 16 , 259
..... 263 , 284 , 295 , 303 , 313 ,	\l__siunitx_table_tmp_box 3 ,
324 , 451 , 476 , 496 , 522 , 587 , 598 ,	281 , 288 , 296 , 305 , 316 , 325 , 446 ,
656 , 670 , 679 , 683 , 694 , 696 , 698 ,	449 , 490 , 493 , 495 , 497 , 612 , 619 ,
702 , 710 , 712 , 717 , 723 , 724 , 726 , 733	644 , 646 , 667 , 671 , 682 , 691 , 699 ,
\l__siunitx_table_model_tl	713 , 731 , 747 , 751 , 772 , 773 , 774 ,
..... 356 , 358 , 367 , 387 , 483 , 638	783 , 784 , 785 , 805 , 810 , 815 , 822 , 827
\l__siunitx_table_number_tl	\l__siunitx_table_tmp_dim
..... 107 , 117 , 119 , 124	.. 136 , 3 , 722 , 732 , 773 , 784 , 814 , 826
__siunitx_table_print:nnn . 122 , 545	\l__siunitx_table_tmp_tl
__siunitx_table_print_format:nnn 3 , 352 , 354 , 482 ,
..... 545	485 , 576 , 577 , 578 , 579 , 580 , 582 ,
__siunitx_table_print_format:nnnnnn	620 , 633 , 635 , 636 , 640 , 643 , 836 , 837
..... 545	__siunitx_tmp:w
__siunitx_table_print_format_-	237 , 239 , 243 , 245 , 630 , 631 , 635 , 636
after:N 545	\l__siunitx_tmp_tl 43 , 113 ,
__siunitx_table_print_format_-	114 , 134 , 135 , 197 , 643 , 644 , 654 , 655
auxi:w 545	\l__siunitx_unit_autofrac_bool ..
__siunitx_table_print_format_-	523 , 530 , 537 , 544 , 551 , 558 , 577 , 955
auxii:w 545	\l__siunitx_unit_bracket_bool ...
__siunitx_table_print_format_- 571 , 609 , 702 , 772 , 879 , 900
auxiii:w 545	\l__siunitx_unit_bracket_close_-
__siunitx_table_print_format_-	tl 572 , 706 , 831
auxiv:w 545	\l__siunitx_unit_bracket_open_tl
__siunitx_table_print_format_- 572 , 704 , 828
auxv:w 545	\l__siunitx_unit_combine_exp_fp .
__siunitx_table_print_format_- 135 ,
auxvi:w 545	142 , 149 , 156 , 164 , 172 , 583 , 598 , 633
__siunitx_table_print_format_-	\l__siunitx_unit_current_tl 587 ,
auxvii:w 545	610 , 748 , 750 , 766 , 768 , 808 , 810 ,
__siunitx_table_print_format_-	813 , 821 , 859 , 866 , 874 , 926 , 934 , 937
box:Nn 545	\l__siunitx_unit_denominator_-
__siunitx_table_print_marker:nnn	bracket_bool 510 , 898
..... 545	\l__siunitx_unit_denominator_tl .
__siunitx_table_print_marker:w 545	589 , 594 , 899 , 944 , 985 , 994 , 1003 , 1010

\l_siunitx_unit_font_bool	_siunitx_unit_format_literal_-
..... 576, 611, 863, 869, 932, 939	super:nn 211
\l_siunitx_unit_forbid_literal_-	_siunitx_unit_format_literal_-
bool 192, 510	superscript: 211
_siunitx_unit_format:nnn 132	_siunitx_unit_format_literal_-
_siunitx_unit_format_aux: ... 132	tilde: 211
_siunitx_unit_format_bracket:N	_siunitx_unit_format_mass_to_-
..... 700, 750, 768, 994	kilogram: 605, 676
_siunitx_unit_format_combine_-	_siunitx_unit_format_multiply:
exp: 599, 628 601, 660
_siunitx_unit_format_finalise:	_siunitx_unit_format_output: ..
..... 618, 942 616, 876
_siunitx_unit_format_finalise_-	_siunitx_unit_format_output_-
autofrac: 942	aux: 876
_siunitx_unit_format_finalise_-	_siunitx_unit_format_output_-
fraction: 960, 966, 979	aux:nn 876
_siunitx_unit_format_finalise_-	_siunitx_unit_format_output_-
fractional: 942	denominator: 876
_siunitx_unit_format_finalise_-	_siunitx_unit_format_parsed: ..
power: 942 189, 591
_siunitx_unit_format_finalise_-	_siunitx_unit_format_parsed_-
symbol: 959, 969, 988	aux:n 591
_siunitx_unit_format_font: ...	_siunitx_unit_format_power: .. 710
..... 712, 825, 837, 847, 878, 930	_siunitx_unit_format_power_-
_siunitx_unit_format_literal:n	aux:wTF 710
..... 194, 197, 211	_siunitx_unit_format_power_-
_siunitx_unit_format_literal_-	negative: 710
add:n 211	_siunitx_unit_format_power_-
_siunitx_unit_format_literal_-	negative_aux:w 710
auxi:w 211	_siunitx_unit_format_power_-
_siunitx_unit_format_literal_-	positive: 710
auxii:n 251, 255	_siunitx_unit_format_power_-
_siunitx_unit_format_literal_-	superscript:w 741, 744
auxii:w 211	_siunitx_unit_format_power_-
_siunitx_unit_format_literal_-	superscript: 710
auxiii:w 211	_siunitx_unit_format_prefix: . 774
_siunitx_unit_format_literal_-	_siunitx_unit_format_prefix_-
auxiv:n 211	exp: 774
_siunitx_unit_format_literal_-	_siunitx_unit_format_prefix_-
auxix:nn 211	gram: 774
_siunitx_unit_format_literal_-	_siunitx_unit_format_prefix_-
auxv:nw 211	symbol: 774
_siunitx_unit_format_literal_-	_siunitx_unit_format_qualifier:
auxvi:nN 211 814
_siunitx_unit_format_literal_-	_siunitx_unit_format_qualifier_-
auxvii:nN 211	bracket: 814
_siunitx_unit_format_literal_-	_siunitx_unit_format_qualifier_-
auxviii:nN 211	combine: 814
_siunitx_unit_format_literal_-	_siunitx_unit_format_qualifier_-
auxx:nw 211	phrase: 814
_siunitx_unit_format_literal_-	_siunitx_unit_format_qualifier_-
sub:nn 211	subscript: 814
_siunitx_unit_format_literal_-	_siunitx_unit_format_special: 857
subscript: 211	_siunitx_unit_format_unit: .. 871

\l__siunitx_unit_formatted_tl ...	\l__siunitx_unit_part_tl
..... 167, 480, 482, 483, 484,
131, 181, 201, 232, 240, 241, 309,	491, 587, 625, 714, 727, 730, 732,
316, 329, 331, 595, 907, 908, 953,	742, 783, 798, 804, 805, 813, 821,
954, 968, 970, 974, 975, 976, 981,	826, 830, 838, 842, 848, 853, 861, 874
984, 990, 992, 999, 1002, 1006, 1008	\l__siunitx_unit_per_bool
__siunitx_unit_if_symbolic:n ... 11 162, 357, 364, 448, 459
__siunitx_unit_if_symbolic:nTF .	\l__siunitx_unit_per_symbol_bool
..... 11, 79, 185 524, 531,
__siunitx_unit_literal_power:nn	538, 545, 552, 559, 577, 906, 912, 958
..... 43, 117, 209	\l__siunitx_unit_per_symbol_tl ..
__siunitx_unit_literal_special:nN 510, 993
..... 111, 210	\l__siunitx_unit_position_int ...
\l__siunitx_unit_mass_kilogram_- 167,
bool	357, 366, 369, 389, 396, 407, 419,
122, 604, 785	423, 433, 438, 442, 446, 451, 465,
\c__siunitx_unit_math_subscript_-	505, 593, 597, 607, 613, 623, 788, 793
tl	\l__siunitx_unit_powers_positive_-
7, 221, 245, 292, 318, 850	bool
\l__siunitx_unit_multiple_fp 136,	525,
143, 150, 157, 165, 173, 586, 600, 667	532, 539, 546, 553, 560, 577, 725, 946
__siunitx_unit_non_latin:n 1014,	\l__siunitx_unit_prefix_exp_bool
1049, 1065, 1076, 1099, 1100, 1101 134,
__siunitx_unit_non_latin:nnnn 1014	141, 148, 155, 163, 171, 584, 603, 776
\l__siunitx_unit_numerator_bool .	\l__siunitx_unit_prefix_fp
..... 173, 581, 612, 724, 883 182, 203, 585, 596, 694, 695, 797
\l__siunitx_unit_options_bool ...	\l__siunitx_unit_prefixes_-
..... 88, 91, 102	forward_prop
__siunitx_unit_parse:n 187, 360	49, 636, 782
__siunitx_unit_parse_add:nnnn ..	\l__siunitx_unit_prefixes_-
..... 373,	reverse_prop
390, 404, 422, 432, 441, 445, 450, 464	49, 651
\l__siunitx_unit_parse_bool 183, 510	\l__siunitx_unit_product_tl
__siunitx_unit_parse_finalise: 122, 252, 891, 902, 1009
..... 371, 500	\l__siunitx_unit_qualifier_mode_-
__siunitx_unit_parse_finalise:n	tl
..... 370, 469	510, 582, 819
__siunitx_unit_parse_per: . 106, 455	\l__siunitx_unit_qualifier_-
__siunitx_unit_parse_power:nnN .	phrase_tl
..... 44, 47, 118, 121, 387	510, 840
__siunitx_unit_parse_prefix:Nn .	\l__siunitx_unit_separator_tl .. 211
..... 53, 387	__siunitx_unit_set_symbolic:Nnn
__siunitx_unit_parse_qualifier:nn 23, 42, 45, 51, 66, 76, 104
..... 68, 115, 387	__siunitx_unit_set_symbolic:Nnnn
__siunitx_unit_parse_special:n 23
..... 109, 112, 387	__siunitx_unit_set_symbolic:Npnn
__siunitx_unit_parse_unit:Nn 81, 436 23, 107, 110, 113, 116, 119
\l__siunitx_unit_parsed_prop ...	\l__siunitx_unit_sticky_per_bool
..... 160, 188, 162, 353, 457
357, 362, 376, 383, 401, 420, 472,	\l__siunitx_unit_test_bool
474, 478, 486, 490, 495, 506, 624, 10, 14, 32, 365
630, 634, 639, 649, 653, 662, 664,	\l__siunitx_unit_tmp_fp .. 4, 137,
669, 671, 680, 682, 689, 691, 789, 794	151, 158, 174, 632, 647, 666, 668, 672
\l__siunitx_unit_parsing_bool ...	\l__siunitx_unit_tmp_int . 4, 389, 391
..... 9, 34, 216, 349, 363, 685, 807	\l__siunitx_unit_tmp_tl
 4, 15, 17, 217, 218, 220,
	230, 231, 233, 236, 375, 377, 384,
	395, 401, 418, 420, 471, 472, 475,
	476, 479, 487, 491, 496, 504, 506,

622, 625, 630, 631, 633, 634, 637, 639, 641, 642, 645, 646, 647, 648, 652, 653, 656, 664, 665, 667, 686, 688, 690, 692, 693, 695, 755, 769, 787, 789, 792, 795, 796, 798, 968, 973	table-column-width 118
\l_siunitx_unit_total_int 590, 593, 607, 678	table-fixed-width 118
\l_siunitx_unit_two_part_bool . . 526, 533, 540, 547, 554, 561, 577, 895	table-format 118
skip commands:	table-number-alignment 118
\skip_horizontal:N 250	table-text-alignment 118
\skip_horizontal:n 195, 225, 649	\tablenum 209
\c_zero_skip 196	\tabularnewline 55, 57, 84, 86
\sp 170	\tebi 188, 2
\space 50, 52, 56, 58, 62, 64, 542, 544, 548, 550, 553, 559, 561, 565, 567, 570, 576, 578, 585, 587	\tera 145, 13, 57, 1053
space-before-unit 189	\tesla 146, 1075
\SplitArgument 100	\TeV 183, 42
\SplitList 139, 148, 157, 167, 662	T _E X and L ^A T _E X 2 _ε commands:
\square 146, 1103	\@ifl@t@r 12, 42
\squared 147, 1103	\@ifpackagelater 1
\steradian 146, 1075	\@ifpackageloaded 30, 46, 48, 69, 75, 80, 89, 107, 114, 117, 153, 229, 236, 260, 326, 334, 708, 822
sticky-per 149	\@ifundefined 9
str commands:	\@maybe@unskip 81
\str_case_e:nnTF 153	\@temptokena 248, 250
\str_if_eq:nnTF 41, 69, 82, 130, 134, 189, 237, 265, 299, 299, 334, 424, 610, 727, 732, 759, 776, 821, 831, 839, 846, 852, 861, 963, 1141, 1164, 1233, 1280, 1430, 1443, 1456, 1605, 1630, 1774, 1823, 1866, 1912, 2085	\array@row@rst 122, 134, 135
\str_if_eq_p:nn . . . 103, 320, 445, 545, 683, 685, 707, 709, 1323, 1325, 1421, 1423, 1525, 1526, 1735, 1741, 1756, 1809, 1812, 1992, 2005, 2030	\f@family 153
\str_range:nnn 422, 424	\f@series 106
\symoperators 93, 179	\FB@fg 347
sys commands:	\m@th 381, 438
\sys_if_engine luatex_p: 11, 127, 143, 172, 682, 739, 755, 1015	\math@version 103, 134
\sys_if_engine xetex:TF 322	\NC@do 237, 243, 244
\sys_if_engine xetex_p: 12, 128, 144, 173, 683, 740, 756, 1016	\NC@find 253
	\NC@list 7, 238, 239, 244, 245
	\newcommand 191
	\protected@edef 122, 156, 15, 35, 78, 125, 136, 230, 352
	\sf@size 394
	\tab@setcr 82
	\use@mathgroup 189, 191
	\z@ 394
	tex commands:
	\tex_cr:D 32
	\tex_hfil:D 247, 265
	\tex_hfill:D 266
	\tex_hss:D 267, 269
	\tex_kern:D 196
	\text 93, 103, 51, 57, 63, 67, 87, 116, 124, 138, 140, 243, 543, 549, 560, 566, 577, 586, 796, 801, 806, 809, 812, 819, 828
	text-family-to-math 95
	text-font-command 95
	text-series-to-math 95
	\textcenteredperiod 93
	\textcolor . . . 93–95, 141, 147, 70, 111, 112
	\textdegree 70, 90
	\textminus 93, 286
	\textmu 146, 742
T	
table-align-comparator 117	
table-align-exponent 117	
table-align-text-after 117	
table-align-text-before 117	
table-align-uncertainty 117	
table-alignment 118	
table-alignment-mode 118	
table-auto-round 118	

<code>\textohm</code>	130, 758	<code>\tl_if_head_eq_meaning:nTF</code>	259, 267, 274, 318
<code>\textperiodcentered</code>	290	<code>\tl_if_head_is_group:nTF</code>	278
<code>\textpm</code>	93, 282	<code>\tl_if_head_is_N_type:nTF</code>	275
<code>\textsubscript</code>	93, 343, 374	<code>\tl_if_in:NnTF</code> ...	5, 57, 157, 251,
<code>\textsuperscript</code>	93, 348		310, 341, 385, 391, 403, 406, 413,
<code>\texttimes</code>	93, 288		418, 487, 513, 531, 542, 556, 562, 569
<code>\the</code>	239, 245, 250	<code>\tl_if_novalue:nTF</code>	223, 226
<code>\THz</code>	182, 8	<code>\tl_if_single_token:nTF</code>	93
<code>tight-spacing</code>	42	<code>\tl_map_function:nN</code>	103, 129
<code>\times</code> ...	38, 14, 20, 26, 32, 287, 580, 2111	<code>\tl_map_inline:Nn</code>	120, 133, 271, 274, 453, 499
tl commands:		<code>\tl_map_inline:nn</code>	55
<code>\c_empty_tl</code>	55, 56, 1876	<code>\tl_new:N</code>	3, 3, 3, 4,
<code>\c_space_tl</code>	188, 287, 341, 343, 366, 371, 410, 418		4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 8, 9,
<code>\tl_clear:N</code>	27, 101, 113, 119, 146, 147,		9, 9, 10, 10, 13, 14, 15, 23, 29, 38,
	148, 149, 160, 162, 181, 232, 256,		39, 43, 69, 70, 71, 72, 73, 74, 75, 76,
	266, 302, 314, 324, 336, 350, 371,		87, 88, 89, 90, 107, 108, 109, 131,
	479, 503, 504, 524, 594, 595, 610, 695		256, 346, 365, 366, 367, 368, 369,
<code>\tl_clear_new:N</code>	83		370, 403, 404, 447, 449, 501, 502,
<code>\tl_const:Nn</code>	7, 92, 96, 98		572, 573, 582, 587, 588, 589, 670,
<code>\tl_count:N</code>	600, 612		671, 672, 1632, 1633, 1637, 1700, 2037
<code>\tl_count:n</code>	113, 184, 185, 412, 600, 608,	<code>\tl_put_right:Nn</code>	47,
	687, 710, 751, 988, 989, 1244, 1326,		58, 64, 162, 163, 172, 175, 176, 179,
	1392, 1395, 1402, 1412, 1424, 1473,		236, 309, 316, 329, 343, 382, 393,
	1492, 1522, 1534, 1561, 1584, 1592,		435, 447, 473, 489, 547, 580, 821, 873
	1607, 1837, 1848, 1918, 1976, 1984	<code>\tl_remove_all:Nn</code>	6, 134
<code>\tl_head:n</code>	99, 215,	<code>\tl_replace_all:Nnn</code>	3, 5, 5, 7, 89, 90,
	270, 395, 1106, 1124, 1296, 1456, 1904		105, 202, 205, 218, 220, 234, 273, 302
<code>\tl_head:w</code>	921, 945	<code>\tl_reverse:n</code> ..	806, 1198, 1199, 1205
<code>\tl_if_blank:nTF</code> 3, 17, 44, 103, 121,		<code>\tl_set:Nn</code>	62,
	132, 142, 146, 158, 184, 192, 211,		156, 175, 7, 8, 16, 17, 22, 25, 27, 51,
	301, 323, 362, 364, 366, 369, 371,		53, 54, 66, 77, 82, 85, 99, 105, 111,
	377, 393, 399, 417, 483, 607, 674,		118, 120, 124, 127, 132, 133, 139,
	746, 759, 781, 793, 799, 811, 817,		140, 146, 148, 155, 161, 163, 168,
	839, 855, 915, 929, 939, 952, 982,		170, 190, 200, 200, 213, 217, 222,
	1054, 1171, 1333, 1374, 1485, 1533,		231, 233, 240, 253, 254, 254, 264,
	1762, 1768, 1786, 1821, 1901, 2026		264, 269, 270, 298, 300, 321, 322,
<code>\tl_if_blank_p:n</code> .	4, 147, 149, 151,		331, 332, 333, 333, 355, 359, 367,
	219, 220, 392, 393, 1452, 1737, 1975		375, 378, 387, 395, 418, 434, 441,
<code>\tl_if_empty:NnTF</code>	68,		448, 456, 465, 471, 472, 476, 482,
	88, 105, 107, 109, 119, 125, 135,		482, 504, 515, 519, 520, 522, 574,
	143, 161, 164, 169, 174, 184, 233,		575, 578, 589, 622, 628, 631, 635,
	257, 262, 265, 305, 313, 331, 353,		636, 642, 645, 646, 665, 680, 686,
	379, 471, 523, 578, 694, 924, 944,		688, 693, 720, 727, 729, 748, 755,
	953, 999, 1427, 1616, 1711, 1994, 2045		766, 787, 792, 796, 804, 808, 810,
<code>\tl_if_empty:nTF</code> ...	84, 333, 719, 1716		826, 838, 848, 859, 908, 921, 934,
<code>\tl_if_empty_p:N</code>	272, 287, 433, 899, 907, 1757		954, 968, 968, 970, 981, 990, 1005,
<code>\tl_if_eq:nnTF</code>	439		1006, 1015, 1025, 1598, 1634, 1635
<code>\tl_if_exist:NnTF</code>	95	<code>\tl_set_eq:NN</code>	18, 20, 44, 131, 159, 252, 346,
<code>\tl_if_head_eq_charcode:nNTF</code> ..	758		363, 409, 415, 454, 464, 507, 511,
<code>\tl_if_head_eq_charcode_p:nN</code> ..	150		567, 647, 663, 813, 1001, 1018, 1694

<code>\tl_tail:n</code>	100, 922, 946	use commands:	
<code>\tl_use:N</code>	75, 124, 152, 208, 224, 258, 272, 311, 313, 324, 333	<code>\use:N</code>	65, 72, 184, 188, 260, 395, 439, 443, 474, 520, 546, 595, 626, 653, 816, 833, 867, 880, 891, 954, 994, 1235, 1542, 1550, 1557, 1839, 1845, 1904, 1924
<code>\l_tmpa_tl</code>	141, 142	<code>\use:n</code>	69, 70, 92, 139, 157, 180, 186, 189, 190, 220, 227, 258, 276, 285, 340, 395, 835, 848, 854, 864, 1825, 1868, 2009
token commands:		<code>\use_i:nn</code>	75, 867, 954
<code>\c_math_toggle_token</code>	452, 461, 464, 469, 498, 508, 512, 517, 526, 527	<code>\use_i:nnnn</code>	156
<code>\token_if_eq_charcode_p:NN</code>	517	<code>\use_i_delimit_by_q_recursion_-</code> stop:nw	194, 1262, 1309, 2087, 2089
<code>\token_if_eq_meaning:NNTF</code>	103, 288, 291, 1628	<code>\use_i_delimit_by_q_stop:nw</code> . . .	104
<code>\token_if_macro:NNTF</code>	122	<code>\use_ii:nn</code>	1460
<code>\token_to_str:N</code>	30, 83, 85, 85, 95, 98, 106, 122, 180, 219, 221, 235, 351, 352, 380, 413, 427, 768, 770, 775, 779, 782, 784, 1118, 1121	<code>\use_iv:nnnn</code>	148, 152
<code>\tonne</code>	146, 1086	<code>\use_none:n</code>	159, 483, 759, 1298, 1886
<code>\tothe</code>	147, 116	<code>\use_none:nn</code>	1890
<code>\TrimSpaces</code>	91, 128, 139, 157, 167, 175, 662, 671	<code>\use_none:nnnn</code>	88, 125
<code>\ttdefault</code>	157	use-xspace	189
U		<code>\uV</code>	183, 21
<code>\uA</code>	182, 2	<code>\uW</code>	183, 42
<code>\uF</code>	184, 70	V	
<code>\ug</code>	182, 35	<code>\V</code>	183, 21
<code>\uH</code>	75	<code>\volt</code>	146, 21, 22, 23, 24, 25, 26, 1075
<code>\uJ</code>	183, 42	W	
<code>\uL</code>	183, 27	<code>\W</code>	183, 42
<code>\ul</code>	183, 191, 27, 52	<code>\watt</code>	146, 42, 43, 44, 45, 46, 47, 58, 1075
<code>\um</code>	182, 60	<code>\weber</code>	146, 1075
<code>\umol</code>	183, 14	X	
uncertainty-mode	43	<code>\xspace</code>	87
uncertainty-separator	43	Y	
<code>\unit</code>	206, 100, 293, 327	<code>\yobi</code>	188, 2
unit-color	95	<code>\yocto</code>	145, 1043
unit-font-command	149	<code>\yotta</code>	145, 1053
unit-mode	95	Z	
unit-optional-argument	189	<code>\zebi</code>	188, 2
<code>\unskip</code>	54, 83	<code>\zepto</code>	145, 1043
<code>\upOmega</code>	101, 102, 748, 749	<code>\zetta</code>	145, 1053
<code>\upshape</code>	94		
<code>\us</code>	182, 89		