

siunitx – A comprehensive (SI) units package*

Joseph Wright[†]

Released 2021-06-01

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 v3.0.9, last revised 2021-06-01.

[†]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	78
2.6	Outputting parsed numbers	79
2.7	Miscellaneous tools	87
2.8	Messages	89
2.9	Standard settings for module options	89
VI	siunitx-print – Printing material with font control	91
1	Printing quantities	91
1.1	Key-value options	92
2	siunitx-print implementation	94
2.1	Initial set up	94
2.2	Printing routines	94
2.3	Standard settings for module options	103
VII	siunitx-quantity – Quantities	104
1	siunitx-quantity implementation	104
1.1	Initial set-up	105
1.2	Main formatting routine	105
1.3	Standard settings for module options	108
1.4	Adjustments to units	108
VIII	siunitx-symbol – Symbol-related settings	110
1	siunitx-symbol implementation	110
1.1	Bookmark definitions	113
IX	siunitx-table – Formatting numbers in tables	114

1	Numbers in tables	114
1.1	Key-value options	114
2	siunitx-table implementation	116
2.1	Interface functions	116
2.2	Collecting tokens	116
2.3	Separating collected material	119
2.4	Printing numbers in cells: spacing	121
2.5	Printing just text	122
2.6	Number alignment: core ideas	123
2.7	Directly printing without collection	126
2.8	Printing numbers in cells: main functions	128
2.9	Standard settings for module options	135
X	siunitx-unit – Parsing and formatting units	136
1	Formatting units	136
2	Defining symbolic units	138
3	Per-unit options	139
4	Units in (PDF) strings	139
5	Pre-defined symbolic unit components	139
5.1	Key-value options	142
6	siunitx-unit implementation	144
6.1	Initial set up	144
6.2	Defining symbolic unit	145
6.3	Applying unit options	147
6.4	Non-standard symbolic units	148
6.5	Main formatting routine	149
6.6	Formatting literal units	151
6.7	(PDF) String creation	154
6.8	Parsing symbolic units	154
6.9	Formatting parsed units	159
6.10	Non-Latin character support	171
6.11	Pre-defined unit components	172
6.12	Messages	174
6.13	Standard settings for module options	175
XI	siunitx-abbreviations – Abbreviations	176
1	siunitx-abbreviation implementation	178
XII	siunitx-binary – Binary units	182

1	siunitx-binary implementation	182
XIII	siunitx-command – Units as document command	183
1	Creating units as document commands	183
1.1	Key-value options	183
2	siunitx-command implementation	184
2.1	Options	184
2.2	Creation of unit document commands	185
2.3	Standard settings for module options	186
XIV	siunitx-emulation – Emulation	187
1	siunitx-emulation implementation	187
1.1	Load-time option	188
1.2	Angle options	188
1.3	Combination functions options	188
1.4	Command options	189
1.5	Print options	189
1.6	Symbol options	191
1.7	Number options	191
1.7.1	Table options	195
1.8	Unit options	197
1.9	Quantity units	198
1.10	Preamble commands	199
1.11	Document commands	200
1.12	Symbol commands	201
	Index	204

Part I

siunitx — Overall set up

1 siunitx implementation

Start the DocStrip guards.

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

1.1 Initial set up

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

```
6 \DeclareRelease{2}{2010-05-23}{siunitx-v2.sty}
7 \DeclareRelease{v2}{2010-05-23}{siunitx-v2.sty}
8 \DeclareCurrentRelease{}{2021-05-17}
```

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

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

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

Identify the package and give the over all version information.

```
23 \ProvidesExplPackage {siunitx} {2021-06-01} {3.0.9}
24   {A comprehensive (SI) units package}
```

1.2 Safety checks

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

```

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

```

(End definition for __siunitx_load_check:.)

1.3 Provide a kernel command

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

```

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

```

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

1.4 Top-level scratch space

`\l__siunitx_tmp_tl` Scratch space for the interfaces.

```

43 \tl_new:N \l__siunitx_tmp_tl

```

(End definition for \l__siunitx_tmp_tl.)

```

44 \</init>

```

```

45 \<*options>

```

1.5 Load time options

`\l__siunitx_column_type_tl`

```

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

```

(End definition for \l__siunitx_column_type_tl.)

1.6 Option handling

```

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

```

1.7 User interfaces

```

58 \<*interfaces>

```

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

User level interfaces are all created by xparse

```

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

```

1.7.1 Preamble commands

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

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

1.7.2 Document commands

```

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

```

```

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

```

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

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

```

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

```

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

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


```

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

```

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

\complexnum Interfaces for complex numbers.

```

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

```

```

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

```

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

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

```

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

```

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

`\sisetup` A very thin wrapper.

```

208 \NewDocumentCommand \sisetup { m }
209 { \keys_set:nn { siunitx } {#1} }

```

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

1.8 “Glue” commands

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

```

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

```

(End definition for `__siunitx_angle:nnn`.)

1.9 Table column

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

```
220 \RequirePackage { array }
```

```
\_siunitx_declare_column:Nnn
```

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

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

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

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

```

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

```

(End definition for __siunitx_declare_column:Nnn.)

1.10 Document commands in bookmarks

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

__siunitx_bookmark_cmd:Nn

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

```

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

```

(End definition for __siunitx_bookmark_cmd:Nn.)

We also need the v2 names.

```

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

```

```

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

```

\c__siunitx_bookmark_seq Commands usable in bookmarks

```

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

```

(End definition for \c__siunitx_bookmark_seq.)

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

```

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

```

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

```

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

```

```

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

```

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

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

```

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

```

```

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

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

406 \end{interfaces}
407 \end{package}

```

Part II

siunitx-angle – Formatting angles

1 Formatting angles

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

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

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

1.1 Key-value options

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

`angle-mode`

`angle-mode = ⟨choice⟩`

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

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

`angle-symbol-degree = ⟨symbol⟩`

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

`angle-symbol-over-decimal`

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

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

`arc-separator`

`arc-separator = ⟨separator⟩`

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

`fill-angle-degrees`

`fill-arc-degrees = true|false`

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

`fill-angle-minutes`

`fill-arc-minutes = true|false`

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

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

2 siunitx-angle implementation

Start the DocStrip guards.

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

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

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

```
\l__siunitx_angle_tmp_bool
\l__siunitx_angle_tmp_dim
\l__siunitx_angle_tmp_tl
```

Scratch space.

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

(End definition for `\l__siunitx_angle_tmp_bool`, `\l__siunitx_angle_tmp_dim`, and `\l__siunitx_angle_tmp_tl`.)

```
\l__siunitx_angle_symbol_degree_tl
\l__siunitx_angle_symbol_minute_tl
\l__siunitx_angle_symbol_second_tl
\l__siunitx_angle_force_arc_bool
\l__siunitx_angle_force_decimal_bool
\l__siunitx_angle_astronomy_bool
\l__siunitx_angle_separator_tl
\l__siunitx_angle_fill_degrees_bool
\l__siunitx_angle_fill_minutes_bool
\l__siunitx_angle_fill_seconds_bool
\l__siunitx_angle_product_tl
```

```
6 \keys_define:nn { siunitx }
7 {
8   angle-mode .choice: ,
9   angle-mode / arc .code:n =
10   {
11     \bool_set_true:N \l__siunitx_angle_force_arc_bool
12     \bool_set_false:N \l__siunitx_angle_force_decimal_bool
13   } ,
14   angle-mode / decimal .code:n =
15   {
16     \bool_set_false:N \l__siunitx_angle_force_arc_bool
17     \bool_set_true:N \l__siunitx_angle_force_decimal_bool
18   } ,
19   angle-mode / input .code:n =
20   {
21     \bool_set_false:N \l__siunitx_angle_force_arc_bool
22     \bool_set_false:N \l__siunitx_angle_force_decimal_bool
23   } ,
24   angle-symbol-degree .tl_set:N =
25     \l__siunitx_angle_symbol_degree_tl ,
26   angle-symbol-minute .tl_set:N =
27     \l__siunitx_angle_symbol_minute_tl ,
28   angle-symbol-second .tl_set:N =
29     \l__siunitx_angle_symbol_second_tl ,
30   angle-symbol-over-decimal .bool_set:N =
31     \l__siunitx_angle_astronomy_bool ,
```

```

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

```

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

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

```

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

```

```

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

```

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

```

\l__siunitx_angle_degrees_tl Space for formatting parsed numbers.
\l__siunitx_angle_minutes_tl 87 \tl_new:N \l__siunitx_angle_degrees_tl
\l__siunitx_angle_seconds_tl 88 \tl_new:N \l__siunitx_angle_minutes_tl
                             89 \tl_new:N \l__siunitx_angle_seconds_tl

```

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

```

\l__siunitx_angle_sign_tl For the “sign shuffle”.
                             90 \tl_new:N \l__siunitx_angle_sign_tl
                             (End definition for \l__siunitx_angle_sign_tl.)

```

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

```

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

```

```

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

```

(End definition for __siunitx_angle_arc_sign:nnn and others.)

\l__siunitx_angle_marker_box For “astronomy style” angles.
\l__siunitx_angle_unit_box

```

170 \box_new:N \l__siunitx_angle_marker_box
171 \box_new:N \l__siunitx_angle_unit_box

```

(End definition for `\l__siunitx_angle_marker_box` and `\l__siunitx_angle_unit_box`.)

```

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

```

The final stage of printing an angle is to put together the three parts: this works even for decimal angles as they will blank arguments for the other two parts. The need to handle astronomy-style formatting means that the number has to be decomposed into parts.

```

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

```

```

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

```

```

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

```

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

```

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

```

Part III

siunitx-compound – Compound numbers and quantities

<hr/> <hr/> <code>\siunitx_compound_number:n</code>	<code>\siunitx_compound_number:n {<entries>}</code> Prints a set of numbers in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$. Unlike <code>\siunitx_number_list:nn</code> , this function may semantically take any form
<hr/> <hr/> <code>\siunitx_compound_quantity:nn</code>	<code>\siunitx_compound_quantity:nn {<entries>} {<unit>}</code> Prints a set of quantities in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$. Unlike <code>\siunitx_quantity_list:nn</code> , this function may semantically take any form
<hr/> <hr/> <code>\siunitx_number_list:nn</code>	<code>\siunitx_number_list:nn {<entries>}</code> Prints the list of numbers in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$.
<hr/> <hr/> <code>\siunitx_quantity_list:nn</code>	<code>\siunitx_quantity_list:nn {<entries>} {<unit>}</code> Prints the list of quantities in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$.
<hr/> <hr/> <code>\siunitx_number_product:n</code>	<code>\siunitx_number_product:n {<entries>}</code> Prints the series of numbers in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$.
<hr/> <hr/> <code>\siunitx_quantity_product:nn</code>	<code>\siunitx_number_product:n {<entries>} {<unit>}</code> Prints the series of quantities in the $\langle entries \rangle$, each of which should be given as a $\langle balanced text \rangle$.
<hr/> <hr/> <code>\siunitx_number_range:nn</code>	<code>\siunitx_number_range:nn {<start>} {<end>}</code> Prints the range of numbers from the $\langle start \rangle$ to the $\langle end \rangle$.
<hr/> <hr/> <code>\siunitx_quantity_range:nnn</code>	<code>\siunitx_number_range:nn {<start>} {<end>} {<unit>}</code> Prints the range of quantities from the $\langle start \rangle$ to the $\langle end \rangle$.
<hr/> <hr/> <code>\l_siunitx_list_separator_pair_tl</code> <code>\l_siunitx_list_separator_tl</code> <code>\l_siunitx_list_separator_final_tl</code>	Separators for lists of numbers and quantities.

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

<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`

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

```

\__siunitx_compound_format_combine-exponent:nn
\__siunitx_compound_format_extract-exponent:nn
\__siunitx_compound_format_input:n
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   \str_if_eq:VnT \l__siunitx_quantity_prefix_mode_tl { combine-exponent }
264     { \tl_clear:N \l__siunitx_compound_exp_tl }
265   \bool_if:NTF \l__siunitx_compound_unit_repeat_bool
266     { \__siunitx_compound_print:N \__siunitx_compound_print_quantity:x }
267     {
268       \bool_lazy_and:nnTF
269         { \l__siunitx_compound_unit_bracket_bool }
270         { \tl_if_empty_p:N \l__siunitx_compound_exp_tl }
271         {
272           \siunitx_print_number:V \l__siunitx_compound_bracket_open_tl
273           \__siunitx_compound_print:N \siunitx_print_number:x
274           \siunitx_print_number:V \l__siunitx_compound_bracket_close_tl
275         }
276         { \__siunitx_compound_print:N \siunitx_print_number:x }
277       \__siunitx_compound_print_quantity:n { }
278     }
279   \group_end:
280 }

```

(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.

```

281 \cs_new_protected:Npn \__siunitx_compound_print:N #1
282 {
283   \bool_lazy_and:nnTF
284     { \l__siunitx_compound_exp_bracket_bool }
285     { ! \tl_if_empty_p:N \l__siunitx_compound_exp_tl }
286     {
287       \__siunitx_compound_print:xxN
288       { \exp_not:V \l__siunitx_compound_bracket_open_tl }
289       {
290         \exp_not:V \l__siunitx_compound_bracket_close_tl
291         \exp_not:V \l__siunitx_compound_exp_tl
292       }
293       #1
294     }
295   { \__siunitx_compound_print:xxN { } { \exp_not:V \l__siunitx_compound_exp_tl } #1 }

```



```

296 }
297 \cs_new_protected:Npn \__siunitx_compound_print:nnN #1#2#3
298 {
299   \exp_args:Nx \__siunitx_compound_print:nnnN
300   { \seq_count:N \l__siunitx_compound_tmp_seq } {#1} {#2} #3
301 }
302 \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.

```

303 \cs_new_protected:Npn \__siunitx_compound_print:nnnN #1#2#3#4
304 {
305   \int_case:nnF {#1}
306   {
307     { 0 } { }
308     { 1 }
309     {
310       #4
311       { \seq_item:Nn \l__siunitx_compound_tmp_seq { 1 } }
312     }
313     { 2 }
314     {
315       #4
316       {
317         \exp_not:n {#2}
318         \seq_item:Nn \l__siunitx_compound_tmp_seq { 1 }
319       }
320       \__siunitx_compound_print_separator:V \l__siunitx_compound_separator_pair_tl
321       #4
322       {
323         \seq_item:Nn \l__siunitx_compound_tmp_seq { 2 }
324         \exp_not:n {#3}
325       }
326     }
327   }
328   {
329     \int_set:Nn \l__siunitx_compound_count_int {#1}
330     \tl_set:Nn \l__siunitx_compound_start_tl {#2}
331     \tl_set:Nn \l__siunitx_compound_end_tl {#3}
332     \cs_set_eq:NN \__siunitx_compound_print_aux:n #4
333     \seq_map_indexed_function:NN
334       \l__siunitx_compound_tmp_seq
335       \__siunitx_compound_print_aux:nn
336   }
337 }
338 \cs_new_protected:Npn \__siunitx_compound_print_aux:n #1 { }
339 \cs_new_protected:Npn \__siunitx_compound_print_aux:nn #1#2
340 {
341   \int_case:nnF {#1}
342   {
343     { 1 }
344     {

```

```

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

```

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

```

```

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

```

(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.

```

400 <@@=siunitx_list>

```

`\l_siunitx_list_separator_tl`

Options for products.

```

\l_siunitx_list_separator_final_tl 401 \tl_new:N \l__siunitx_list_exp_tl
\l_siunitx_list_separator_pair_tl 402 \tl_new:N \l__siunitx_list_units_tl
\l__siunitx_list_exp_tl 403 \keys_define:nn { siunitx }
\l__siunitx_list_units_tl 404 {
405     list-exponents .choices:nn =
406     { combine , combine-bracket , individual }
407     { \tl_set_eq:NN \l__siunitx_list_exp_tl \l_keys_choice_tl } ,
408     list-final-separator .tl_set:N = \l_siunitx_list_separator_final_tl ,
409     list-pair-separator .tl_set:N = \l_siunitx_list_separator_pair_tl ,
410     list-separator .tl_set:N = \l_siunitx_list_separator_tl ,
411     list-units .choices:nn =
412     { bracket , repeat , single }
413     { \tl_set_eq:NN \l__siunitx_list_units_tl \l_keys_choice_tl }
414 }

```

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

`\siunitx_number_list:nn`

Simply recover the settings and use as a list.

`\siunitx_quantity_list:nn`

```

\__siunitx_list_aux: 415 \cs_new_protected:Npn \siunitx_number_list:nn #1
416 {
417     \group_begin:
418     \__siunitx_list_aux:
419     \siunitx_compound_number:n {#1}
420     \group_end:
421 }
422 \cs_new_protected:Npn \siunitx_quantity_list:nn #1#2
423 {
424     \group_begin:
425     \__siunitx_list_aux:
426     \siunitx_compound_quantity:nn {#1} {#2}
427     \group_end:
428 }
429 \cs_new_protected:Npn \__siunitx_list_aux:
430 {
431     \keys_set:nx { siunitx }
432     {

```

```

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

```

(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.

```

444 <@@=siunitx_product>

```

```

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

```

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

```

\siunitx_number_product:n Simply recover the settings and use as a list.
\siunitx_quantity_product:nn
464 \cs_new_protected:Npn \siunitx_number_product:n #1
465 {
466   \group_begin:
467   \_siunitx_product_aux:
468   \siunitx_compound_number:n {#1}
469   \group_end:
470 }
471 \cs_new_protected:Npn \siunitx_quantity_product:nn #1#2
472 {

```

```

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

```

498 <@@=siunitx_range>

```

```

\l__siunitx_range_exp_tl
\l__siunitx_range_phrase_tl
\l__siunitx_range_units_tl

```

Options for products.

```

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

```

(End definition for `\l__siunitx_range_exp_tl`, `\l__siunitx_range_phrase_tl`, and `\l__siunitx_range_units_tl`. This variable is documented on page 21.)

```

\siunitx_number_range:nn
\siunitx_quantity_range:nnn
\__siunitx_range_aux:

```

Simply recover the settings and use as a list.

```

511 \cs_new_protected:Npn \siunitx_number_range:nn #1#2

```

```

512 {
513   \group_begin:
514     \_siunitx_range_aux:
515     \siunitx_compound_number:n { {#1} {#2} }
516   \group_end:
517 }
518 \cs_new_protected:Npn \siunitx_quantity_range:nnn #1#2#3
519 {
520   \group_begin:
521     \_siunitx_range_aux:
522     \siunitx_compound_quantity:nn { {#1} {#2} } {#3}
523   \group_end:
524 }
525 \cs_new_protected:Npn \_siunitx_range_aux:
526 {
527   \keys_set:nx { siunitx }
528   {
529     compound-exponents      = \l__siunitx_range_exp_tl ,
530     compound-pair-separator = { \exp_not:V \l_siunitx_range_phrase_tl } ,
531     compound-separator-mode = text ,
532     compound-units          = \l__siunitx_range_units_tl
533   }
534 }

```

(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.

```

535 \keys_set:nn { siunitx }
536 {
537   compound-exponents      = individual ,
538   compound-final-separator =
539   {
540     \ifmmode \ \else \space \fi
541     \text { and }
542     \ifmmode \ \else \space \fi
543   } ,
544   compound-pair-separator =
545   {
546     \ifmmode \ \else \space \fi
547     \text { and }
548     \ifmmode \ \else \space \fi
549   } ,
550   compound-separator      =
551   { , \ifmmode \ \else \space \fi } ,
552   compound-separator-mode = text ,
553   compound-units          = repeat ,
554   list-exponents          = individual ,
555   list-final-separator    =
556   {
557     \ifmmode \ \else \space \fi

```

```

558         \text { and }
559         \ifmmode \ \else \space \fi
560     } ,
561     list-pair-separator      =
562     {
563         \ifmmode \ \else \space \fi
564         \text { and }
565         \ifmmode \ \else \space \fi
566     } ,
567     list-separator          =
568     { , \ifmmode \ \else \space \fi } ,
569     list-units               = repeat ,
570     product-exponents       = individual ,
571     product-mode            = symbol ,
572     product-phrase          =
573     {
574         \ifmmode \ \else \space \fi
575         \text { by }
576         \ifmmode \ \else \space \fi
577     } ,
578     product-symbol          = \times ,
579     product-units           = repeat ,
580     range-exponents        = individual ,
581     range-phrase           =
582     {
583         \ifmmode \ \else \space \fi
584         \text { to }
585         \ifmmode \ \else \space \fi
586     } ,
587     range-units             = repeat
588 }
589 \end{package}

```

Part IV

siunitx-locale — Localisation

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

locale `locale = <locale>`

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

1 siunitx-locale implementation

Start the DocStrip guards.

```
1 <*package>
```

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

```
2 <@@=siunitx_locale>
```

1.1 Locales

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

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



```

30 locale / ZA .meta:n =
31 {
32     exponent-product      = \times ,
33     inter-unit-product    = \cdot ,
34     output-decimal-marker = { , }
35 }
36 }

```

1.2 Localisation

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

```

37 \file_if_exist:nT { translations.sty }
38 {
39     \RequirePackage { translations }
40     \DeclareTranslation { English } { to~(numerical~range) } { to }
41     \DeclareTranslation { French } { to~(numerical~range) } { à }
42     \DeclareTranslation { German } { to~(numerical~range) } { bis }
43     \DeclareTranslation { Spanish } { to~(numerical~range) } { a }
44     \keys_set:nn { siunitx }
45     {
46         list-final-separator =
47         {
48             \ifmmode \ \else \space \fi
49             \text { \GetTranslation { and } }
50             \ifmmode \ \else \space \fi
51         } ,
52         list-pair-separator =
53         {
54             \ifmmode \ \else \space \fi
55             \text { \GetTranslation { and } }
56             \ifmmode \ \else \space \fi
57         } ,
58         range-phrase =
59         {
60             \ifmmode \ \else \space \fi
61             \text { \GetTranslation { to~(numerical~range) } }
62             \ifmmode \ \else \space \fi
63         }
64     }
65 }

```

Part V

siunitx-number – Parsing and formatting numbers

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)

`\siunitx_number_process:NN`

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

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

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

with the result stored in $\langle tl\ var2 \rangle$.

`\siunitx_number_output:N` ☆ `\siunitx_number_output:N` $\langle number \rangle$
`\siunitx_number_output:n` ☆ `\siunitx_number_output:NN` $\langle number \rangle$ $\langle marker \rangle$
`\siunitx_number_output:NN` ☆
`\siunitx_number_output:nN` ☆

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.

`\siunitx_number_format:nN`

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

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`.

`\siunitx_number_adjust_exponent:Nn` ☆ `\siunitx_number_adjust_exponent:Nn` $\langle number \rangle$ $\{ \langle fp\ expr \rangle \}$
`\siunitx_number_adjust_exponent:nn` ☆

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.

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

Replaces all multi-token signs and comparators in the $\langle \text{tl } \text{var} \rangle$ 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_replace_all:Nnn { NnV }
```

`\l__siunitx_number_tmp_tl` Scratch space.

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

(End definition for `\l__siunitx_number_tmp_tl`.)

2.2 Main formatting routine

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

```
8 \tl_new:N \l_siunitx_number_outputted_tl
```

(End definition for `\l_siunitx_number_outputted_tl`.)

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

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

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

`\l_siunitx_number_parse_bool` Top-level options.

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

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

`\siunitx_number_format:nN`

```

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

```

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

```

28 \tl_new:N \l_siunitx_number_input_decimal_tl

```

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

```

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

```

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

```
\l_siunitx_number_arg_tl
```

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

```
69 \tl_new:N \l_siunitx_number_arg_tl
```

(End definition for \l_siunitx_number_arg_tl.)

```
\l_siunitx_number_comparator_tl
```

A comparator, if found, is held here.

```
70 \tl_new:N \l_siunitx_number_comparator_tl
```

(End definition for \l_siunitx_number_comparator_tl.)

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

71 `\tl_new:N \l_siunitx_number_exponent_tl`

(End definition for `\l_siunitx_number_exponent_tl`.)

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

72 `\tl_new:N \l_siunitx_number_flex_tl`

(End definition for `\l_siunitx_number_flex_tl`.)

`\l_siunitx_number_parsed_tl` The number parsed into internal format.

73 `\tl_new:N \l_siunitx_number_parsed_tl`

(End definition for `\l_siunitx_number_parsed_tl`.)

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

74 `\tl_new:N \l_siunitx_number_input_tl`

(End definition for `\l_siunitx_number_input_tl`.)

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

75 `\tl_new:N \l_siunitx_number_partial_tl`

(End definition for `\l_siunitx_number_partial_tl`.)

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

76 `\bool_new:N \l_siunitx_number_validate_bool`

(End definition for `\l_siunitx_number_validate_bool`.)

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

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

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

```

92 {
93   { +- } \mp
94   { +- } \pm
95   { << } \ll
96   { <= } \le
97   { >> } \gg
98   { >= } \ge
99 }
100 \group_begin:
101   \char_set_catcode_active:N \-
102   \cs_new_protected:Npx \__siunitx_number_normalize_minus:N #1
103   {
104     \tl_replace_all:Nnn #1
105     { \exp_not:N - } { \token_to_str:N - }
106   }
107 \group_end:

```

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

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

```

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

```

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

```

133 \cs_new_protected:Npn \__siunitx_number_parse_check:
134 {
135   \tl_if_empty:NF \l__siunitx_number_flex_tl
136   {
137     \bool_lazy_and:nnTF
138     {
139       \tl_if_blank_p:f
140       { \exp_after:wN \use_iv:nnnn \l__siunitx_number_parsed_tl }
141     }
142     {
143       \tl_if_blank_p:f
144       { \exp_after:wN \use_iv:nnnn \l__siunitx_number_flex_tl }
145     }
146     {
147       \tl_set:Nx \l__siunitx_number_tmp_tl
148       { \exp_after:wN \use_i:nnnn \l__siunitx_number_flex_tl }
149       \tl_if_in:NVTF \l__siunitx_number_input_uncert_sign_tl
150       \l__siunitx_number_tmp_tl
151       { \__siunitx_number_parse_combine_uncert: }
152       { \tl_clear:N \l__siunitx_number_parsed_tl }
153     }
154     { \tl_clear:N \l__siunitx_number_parsed_tl }
155   }
156   \tl_if_empty:NTF \l__siunitx_number_parsed_tl
157   {
158     \bool_if:NF \l__siunitx_number_validate_bool
159     {
160       \msg_error:nnx { siunitx } { invalid-number }
161       { \exp_not:V \l__siunitx_number_input_tl }
162     }
163   }
164   { \__siunitx_number_parse_finalise: }
165 }

```

(End definition for __siunitx_number_parse_check:.)

__siunitx_number_parse_combine_uncert:
 x_number_parse_combine_uncert_auxi:nnnnnnnn
 itx_number_parse_combine_uncert_auxii:nnnnn
 itx_number_parse_combine_uncert_auxiii:fnnnn
 x_number_parse_combine_uncert_auxiiii:nnnnnn
 x_number_parse_combine_uncert_auxiii:fnnnnn
 mitx_number_parse_combine_uncert_auxiv:nnnn
 _siunitx_number_parse_combine_uncert_auxv:w
 siunitx_number_parse_combine_uncert_auxvi:w

Conversion of a second numerical part to an uncertainty needs a bit of work. The first step is to extract the useful information from the two stores: the sign, integer and decimal parts from the real number and the integer and decimal parts from the second number. That is done using the input stack to avoid lots of assignments.

```

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

```

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

Finally, everything is recombined: the integer part only needs a test to avoid an empty main number.

```

172 \cs_new_protected:Npn
173   \__siunitx_number_parse_combine_uncert_auxi:nnnnnnnn #1#2#3#4#5#6#7#8
174   {
175     \int_compare:nNnTF { \tl_count:n {#6} } > { \tl_count:n {#2} }
176     {
177       \tl_clear:N \l__siunitx_number_parsed_tl
178       \tl_clear:N \l__siunitx_number_flex_tl
179     }
180     {
181       \__siunitx_number_parse_combine_uncert_auxii:fnnnnn
182       { \int_eval:n { \tl_count:n {#3} - \tl_count:n {#7} } }
183       {#2} {#3} {#6} {#7}
184     }
185   }
186 \cs_new_protected:Npn
187   \__siunitx_number_parse_combine_uncert_auxii:nnnnn #1
188   {
189     \__siunitx_number_parse_combine_uncert_auxiii:fnnnnnn
190     { \prg_replicate:nn { \int_abs:n {#1} } { 0 } }
191     {#1}
192   }
193 \cs_generate_variant:Nn \__siunitx_number_parse_combine_uncert_auxii:nnnnn { f }
194 \cs_new_protected:Npn
195   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn #1#2#3#4#5#6
196   {
197     \int_compare:nNnTF {#2} > 0
198     {
199       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
200       {#3} {#4} {#5} { #6 #1 }
201     }
202     {
203       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
204       {#3} { #4 #1 } {#5} {#6}
205     }
206   }
207 \cs_generate_variant:Nn
208   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn { f }
209 \cs_new_protected:Npn
210   \__siunitx_number_parse_combine_uncert_auxiv:nnnn #1#2#3#4
211   {
212     \tl_set:Nx \l__siunitx_number_parsed_tl
213     {
214       { \tl_head:V \l__siunitx_number_parsed_tl }
215       { \exp_not:n {#1} }
216     }
217     {
218       \bool_lazy_and:nnTF
219       { \tl_if_blank_p:n {#2} }
220       { ! \tl_if_blank_p:n {#4} }
221       { 0 }
222       { \exp_not:n {#2} }
223     }
224   }

```

```

224         \__siunitx_number_parse_combine_uncert_auxv:w #3#4
225         \q_recursion_tail \q_recursion_stop
226     }
227 }
228 }

```

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

```

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

```

(End definition for __siunitx_number_parse_combine_uncert: and others.)

```

\__siunitx_number_parse_comparator:
\__siunitx_number_parse_comparator_aux:Nw

```

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

```

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

```

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

```

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

```

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

of the allowed list, then using that in a delimited argument set up. Once an exponent part is found, there is a loop to check that each of the tokens is a digit then a tidy up step to remove any leading zeros.

```

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

```

```

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

```

(End definition for _siunitx_number_parse_exponent: and others.)

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

```

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

```



```

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

```

(End definition for __siunitx_number_parse_finalise: and __siunitx_number_parse_finalise:nw.)

```

\__siunitx_number_parse_loop:
\__siunitx_number_parse_loop_first:N
\__siunitx_number_parse_loop_main:NNNNN
\__siunitx_number_parse_loop_main_end:NN
\__siunitx_number_parse_loop_main_digit:NNNNN
\__siunitx_number_parse_loop_main_decimal:NN
\__siunitx_number_parse_loop_main_uncert:NNN
\__siunitx_number_parse_loop_main_sign:NNN
\__siunitx_number_parse_loop_main_store:NNN
\__siunitx_number_parse_loop_after_decimal:NNN
\__siunitx_number_parse_loop_root_swap:NNwNN
\__siunitx_number_parse_loop_break:wN

```

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

```

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

```

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

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

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

```

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

```

```

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

```

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

The possibilities are

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

459   \__siunitx_number_parse_uncert:NN #1
460 }

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

499 \cs_new_protected:Npn \__siunitx_number_parse_loop_break:wN

```

```

500   #1 \q_recursion_stop
501   {
502     \tl_clear:N \l__siunitx_number_flex_tl
503     \tl_clear:N \l__siunitx_number_parsed_tl
504   }

```

(End definition for __siunitx_number_parse_loop: and others.)

__siunitx_number_parse_sign:
__siunitx_number_parse_sign_aux:Nw

The first token of a number after a comparator could be a sign. A quick check is made and if found stored. For the number to be valid it has to be more than just a sign, so the next part of the chain is only called if that is the case.

```

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

```

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

__siunitx_number_parse_uncert:NN
__siunitx_number_parse_uncert:NNNN
__siunitx_number_parse_uncert_auxi:NN
__siunitx_number_parse_uncert_auxii:NN
__siunitx_number_parse_uncert_auxii:N
__siunitx_number_parse_uncert_marker:N
__siunitx_number_parse_uncert_after:N

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

No further tokens are allowed after an uncertainty in parenthesis.

```

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

```

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

2.4 Processing numbers

```

\l_siunitx_number_drop_exponent_bool
\l_siunitx_number_drop_uncertainty_bool
\l_siunitx_number_drop_zero_decimal_bool
\l_siunitx_number_exponent_mode_tl
\l_siunitx_number_exponent_fixed_int
\l_siunitx_number_min_decimal_int
\l_siunitx_number_min_integer_int
\l_siunitx_number_round_half_even_bool
\l_siunitx_number_round_mode_tl
\l_siunitx_number_round_pad_bool
\l_siunitx_number_round_precision_int

630 \keys_define:nn { siunitx }
631 {
632   drop-exponent .bool_set:N =
633     \l_siunitx_number_drop_exponent_bool ,
634   drop-uncertainty .bool_set:N =
635     \l_siunitx_number_drop_uncertainty_bool ,
636   drop-zero-decimal .bool_set:N =
637     \l_siunitx_number_drop_zero_decimal_bool ,
638   exponent-mode .choices:nn =
639     { engineering , fixed , input , scientific }
640     { \tl_set_eq:NN \l_siunitx_number_exponent_mode_tl \l_keys_choice_tl } ,
641   fixed-exponent .int_set:N =
642     \l_siunitx_number_exponent_fixed_int ,
643   minimum-decimal-digits .int_set:N =
644     \l_siunitx_number_min_decimal_int ,
645   minimum-integer-digits .int_set:N =
646     \l_siunitx_number_min_integer_int ,
647   round-half .choice: ,
648   round-half / even .code:n =
649     { \bool_set_true:N \l_siunitx_number_round_half_even_bool } ,
650   round-half / up .code:n =
651     { \bool_set_false:N \l_siunitx_number_round_half_even_bool } ,
652   round-minimum .code:n =
653     { \_siunitx_number_set_round_min:n {#1} } ,
654   round-mode .choices:nn =
655     { figures , none , places , uncertainty }
656     { \tl_set_eq:NN \l_siunitx_number_round_mode_tl \l_keys_choice_tl } ,
657   round-pad .bool_set:N =
658     \l_siunitx_number_round_pad_bool ,
659   round-precision .int_set:N =
660     \l_siunitx_number_round_precision_int ,
661 }
662 \bool_new:N \l_siunitx_number_round_half_even_bool
663 \tl_new:N \l_siunitx_number_exponent_mode_tl
664 \tl_new:N \l_siunitx_number_round_mode_tl

```

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

`\l_siunitx_number_round_min_tl` For storing the minimum for rounding.

```
665 \tl_new:N \l_siunitx_number_round_min_tl
```

(End definition for `\l_siunitx_number_round_min_tl`.)

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

```

666 \cs_new_protected:Npn \_siunitx_number_set_round_min:n #1
667 {
668   \siunitx_number_parse:nN {#1} \l_siunitx_number_tmp_tl
669   \exp_after:wN \_siunitx_number_set_round_min:nnnnnnn \l_siunitx_number_tmp_tl
670 }

```



```

671 \cs_new:Npn \__siunitx_number_set_round_min:nnnnnnn #1#2#3#4#5#6#7
672 {
673   \tl_set:Nx \l__siunitx_number_round_min_tl
674   {
675     \bool_lazy_and:nnF
676     { \str_if_eq_p:nn {#3} { 0 } }
677     {
678       \str_if_eq_p:ee
679       { \exp_not:n {#4} }
680       { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
681     }
682     { \exp_not:n { {#3} {#4} } }
683   }
684 }

```

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

\siunitx_number_process:NN

A top-level interface for the processing tools.

__siunitx_number_process:nnnnnnnNN

```

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

```

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

__siunitx_number_exponent:NN

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

siunitx_number_exponent_engineering:nnnnnnn

__siunitx_number_exponent_fixed:nnnnnnn

__siunitx_number_exponent_input:nnnnnnn

__siunitx_number_exponent_scientific:nnnnnnn

__siunitx_number_exponent_fixed:nnnnnnnn

siunitx_number_exponent_scientific:nnnnnnnn

__siunitx_number_exponent_scientific:nnnw

__siunitx_number_exponent_shift:nnn

__siunitx_number_exponent_shift:nnf

__siunitx_number_exponent_shift_down:nnnw

__siunitx_number_exponent_shift_down:nnn

__siunitx_number_exponent_shift_down:nw

__siunitx_number_exponent_shift_up:nnn

__siunitx_number_exponent_shift_up:nnw

__siunitx_number_exponent_shift_up_aux:nnn

__siunitx_number_exponent_shift_up_aux:fnn

```

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

```

```

713 {
714   \cs:w
715     __siunitx_number_exponent_ \l__siunitx_number_exponent_mode_tl :nnnnnnn
716     \exp_after:wN
717     \cs_end: #1
718   }
719   \str_if_eq:VnT \l__siunitx_number_exponent_mode_tl { engineering }
720   {
721     \tl_set:Nx #2
722     { \exp_after:wN \__siunitx_number_exponent_engineering_aux:nnnnnn #2 }
723   }
724 }
725 \cs_new:Npn \__siunitx_number_exponent_fixed:nnnnnnn #1#2#3#4#5#6#7
726 {
727   \exp_args:Nf \__siunitx_number_exponent_fixed:nnnnnnn
728   { \int_eval:n { \l__siunitx_number_exponent_fixed_int - (#6#7) } }
729   {#1} {#2} {#3} {#4} {#5} {#6} {#7}
730 }
731 \cs_new:Npn \__siunitx_number_exponent_fixed:nnnnnnnn #1#2#3#4#5#6#7#8
732 {
733   \exp_not:n { {#2} {#3} }
734   \__siunitx_number_exponent_shift:nnn {#1} {#4} {#5}
735   \__siunitx_number_exponent_uncert:n {#6}
736   \exp_not:n { {#7} } { \int_use:N \l__siunitx_number_exponent_fixed_int }
737 }
738 \cs_new:Npn \__siunitx_number_exponent_input:nnnnnnn #1#2#3#4#5#6#7
739 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }

```

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

```

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

```

```

763 }
764 \cs_new_eq:NN \__siunitx_number_exponent_engineering:nnnnnnn
765 \__siunitx_number_exponent_scientific:nnnnnnn
766 \cs_new:Npn \__siunitx_number_exponent_scientific:nnnw #1#2#3#4#5 \q_stop
767 {
768   \str_if_eq:nnTF {#4} { 0 }
769   {
770     \__siunitx_number_exponent_scientific:nnnw
771     { #1 - 1 } {#2} {#3} #5 \q_stop
772   }
773   {
774     \exp_not:n { {#4} {#5} {#2} }
775     \__siunitx_number_exponent_finalise:n { #1 + #3 - 1 }
776   }
777 }

```

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

```

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

```

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

```

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

```

For shifting the exponent up, we can run out of decimal digits, at which point filling is easy. Other than that a simple loop as we are picking input off the front of the decimal part. We also need to deal with leading zeros: these cannot accumulate.

```

807 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnn #1#2#3
808 {
809   \tl_if_blank:nTF {#3}
810   {
811     \__siunitx_number_exponent_shift_up_aux:ffn
812     { \int_eval:n { #1 + 1 } }
813     { \str_if_eq:nnF {#2} { 0 } {#2} 0 }
814     { }
815     \__siunitx_number_exponent_shift_uncert:nw { 1 }
816   }
817   { \__siunitx_number_exponent_shift_up:nnw {#1} {#2} #3 \q_stop }
818 }
819 \cs_new:Npn \__siunitx_number_exponent_shift_up:nnw #1#2#3#4 \q_stop
820 {
821   \__siunitx_number_exponent_shift_up_aux:ffn
822   { \int_eval:n { #1 + 1 } }
823   { \str_if_eq:nnF {#2} { 0 } {#2} #3 }
824   {#4}
825 }
826 \cs_new:Npn \__siunitx_number_exponent_shift_up_aux:nnn #1#2#3
827 {
828   \int_compare:nNnTF {#1} = 0
829   { \exp_not:n { {#2} {#3} } }
830   {
831     \tl_if_blank:nTF {#3}
832     {
833       {
834         \exp_not:n {#2}
835         \prg_replicate:nn { \int_abs:n {#1} } { 0 }
836       }
837       { }
838       \__siunitx_number_exponent_shift_uncert:nw { \int_abs:n {#1} }
839     }
840     { \__siunitx_number_exponent_shift_up:nnn {#1} {#2} {#3} }
841   }
842 }
843 \cs_generate_variant:Nn \__siunitx_number_exponent_shift_up_aux:nnn { f , ff }

```

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

```

844 \cs_new:Npn \__siunitx_number_exponent_shift_uncert:nw
845   #1#2 \__siunitx_number_exponent_uncert:n #3
846 {
847   \tl_if_blank:nTF {#3}
848   {
849     #2
850     \__siunitx_number_exponent_uncert:n { }
851   }
852   {
853     \str_if_eq:nnTF {#3} { 0 }
854     {
855       #2
856       \__siunitx_number_exponent_uncert:n { { S } { 0 } }

```

```

857     }
858     {
859         \use:c { __siunitx_number_exponent_shift_uncert_ \use_i:nn #3 :fnnn }
860         { \prg_replicate:nn {#1} { 0 } }
861         {#2}
862         #3
863     }
864 }
865 }
866 \cs_new:Npn \__siunitx_number_exponent_shift_uncert_S:nnnn #1#2#3#4
867 {
868     #2
869     \__siunitx_number_exponent_uncert:n { { S } { #4#1 } }
870 }
871 \cs_generate_variant:Nn \__siunitx_number_exponent_shift_uncert_S:nnnn { f }
872 \cs_new:Npn \__siunitx_number_exponent_uncert:n #1 { { \exp_not:n {#1} } }

```

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

```

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

```

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

```

880 \cs_new:Npn \__siunitx_number_exponent_engineering_aux:nnnnnnn #1#2#3#4#5#6#7
881 {
882     \exp_not:n { {#1} {#2} }
883     \use:c
884     {
885         __siunitx_number_exponent_engineering_
886         \int_compare:nNnTF {#6#7} < 0
887         {
888             \int_case:nnF { \int_mod:nn { #7 } { 3 } }
889             {
890                 { 1 } { 2 }
891                 { 2 } { 1 }
892             }
893             { 0 }
894         }
895         { \int_mod:nn {#7} { 3 } }
896         :nnnn
897     }
898     {#3} {#4} {#5} {#6#7}
899 }
900 \cs_new:Npn \__siunitx_number_exponent_engineering_0:nnnn { #1#2#3#4
901 {
902     \exp_not:n { {#1} {#2} {#3} }
903     \__siunitx_number_exponent_finalise:n {#4}
904 }

```

```

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

```

(End definition for __siunitx_number_exponent:NN and others.)

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

```

958 \\cs_new_protected:Npn \\_siunitx_number_digits:NN #1#2
959 {
960   \\tl_set:Nx #2
961   { \\exp_after:wN \\_siunitx_number_digits:nnnnnnn #1 }
962 }
963 \\cs_new:Npn \\_siunitx_number_digits:nnnnnnn #1#2#3#4#5#6#7
964 {
965   \\exp_not:n { {#1} {#2} }
966   {
967     \\_siunitx_number_digits:Nn \\l__siunitx_number_min_integer_int {#3}
968     \\exp_not:n {#3}
969   }
970   {
971     \\exp_not:n {#4}
972     \\_siunitx_number_digits:Nn \\l__siunitx_number_min_decimal_int {#4}
973   }
974   { \\tl_if_blank:nF {#5} { \\_siunitx_number_digits_uncert:nn #5 } }
975   \\exp_not:n { {#6} {#7} }
976 }
977 \\cs_new:Npn \\_siunitx_number_digits:Nn #1#2
978 {
979   \\int_compare:nNnT
980     { #1 - \\tl_count:n {#2} } > 0
981     { \\prg_replicate:nn { #1 - \\tl_count:n {#2} } { 0 } }
982 }
983 \\cs_new:Npn \\_siunitx_number_digits_uncert:nn #1#2
984 {
985   { #1 }
986   { \\use:c { \\_siunitx_number_digits_uncert_ #1 :n } {#2} }
987 }
988 \\cs_new:Npn \\_siunitx_number_digits_uncert_S:n #1
989 {
990   \\exp_not:n {#1}
991   \\_siunitx_number_digits:Nn \\l__siunitx_number_min_decimal_int {#1}
992 }

```

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

`_siunitx_number_drop_exponent:NN` Simple stripping of the exponent.
`_siunitx_number_drop_exponent:nnnnnnn`

```

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

```

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

Simple stripping of the uncertainty.

```

1003 \cs_new_protected:Npn \__siunitx_number_drop_uncertainty:NN #1#2
1004 {
1005     \bool_if:NTF \l__siunitx_number_drop_uncertainty_bool
1006     {
1007         \tl_set:Nx #2
1008             { \exp_after:wN \__siunitx_number_drop_uncertainty:nnnnnnn #1 }
1009     }
1010     { \tl_set_eq:NN #2 #1 }
1011 }
1012 }
1013 \cs_new:Npn \__siunitx_number_drop_uncertainty:nnnnnnn #1#2#3#4#5#6#7
1014 { \exp_not:n { {#1} {#2} {#3} {#4} { } {#6} {#7} } }

```

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

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

```

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

```

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

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

```

1027 \cs_new:Npn \__siunitx_number_round:nnn #1#2#3
1028 {
1029     \__siunitx_number_round_auxi:nnnN {#1} {#2} { }
1030     #3 \q_recursion_tail \q_recursion_stop
1031 }
1032 \cs_generate_variant:Nn \__siunitx_number_round:nnn { f }
1033 \cs_new:Npn \__siunitx_number_round_auxi:nnnN #1#2#3#4
1034 {
1035     \quark_if_recursion_tail_stop_do:Nn #4
1036     {
1037         \__siunitx_number_round_auxii:nnnN {#1} {#3} { } #2
1038         \q_recursion_tail \q_recursion_stop
1039     }
1040     \__siunitx_number_round_auxi:nnnN {#1} {#2} {#4#3}
1041 }
1042 \cs_new:Npn \__siunitx_number_round_auxii:nnnN #1#2#3#4
1043 {

```



```

1044 \quark_if_recursion_tail_stop_do:Nn #4
1045 {
1046   \tl_if_blank:nTF {#2}
1047   {
1048     \__siunitx_number_round_auxiv:nnnN {#1} { } { } #3
1049     \q_recursion_tail \q_recursion_stop
1050   }
1051   {
1052     \__siunitx_number_round_auxiii:nnnN {#1} {#3} { } #2
1053     \q_recursion_tail \q_recursion_stop
1054   }
1055 }
1056 \__siunitx_number_round_auxii:nnnN {#1} {#2} {#4#3}
1057 }

```

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

```

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

```

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

```

1083 \cs_new:Npn \__siunitx_number_round_auxv:nnN #1#2#3
1084 {
1085   \quark_if_recursion_tail_stop_do:Nn #3
1086   {
1087     \__siunitx_number_round_auxvi:nnN
1088     {#1} { } #2 \q_recursion_tail \q_recursion_stop

```

```

1089     }
1090     \bool_lazy_or:nnTF
1091     { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
1092     {
1093         \bool_lazy_all_p:n
1094         {
1095             { \l__siunitx_number_round_half_even_bool }
1096             { \int_if_odd_p:n {#3} }
1097             { \__siunitx_number_round_if_half_p:n {#1} }
1098         }
1099     }
1100     { \__siunitx_number_round_final_decimal:nnw }
1101     { \__siunitx_number_round_auxvii:nnN }
1102     {#2} { } #3
1103 }
1104 \cs_new:Npn \__siunitx_number_round_auxvi:nnnN #1#2#3
1105 {
1106     \quark_if_recursion_tail_stop_do:Nn #3
1107     { { 0 } { } }
1108     \bool_lazy_or:nnTF
1109     { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
1110     {
1111         \bool_lazy_all_p:n
1112         {
1113             { \l__siunitx_number_round_half_even_bool }
1114             { \int_if_odd_p:n {#3} }
1115             { \__siunitx_number_round_if_half_p:n {#1} }
1116         }
1117     }
1118     { \__siunitx_number_round_final_integer:nnw }
1119     { \__siunitx_number_round_auxviii:nnN }
1120     { } {#2} #3
1121 }

```

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

```

1122 \cs_new:Npn \__siunitx_number_round_auxvii:nnN #1#2#3
1123 {
1124     \quark_if_recursion_tail_stop_do:Nn #3
1125     {
1126         \str_if_eq:nnTF {#1} { 0 }
1127         {
1128             \__siunitx_number_round_final_output:ff
1129             { 1 }
1130             { \__siunitx_number_round_truncate:n {#2} }
1131         }
1132         {
1133             \__siunitx_number_round_auxviii:nnN {#2} { } #1
1134             \q_recursion_tail \q_recursion_stop
1135         }
1136     }

```

```

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

```

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

```

1179 \cs_new:Npn \__siunitx_number_round_final_decimal:nnw
1180 #1#2#3 \q_recursion_tail \q_recursion_stop
1181 {
1182   \__siunitx_number_round_final_output:ff
1183   { \tl_reverse:n {#1} }
1184   { \tl_reverse:n {#3} #2 }
1185 }
1186 \cs_new:Npn \__siunitx_number_round_final_integer:nnw
1187 #1#2#3 \q_recursion_tail \q_recursion_stop
1188 {
1189   \__siunitx_number_round_final_output:ff

```

```

1190     { \tl_reverse:n {#3} #2 }
1191     {#1}
1192   }
1193   \cs_new:Npn \__siunitx_number_round_final_output:nn #1#2 { {#1} {#2} }
1194   \cs_generate_variant:Nn \__siunitx_number_round_final_output:nn { ff }
1195   \cs_new:Npn \__siunitx_number_round_final:nn #1#2
1196     { #2 #1 }
1197   \cs_generate_variant:Nn \__siunitx_number_round_final:nn { f }

```

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

```

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

```

```

1238     { 1 } {#1} { \__siunitx_number_round_truncate_direct:n {#2} }
1239   { }
1240   \__siunitx_number_round_final_shift:Nw 1
1241 }
1242 \cs_new:Npn \__siunitx_number_round_final_shift:Nw #1#2 \__siunitx_number_round_places_end:nn
1243 { \__siunitx_number_exponent_finalise:n { #3#4 + #1 } }

```

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

```

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

```

(End definition for __siunitx_number_round:nnn and others.)

__siunitx_number_round_if_half_p:n
 __siunitx_number_round_if_half:N

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

```

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

```

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

__siunitx_number_round_pad:nnn

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

```

1277 \cs_new:Npn \__siunitx_number_round_pad:nnn #1#2#3

```

```

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

```

(End definition for __siunitx_number_round_pad:nnn.)

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

```

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

```

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

```

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

```

```

1323     }
1324     \exp_args:Nf \__siunitx_number_round_figures_count:nnnN
1325     { \int_eval:n { #1 + 1 } } {#2} {#3}
1326 }

```

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

```

\__siunitx_number_round_places:nnnnnnn
\__siunitx_number_round_places_end:nn
\__siunitx_number_round_places_decimal:nn
\__siunitx_number_round_places_integer:nn
\__siunitx_number_round_places_finalise:n
siunitx_number_round_places_finalise:nnnnnnn
__siunitx_number_round_places_finalise:nnnnn

```

The first step when rounding to a fixed number of places is to establish if this is in the decimal or integer parts. The two require different calculations for how many digits to drop from the input. The no-op end function here is to allow tidying up in some cases: see the finalisation of rounding.

```

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

```

```

1369     {#1} {#2}
1370 }

```

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

```

1371 \cs_new:Npn \__siunitx_number_round_places_finalise:n #1
1372 { \__siunitx_number_round_places_finalise:nnnnnnn #1 }
1373 \cs_new:Npn \__siunitx_number_round_places_finalise:nnnnnnn #1#2#3#4#5#6#7
1374 {
1375   \bool_lazy_and:nnTF
1376   { \str_if_eq_p:nn {#3} { 0 } }
1377   {
1378     \str_if_eq_p:ee
1379     { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1380   }
1381   {
1382     \tl_if_empty:NTF \l__siunitx_number_round_min_tl
1383     {
1384       \exp_not:n { {#1} }
1385       { \str_if_eq:nnF {#2} { - } { \exp_not:n {#2} } }
1386       \exp_not:n { {#3} {#4} {#5} {#6} {#7} }
1387     }
1388     {
1389       \exp_after:wN \__siunitx_number_round_places_finalise:nnnnn
1390       \l__siunitx_number_round_min_tl {#2} {#6} {#7}
1391     }
1392   }
1393   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1394 }
1395 \cs_new:Npn \__siunitx_number_round_places_finalise:nnnnn #1#2#3#4#5
1396 {
1397   {
1398     \str_if_eq:nnTF {#3} { - }
1399     { > }
1400     { < }
1401   }
1402   \exp_not:n { {#3} {#1} {#2} { } {#4} {#5} }
1403 }

```

(End definition for __siunitx_number_round_places:nnnnnnn and others.)

__siunitx_number_round_uncertainty:nnnnnnn
 __siunitx_number_round_uncertainty:nnn
 __siunitx_number_round_uncertainty:nnnn
 __siunitx_number_round_uncertainty_aux:nnnnn
 siunitx_number_round_uncertainty_aux:nnnnnn

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

```

1404 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnnnn #1#2#3#4#5#6#7
1405 {
1406   \bool_lazy_or:nnTF
1407   { \tl_if_blank_p:n {#5} }
1408   { ! \int_compare_p:nNn \l__siunitx_number_round_precision_int > 0 }
1409   { \exp_not:n { {#1} #2 {#3} {#4} { } #6 {#7} } }
1410   {
1411     \str_if_eq:eeTF { \tl_head:n {#5} } { S }
1412     {

```



```

1413         \exp_not:n { {#1} {#2} }
1414         \exp_args:Nnno \__siunitx_number_round_uncertainty:nnn
1415         {#3} {#4} { \use_ii:nn #5 }
1416         \exp_not:n { {#6} {#7} }
1417     }
1418     { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1419 }
1420 }

```

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

```

1421 \cs_new:Npn \__siunitx_number_round_uncertainty:nnn #1#2#3
1422 {
1423     \exp_last_unbraced:Nf \__siunitx_number_round_uncertainty:nnnnn
1424     {
1425         \__siunitx_number_round:fnn
1426         {
1427             \int_eval:n
1428             { \tl_count:n {#3} - \l__siunitx_number_round_precision_int }
1429         }
1430         { } {#3}
1431     }
1432     {#1} {#2} {#3}
1433 }
1434 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnn #1#2#3#4#5
1435 {
1436     \exp_args:Nf \__siunitx_number_round_uncertainty_aux:nnnnn
1437     { \int_eval:n { \tl_count:n {#5} - \tl_count:n {#2} } }
1438     {#1} {#2} {#3} {#4}
1439 }

```

The first argument here deals with the case where we've lost digits in the uncertainty and it's purely located in the integer part.

```

1440 \cs_new:Npn \__siunitx_number_round_uncertainty_aux:nnnnn #1#2#3#4#5
1441 {
1442     \exp_args:Nf \__siunitx_number_round_uncertainty_aux:nnnnnn
1443     {
1444         \tl_if_blank:nT {#5}
1445         { \prg_replicate:nn {#1} { 0 } }
1446     }
1447     {#1} {#2} {#3} {#4} {#5}
1448 }
1449 \cs_new:Npn \__siunitx_number_round_uncertainty_aux:nnnnnn #1#2#3#4#5#6
1450 {
1451     \tl_if_blank:nTF {#3}
1452     {
1453         \__siunitx_number_round:nnn
1454         {#2}
1455         {#5} {#6}
1456         { { S } { #4 #1 } }
1457     }
1458     {
1459         \__siunitx_number_round:fnn
1460         { \int_eval:n { #2 + 1 } }

```

```

1461         {#5} {#6}
1462         { { S } { #3 \_siunitx_number_round_truncate_direct:n {#4} #1 } }
1463     }
1464 }

```

(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
1465 \cs_new_protected:Npn \_siunitx_number_zero_decimal:NN #1#2
1466 {
1467     \bool_if:NT \l__siunitx_number_drop_zero_decimal_bool
1468     {
1469         \tl_set:Nx #2
1470         { \exp_after:wN \_siunitx_number_zero_decimal:nnnnnnn #1 }
1471     }
1472 }
1473 \cs_new:Npn \_siunitx_number_zero_decimal:nnnnnnn #1#2#3#4#5#6#7
1474 {
1475     \exp_not:n { {#1} {#2} {#3} }
1476     \str_if_eq:eeTF
1477     { \exp_not:n {#4} }
1478     { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1479     { { } }
1480     { \exp_not:n { {#4} } }
1481     \exp_not:n { {#5} {#6} {#7} }
1482 }

```

(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
1483 \cs_new:Npn \siunitx_number_adjust_exponent:nn #1#2
1484 { \_siunitx_number_adjust_exp:nnnnnnnn #1 {#2} }
1485 \cs_new:Npn \siunitx_number_adjust_exponent:Nn #1#2
1486 {
1487     \tl_if_empty:NF #1
1488     { \exp_args:NV \siunitx_number_adjust_exponent:nn #1 {#2} }
1489 }
1490 \cs_new:Npn \_siunitx_number_adjust_exp:nnnnnnnn #1#2#3#4#5#6#7#8
1491 {
1492     \exp_not:n { {#1} {#2} {#3} {#4} {#5} }
1493     \exp_args:Ne \_siunitx_number_adjust_exp:nn { \fp_eval:n { #6#7 + #8 } } {#6}
1494 }
1495 \cs_new:Npn \_siunitx_number_adjust_exp:nn #1#2
1496 { \_siunitx_number_adjust_exp:nNw {#2} #1 \q_stop }
1497 \cs_new:Npn \_siunitx_number_adjust_exp:nNw #1#2#3 \q_stop
1498 {
1499     \token_if_eq_meaning:NNTF #2 -
1500     { { - } { \exp_not:n {#3} } }
1501     { { \str_if_eq:nnT {#1} { + } { + } } { \exp_not:n {#2#3} } }
1502 }

```

(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`

```
1503 \tl_new:N \l_siunitx_number_bracket_close_tl
1504 \tl_new:N \l_siunitx_number_bracket_open_tl
1505 \tl_set:Nn \l_siunitx_number_bracket_open_tl { ( }
1506 \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`

```
1507 \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`

```
1508 \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.)

`\l_siunitx_number_bracket_negative_bool`

Keys producing tokens in the output.

`\l_siunitx_number_implicit_plus_bool`

```
1509 \keys_define:nn { siunitx }
```

`\l_siunitx_number_exponent_base_tl`

```
1510 {
```

`\l_siunitx_number_exponent_product_tl`

```
1511   bracket-ambiguous-numbers .bool_set:N =
```

`\l_siunitx_number_group_decimal_bool`

```
1512   \l_siunitx_number_bracket_ambiguous_bool ,
```

`\l_siunitx_number_group_integer_bool`

```
1513   bracket-negative-numbers .bool_set:N =
```

`\l_siunitx_number_group_minimum_int`

```
1514   \l_siunitx_number_bracket_negative_bool ,
```

`\l_siunitx_number_group_separator_tl`

```
1515   exponent-base .tl_set:N =
```

`\l_siunitx_number_negative_color_tl`

```
1516   \l_siunitx_number_exponent_base_tl ,
```

`\l_siunitx_number_output_exp_marker_tl`

```
1517   exponent-product .tl_set:N =
```

`\l_siunitx_number_output_uncert_close_tl`

```
1518   \l_siunitx_number_exponent_product_tl ,
```

`\l_siunitx_number_output_uncert_open_tl`

```
1519   group-digits .choice: ,
```

`\l_siunitx_number_uncert_mode_tl`

```
1520   group-digits / all .code:n =
```

`\l_siunitx_number_uncert_separator_tl`

```
1521   {
```

`\l_siunitx_number_tight_bool`

```
1522     \bool_set_true:N \l_siunitx_number_group_decimal_bool
```

`\l_siunitx_number_unity_mantissa_bool`

```
1523     \bool_set_true:N \l_siunitx_number_group_integer_bool
```

`\l_siunitx_number_zero_exponent_bool`

```
1524   } ,
```

```
1525   group-digits / decimal .code:n =
```

```
1526   {
```

```
1527     \bool_set_true:N \l_siunitx_number_group_decimal_bool
```

```
1528     \bool_set_false:N \l_siunitx_number_group_integer_bool
```

```
1529   } ,
```

```
1530   group-digits / integer .code:n =
```

```
1531   {
```

```
1532     \bool_set_false:N \l_siunitx_number_group_decimal_bool
```

```
1533     \bool_set_true:N \l_siunitx_number_group_integer_bool
```

```
1534   } ,
```

```
1535   group-digits / none .code:n =
```

```
1536   {
```

```
1537     \bool_set_false:N \l_siunitx_number_group_decimal_bool
```

```
1538     \bool_set_false:N \l_siunitx_number_group_integer_bool
```

```
1539   } ,
```

```
1540   group-digits .default:n = all ,
```

```

1541 group-minimum-digits .int_set:N =
1542   \l__siunitx_number_group_minimum_int ,
1543 group-separator .tl_set:N =
1544   \l__siunitx_number_group_separator_tl ,
1545 negative-color .tl_set:N =
1546   \l__siunitx_number_negative_color_tl ,
1547 output-close-uncertainty .tl_set:N =
1548   \l__siunitx_number_output_uncert_close_tl ,
1549 output-decimal-marker .tl_set:N =
1550   \l__siunitx_number_output_decimal_tl ,
1551 output-exponent-marker .tl_set:N =
1552   \l__siunitx_number_output_exp_marker_tl ,
1553 output-open-uncertainty .tl_set:N =
1554   \l__siunitx_number_output_uncert_open_tl ,
1555 print-implicit-plus .bool_set:N =
1556   \l__siunitx_number_implicit_plus_bool ,
1557 print-unity-mantissa .bool_set:N =
1558   \l__siunitx_number_unity_mantissa_bool ,
1559 print-zero-exponent .bool_set:N =
1560   \l__siunitx_number_zero_exponent_bool ,
1561 tight-spacing .bool_set:N =
1562   \l__siunitx_number_tight_bool ,
1563 uncertainty-mode .choices:nn =
1564   { compact , compact-marker , full , separate }
1565   { \tl_set_eq:NN \l__siunitx_number_uncert_mode_tl \l_keys_choice_tl } ,
1566 uncertainty-separator .tl_set:N =
1567   \l__siunitx_number_uncert_separator_tl
1568 }
1569 \bool_new:N \l__siunitx_number_group_decimal_bool
1570 \bool_new:N \l__siunitx_number_group_integer_bool
1571 \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.

```

1572 \cs_new:Npn \siunitx_number_output:N #1
1573   { \__siunitx_number_output:Nn #1 { } }
1574 \cs_new:Npn \siunitx_number_output:n #1
1575   { \__siunitx_number_output:nn #1 { } }
1576 \cs_new:Npn \siunitx_number_output:NN #1#2
1577   { \__siunitx_number_output:Nn #1 {#2} }
1578 \cs_new:Npn \siunitx_number_output:nN #1#2
1579   { \__siunitx_number_output:nn #1 {#2} }
1580 \cs_new:Npn \__siunitx_number_output:Nn #1#2
1581   {
1582     \tl_if_empty:NF #1
1583       { \exp_after:wN \__siunitx_number_output:nnnnnnn #1 {#2} }
1584   }
1585 \cs_new:Npn \__siunitx_number_output:nn #1#2
1586   {
1587     \tl_if_empty:NF {#1}
1588     { \__siunitx_number_output:nnnnnnn #1 {#2} }

```

```

1589 }
1590 \cs_new:Npn \__siunitx_number_output:nnnnnnn #1#2#3#4#5#6#7#8
1591 {
1592   \__siunitx_number_output_color:n {#2}
1593   \__siunitx_number_output_comparator:nn {#1} {#8}
1594   \__siunitx_number_output_bracket:nn {#5} {#7}
1595   \__siunitx_number_output_sign:nnn {#1} {#2} {#8}
1596   \__siunitx_number_output_integer:nnn {#3} {#4} {#7}
1597   \__siunitx_number_output_decimal:nn {#4} {#8}
1598   \__siunitx_number_output_uncertainty:nnn {#5} {#4} {#8}
1599   \__siunitx_number_output_exponent:nnnn {#6} {#7} { #3 . #4 } {#8}
1600   \__siunitx_number_output_end:
1601 }

```

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.

```

1602 \cs_new:Npn \__siunitx_number_output_bracket:nn #1#2
1603 {
1604   \bool_lazy_all:nT
1605   {
1606     { \str_if_eq_p:Vn \l__siunitx_number_uncert_mode_tl { separate } }
1607     { \l_siunitx_number_bracket_ambiguous_bool }
1608     { ! \tl_if_blank_p:n {#1} }
1609     {
1610       \bool_lazy_or_p:nn
1611       { \l__siunitx_number_zero_exponent_bool }
1612       { ! \str_if_eq_p:nn {#2} { 0 } }
1613     }
1614   }
1615   \__siunitx_number_output_bracket:w
1616 }
1617 \cs_new:Npn \__siunitx_number_output_bracket:w #1 \__siunitx_number_output_exponent:nnnn
1618 {
1619   \exp_not:V \l__siunitx_number_bracket_open_tl
1620   #1
1621   \exp_not:V \l__siunitx_number_bracket_close_tl
1622   \__siunitx_number_output_exponent:nnnn
1623 }

```

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

```

1624 \cs_new:Npn \__siunitx_number_output_color:n #1
1625 {
1626   \bool_lazy_and:nnT
1627   { \str_if_eq_p:nn {#1} { - } }
1628   { ! \tl_if_empty_p:N \l__siunitx_number_negative_color_tl }
1629   { \exp_not:N \color { \exp_not:V \l__siunitx_number_negative_color_tl } }
1630 }

```

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

```

1631 \cs_new:Npn \__siunitx_number_output_comparator:nn #1#2
1632 {
1633   \tl_if_blank:nF {#1}
1634   { \exp_not:n { \mathord {#1} } }

```

```

1635     \exp_not:n {#2}
1636   }

```

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.

```

1637 \cs_new:Npn \__siunitx_number_output_sign:nnn #1#2#3
1638 {
1639   \tl_if_blank:nTF {#2}
1640   {
1641     \bool_if:NT \l__siunitx_number_implicit_plus_bool
1642     { \__siunitx_number_output_sign:nN {#1} + }
1643   }
1644   {
1645     \str_if_eq:nnTF {#2} { - }
1646     {
1647       \bool_if:NTF \l__siunitx_number_bracket_negative_bool
1648       { \__siunitx_number_output_sign_brackets:w }
1649       { \__siunitx_number_output_sign:nN {#1} #2 }
1650     }
1651     { \__siunitx_number_output_sign:nN {#1} #2 }
1652   }
1653   \exp_not:n {#3}
1654 }
1655 \cs_new:Npn \__siunitx_number_output_sign:nN #1#2
1656 {
1657   \tl_if_blank:nTF {#1}
1658   { \__siunitx_number_output_sign:N #2 }
1659   { \exp_not:n { \mathord {#2} } }
1660 }
1661 \cs_new:Npn \__siunitx_number_output_sign:N #1
1662 {
1663   \bool_if:NTF \l__siunitx_number_tight_bool
1664   { \exp_not:n { \mathord {#1} } }
1665   { \exp_not:n {#1} }
1666 }
1667 \cs_new:Npn
1668   \__siunitx_number_output_sign_brackets:w #1 \__siunitx_number_output_end:
1669 {
1670   \exp_not:V \l__siunitx_number_bracket_open_tl
1671   #1
1672   \exp_not:V \l__siunitx_number_bracket_close_tl
1673   \__siunitx_number_output_end:
1674 }

```

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.

```

1675 \cs_new:Npn \__siunitx_number_output_integer:nnn #1#2#3
1676 {
1677   \bool_lazy_or:nnT
1678   { \l__siunitx_number_unity_mantissa_bool }

```

```

1679 { ! \str_if_eq_p:nn { #1 . #2 } { 1. } }
1680 { \__siunitx_number_output_digits:nn { integer } {#1} }
1681 }
1682 \cs_new:Npn \__siunitx_number_output_decimal:nn #1#2
1683 {
1684   \exp_not:n {#2}
1685   \tl_if_blank:nF {#1}
1686   {
1687     \str_if_eq:VnTF \l_siunitx_number_output_decimal_tl { , }
1688     { \exp_not:N \mathord }
1689     { \use:n }
1690     { \exp_not:V \l_siunitx_number_output_decimal_tl }
1691   }
1692   \exp_not:n {#2}
1693   \__siunitx_number_output_digits:nn { decimal } {#1}
1694 }
1695 \cs_generate_variant:Nn \__siunitx_number_output_decimal:nn { f }
1696 \cs_new:Npn \__siunitx_number_output_digits:nn #1#2
1697 {
1698   \bool_if:cTF { l__siunitx_number_group_ #1 _ bool }
1699   {
1700     \int_compare:nNnTF
1701     { \tl_count:n {#2} } < \l__siunitx_number_group_minimum_int
1702     { \exp_not:n {#2} }
1703     { \use:c { __siunitx_number_output_ #1 _aux:n } {#2} }
1704   }
1705   { \exp_not:n {#2} }
1706 }

```

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.

```

1707 \cs_new:Npn \__siunitx_number_output_integer_aux:n #1
1708 {
1709   \use:c
1710   {
1711     __siunitx_number_output_integer_aux_
1712     \int_eval:n { \int_mod:nn { \tl_count:n {#1} } { 3 } }
1713     :n
1714     } {#1}
1715 }
1716 \cs_new:cpn { __siunitx_number_output_integer_aux_0:n } #1
1717 { \__siunitx_number_output_integer_first:nnNN #1 \q_nil }
1718 \cs_new:cpn { __siunitx_number_output_integer_aux_1:n } #1
1719 { \__siunitx_number_output_integer_first:nnNN { } { } #1 \q_nil }
1720 \cs_new:cpn { __siunitx_number_output_integer_aux_2:n } #1
1721 { \__siunitx_number_output_integer_first:nnNN { } #1 \q_nil }
1722 \cs_new:Npn \__siunitx_number_output_integer_first:nnNN #1#2#3#4
1723 {
1724   \exp_not:n {#1#2#3}
1725   \quark_if_nil:NF #4
1726   { \__siunitx_number_output_integer_loop:NNNN #4 }
1727 }
1728 \cs_new:Npn \__siunitx_number_output_integer_loop:NNNN #1#2#3#4

```

```

1729 {
1730   \str_if_eq:VnTF \l__siunitx_number_group_separator_tl { , }
1731   { \exp_not:N \mathord }
1732   { \use:n }
1733   { \exp_not:V \l__siunitx_number_group_separator_tl }
1734   \exp_not:n {#1#2#3}
1735   \quark_if_nil:NF #4
1736   { \__siunitx_number_output_integer_loop:NNNN #4 }
1737 }

```

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.

```

1738 \cs_new:Npn \__siunitx_number_output_decimal_aux:n #1
1739 {
1740   \__siunitx_number_output_decimal_loop:NNNN \c_empty_tl
1741   #1 \q_nil \q_nil \q_nil
1742 }
1743 \cs_new:Npn \__siunitx_number_output_decimal_loop:NNNN #1#2#3#4
1744 {
1745   \quark_if_nil:NF #2
1746   {
1747     \exp_not:V #1
1748     \exp_not:n {#2}
1749     \quark_if_nil:NTF #3
1750     { \use_none:n }
1751     {
1752       \exp_not:n {#3}
1753       \quark_if_nil:NTF #4
1754       { \use_none:nn }
1755       {
1756         \exp_not:n {#4}
1757         \__siunitx_number_output_decimal_loop:NNNN
1758         \l__siunitx_number_group_separator_tl
1759       }
1760     }
1761   }
1762 }

```

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.

```

1763 \cs_new:Npn \__siunitx_number_output_uncertainty:nnn #1#2#3
1764 {
1765   \tl_if_blank:nTF {#1}
1766   { \__siunitx_number_output_uncertainty_unaligned:n {#3} }
1767   {
1768     \use:c { \__siunitx_number_output_uncert_ \tl_head:n {#1} :nnnw }
1769     {#2} {#3} #1
1770   }
1771 }

```



```

1772 \cs_new:Npn \__siunitx_number_output_uncertainty_unaligned:n #1
1773 { \exp_not:n { #1 #1 #1 #1 } }
1774 \cs_new:Npn \__siunitx_number_output_uncert_S:nnnw #1#2#3#4
1775 {
1776   \str_if_eq:VnTF \l__siunitx_number_uncert_mode_tl { separate }
1777   {
1778     \exp_not:n {#2}
1779     \__siunitx_number_output_sign:N \pm
1780     \exp_not:n {#2}
1781     \__siunitx_number_output_uncert_S_aux:nnn
1782     { \int_eval:n { \tl_count:n {#4} - \tl_count:n {#1} } }
1783     {#4} {#2}
1784   }
1785   {
1786     \exp_not:V \l__siunitx_number_uncert_separator_tl
1787     \exp_not:V \l__siunitx_number_output_uncert_open_tl
1788     \use:c { __siunitx_number_output_uncert_S_ \l__siunitx_number_uncert_mode_tl :nn } {#1}
1789     \exp_not:V \l__siunitx_number_output_uncert_close_tl
1790     \__siunitx_number_output_uncertainty_unaligned:n {#2}
1791   }
1792 }
1793 \cs_new:Npn \__siunitx_number_output_uncert_S_aux:nnn #1#2#3
1794 {
1795   \int_compare:nNnTF {#1} > 0
1796   {
1797     \__siunitx_number_output_uncert_S_aux:fnnw
1798     { \int_eval:n { #1 - 1 } }
1799     {#3}
1800     { }
1801     #2 \q_nil
1802   }
1803   {
1804     0
1805     \__siunitx_number_output_decimal:fn
1806     {
1807       \prg_replicate:nn { \int_abs:n {#1} } { 0 }
1808       #2
1809     }
1810     {#3}
1811   }
1812 }
1813 \cs_generate_variant:Nn \__siunitx_number_output_uncert_S_aux:nnn { f }
1814 \cs_new:Npn \__siunitx_number_output_uncert_S_aux:nnnw #1#2#3#4
1815 {
1816   \quark_if_nil:NF #4
1817   {
1818     \int_compare:nNnTF {#1} = 0
1819     { \__siunitx_number_output_uncert_S_aux:nnw {#3#4} {#2} }
1820     {
1821       \__siunitx_number_output_uncert_S_aux:fnnw
1822       { \int_eval:n { #1 - 1 } }
1823       {#2}
1824       {#3#4}
1825     }
1826   }

```

```

1826     }
1827   }
1828   \cs_generate_variant:Nn \__siunitx_number_output_uncert_S_aux:nnnw { f }
1829   \cs_new:Npn \__siunitx_number_output_uncert_S_aux:nnw #1#2#3 \q_nil
1830   {
1831     \__siunitx_number_output_digits:nn { integer } {#1}
1832     \__siunitx_number_output_decimal:nn {#3} {#2}
1833   }

```

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

```

1834   \cs_new:Npn \__siunitx_number_output_uncert_S_compact:nn #1#2
1835   { \exp_not:n {#2} }
1836   \cs_new:cpn { \__siunitx_number_output_uncert_S_compact-marker:nn } #1#2
1837   {
1838     \bool_lazy_or:nnTF
1839     { \tl_if_blank_p:n {#1} }
1840     { ! \int_compare_p:nNn { \tl_count:n {#2} } > { \tl_count:n {#1} } }
1841     { \__siunitx_number_output_uncert_S_compact:nn }
1842     { \__siunitx_number_output_uncert_S_full:nn }
1843     {#1} {#2}
1844   }
1845   \cs_new:Npn \__siunitx_number_output_uncert_S_full:nn #1#2
1846   {
1847     \__siunitx_number_output_uncert_S_aux:fnn
1848     { \int_eval:n { \tl_count:n {#2} - \tl_count:n {#1} } }
1849     {#2} { }
1850   }

```

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.

```

1851   \cs_new:Npn \__siunitx_number_output_exponent:nnnn #1#2#3#4
1852   {
1853     \exp_not:n {#4}
1854     \bool_lazy_or:nnTF
1855     { \l__siunitx_number_zero_exponent_bool }
1856     { ! \str_if_eq_p:nn {#2} { 0 } }
1857     {
1858       \tl_if_empty:NTF \l__siunitx_number_output_exp_marker_tl
1859       { \__siunitx_number_output_exponent_auxi:nnnn }
1860       { \__siunitx_number_output_exponent_auxii:nnnn }
1861       {#1} {#2} {#3} {#4}
1862     }
1863     { \exp_not:n {#4} }
1864   }
1865   \cs_new:Npn \__siunitx_number_output_exponent_auxi:nnnn #1#2#3#4
1866   {
1867     \bool_lazy_or:nnTF
1868     { \l__siunitx_number_unity_mantissa_bool }
1869     { ! \str_if_eq_p:nn {#3} { 1. } }
1870     {

```

```

1871 \bool_if:NTF \l__siunitx_number_tight_bool
1872 { \exp_not:N \mathord }
1873 { \use:n }
1874 { \exp_not:V \l__siunitx_number_exponent_product_tl }
1875 \exp_not:n {#4}
1876 }
1877 { \exp_not:n {#4} }
1878 \exp_not:V \l__siunitx_number_exponent_base_tl
1879 ^
1880 { \__siunitx_number_output_exponent_auxiii:nn {#1} {#2} }
1881 }
1882 \cs_new:Npn \__siunitx_number_output_exponent_auxii:nnnn #1#2#3#4
1883 {
1884 \exp_not:n {#4}
1885 \exp_not:V \l__siunitx_number_output_exp_marker_tl
1886 \__siunitx_number_output_exponent_auxiii:nn {#1} {#2}
1887 }
1888 \cs_new:Npn \__siunitx_number_output_exponent_auxiii:nn #1#2
1889 {
1890 \tl_if_blank:NTF {#1}
1891 {
1892 \bool_if:NT \l__siunitx_number_implicit_plus_bool
1893 { \__siunitx_number_output_sign:N + }
1894 }
1895 { \__siunitx_number_output_sign:N #1 }
1896 \__siunitx_number_output_digits:nn { integer } {#2}
1897 }

```

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

```

1898 \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.

```

1899 \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.

```

1900 \prg_new_protected_conditional:Npnn \siunitx_if_number:n #1
1901 { T , F , TF }
1902 {
1903 \group_begin:
1904 \bool_set_true:N \l__siunitx_number_validate_bool
1905 \bool_set_true:N \l__siunitx_number_parse_bool
1906 \siunitx_number_parse:nN {#1} \l__siunitx_number_parsed_tl
1907 \tl_if_empty:NTF \l__siunitx_number_parsed_tl
1908 {
1909 \group_end:

```

```

1910         \prg_return_false:
1911     }
1912     {
1913         \group_end:
1914         \prg_return_true:
1915     }
1916 }

```

(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.

```

1917 \prg_new_conditional:Npnn \siunitx_if_number_token:N #1
1918 { p , T , F , TF }
1919 {
1920     \__siunitx_number_token_auxi:NN #1
1921     \l_siunitx_number_input_decimal_tl
1922     \l_siunitx_number_input_uncert_close_tl
1923     \l_siunitx_number_input_comparator_tl
1924     \l_siunitx_number_input_digit_tl
1925     \l_siunitx_number_input_exponent_tl
1926     \l_siunitx_number_input_ignore_tl
1927     \l_siunitx_number_input_uncert_open_tl
1928     \l_siunitx_number_input_sign_tl
1929     \l_siunitx_number_input_uncert_sign_tl
1930     \q_recursion_tail
1931     \q_recursion_stop
1932 }
1933 \cs_new:Npn \__siunitx_number_token_auxi:NN #1#2
1934 {
1935     \quark_if_recursion_tail_stop_do:Nn #2 { \prg_return_false: }
1936     \__siunitx_number_token_auxii:NN #1 #2
1937     \__siunitx_number_token_auxi:NN #1
1938 }
1939 \cs_new:Npn \__siunitx_number_token_auxii:NN #1#2
1940 {
1941     \exp_after:wN \__siunitx_number_token_auxiii:NN \exp_after:wN #1
1942     #2 \q_recursion_tail \q_recursion_stop
1943 }
1944 \cs_new:Npn \__siunitx_number_token_auxiii:NN #1#2
1945 {
1946     \quark_if_recursion_tail_stop:N #2
1947     \str_if_eq:nnT {#1} {#2}
1948     {
1949         \use_i_delimit_by_q_recursion_stop:nw
1950         {
1951             \use_i_delimit_by_q_recursion_stop:nw
1952             { \prg_return_true: }
1953         }
1954     }
1955     \__siunitx_number_token_auxiii:NN #1
1956 }

```

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

2.8 Messages

```

1957 \msg_new:nnnn { siunitx } { invalid-number }
1958   { Invalid-number~'#1'. }
1959   {
1960     The~input~'#1'~could-not~be~parsed~as~a~number~following~the~
1961     format~defined~in~module~documentation.
1962   }

```

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.

```

1963 \keys_set:nn { siunitx }
1964 {
1965   bracket-ambiguous-numbers = true ,
1966   bracket-negative-numbers = false ,
1967   drop-exponent = false ,
1968   drop-uncertainty = false ,
1969   drop-zero-decimal = false ,
1970   evaluate-expression = false ,
1971   exponent-base = 10 ,
1972   exponent-mode = input ,
1973   exponent-product = \times ,
1974   expression = #1 ,
1975   fixed-exponent = 0 ,
1976   group-digits = all ,
1977   group-minimum-digits = 5 ,
1978   group-separator = \, , % (
1979   input-close-uncertainty = ) ,
1980   input-comparators = { <=>\approx\ge\geq\gg\le\leq\ll\sim } ,
1981   input-decimal-markers = { . , } ,
1982   input-digits = 0123456789 ,
1983   input-exponent-markers = dDeE ,
1984   input-ignore = \, ,
1985   input-open-uncertainty = ( , % )
1986   input-signs = +- \mp \pm ,
1987   input-uncertainty-signs = \pm ,
1988   minimum-decimal-digits = 0 ,
1989   minimum-integer-digits = 0 ,
1990   negative-color = , % (
1991   output-close-uncertainty = ) ,
1992   output-decimal-marker = . ,
1993   output-open-uncertainty = ( , % )
1994   parse-numbers = true ,
1995   print-implicit-plus = false ,
1996   print-unity-mantissa = true ,
1997   print-zero-exponent = false ,
1998   retain-explicit-plus = false ,
1999   retain-zero-uncertainty = false ,
2000   round-half = up ,
2001   round-minimum = 0 ,
2002   round-mode = none ,
2003   round-pad = true ,
2004   round-precision = 2 ,

```

```

2005     tight-spacing           = false           ,
2006     uncertainty-mode        = compact          ,
2007     uncertainty-separator    =
2008   }
2009 \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</code>	<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</code>	<code>mode = match math text</code> Selects which mode (math or text) the output is printed in: a choice from the options <code>match</code> , <code>math</code> or <code>text</code> . The option <code>match</code> matches the mode prevailing at the point <code>\siunitx_print_...: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</code>	<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</code>	<code>number-mode = match math text</code> Selects which mode (math or text) the numbers are printed in: a choice from the options <code>match</code> , <code>math</code> or <code>text</code> . The option <code>match</code> matches the mode prevailing at the point <code>\siunitx_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</code>	<code>propagate-math-font = true false</code> Switch to determine if the currently-active math font is applied within printed output. This is relevant only when <code>\siunitx_print_...: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</code>	<code>reset-math-version = true false</code> Switch to determine whether the active <code>\mathversion</code> is reset to <code>normal</code> when printing in math mode. Note that math version is typically used to select <code>\boldmath</code> , though it is also be used by <i>e.g.</i> <code>sansmath</code> . The standard setting is <code>true</code> .
<hr/> <hr/> <code>reset-text-family</code>	<code>reset-text-family = true false</code> Switch to determine whether the active 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</code>	<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</code>	<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 <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_EX mode for printing. The standard setting is <code>math</code>.</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 <code>m</code> weight maps to normal math version whilst all of the <code>b</code> weights map to bold and all of the <code>l</code> 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_weight_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_weight_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
\l__siunitx_print_version_el_tl
\l__siunitx_print_version_l_tl
\l__siunitx_print_version_sl_tl
\l__siunitx_print_version_m_tl
\l__siunitx_print_version_sb_tl
\l__siunitx_print_version_b_tl
\l__siunitx_print_version_eb_tl
\l__siunitx_print_version_ub_tl

```

One set of “focussed” options.

```

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

```

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

```

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 91.)

`\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 91.)

`__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_weight_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_weight_ #1 _tl } { m } }

```

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

`\c_siunitx_print_weight_l_tl` Font widths with one letter.

```

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

```

(End definition for `\c_siunitx_print_weight_l_tl`, `\c_siunitx_print_weight_m_tl`, and `\c_siunitx_print_weight_b_tl`.)

`\siunitx_print_math:n`

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

```

99 \cs_new_protected:Npn \siunitx_print_math:n #1
100 {
101     \bool_if:NTF \l__siunitx_print_math_weight_bool
102     {
103         \tl_set:Nx \l__siunitx_print_tmp_tl
104         { \exp_after:wN \__siunitx_print_extract_series:Nw \f@series ? \q_stop }
105         \tl_if_empty:NTF \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         { \__siunitx_print_math_auxi:n {#1} }
107         { \__siunitx_print_math_version:Vn \l__siunitx_print_tmp_tl {#1} }
108     }
109     { \__siunitx_print_math_auxi:n {#1} }
110 }

```

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.

```

111 \cs_new:Npn \__siunitx_print_extract_series:Nw #1#2 ? #3 \q_stop
112 {
113     \cs_if_exist:cTF { c__siunitx_print_weight_ #1#2 _tl }
114     { \__siunitx_print_convert_series:v { c__siunitx_print_weight_ #1#2 _tl } }
115     {
116         \cs_if_exist:cTF { c__siunitx_print_weight_ #1 _tl }
117         { \__siunitx_print_convert_series:v { c__siunitx_print_weight_ #1 _tl } }
118         { \__siunitx_print_convert_series:n {#1#2} }
119     }
120 }
121 \cs_new:Npn \__siunitx_print_convert_series:n #1
122 { \tl_use:c { l__siunitx_print_version_ #1 _tl } }
123 \cs_generate_variant:Nn \__siunitx_print_convert_series:n { v }
124 \cs_new_protected:Npn \__siunitx_print_math_auxi:n #1
125 {
126     \bool_if:NTF \l__siunitx_print_math_version_bool
127     { \__siunitx_print_math_version:nn { normal } {#1} }
128     { \__siunitx_print_math_auxii:n {#1} }
129 }

```

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

```

130 \cs_new_protected:Npn \__siunitx_print_math_version:nn #1#2
131 {
132     \str_if_eq:VnTF \math@version { #1 }
133     { \__siunitx_print_math_auxii:n {#2} }
134     {
135         \mode_if_math:TF
136         { \text }
137         { \use:n }
138         {
139             \mathversion {#1}
140             \__siunitx_print_math_auxii:n {#2}
141         }
142     }
143 }
144 \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.

```

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

```

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.

```

162 \cs_new_protected:Npn \__siunitx_print_math_auxiv:n #1
163 {
164   \bool_if:NTF \l__siunitx_print_math_font_bool
165   { \__siunitx_print_math_aux:N \mathsf \mathtt \q_recursion_tail \q_recursion_stop }
166   { \__siunitx_print_math_auxv:n }
167   {#1}
168 }
169 \cs_new_protected:Npn \__siunitx_print_math_auxv:n #1
170 {
171   \bool_lazy_or:nnTF
172   { \int_compare_p:nNn \fam = { -1 } }
173   { \int_compare_p:nNn \fam = \symoperators }
174   { \use:n }
175   { \mathrm }
176   {#1}
177 }
178 \cs_new_protected:Npn \__siunitx_print_math_aux:N #1
179 {
180   \quark_if_recursion_tail_stop_do:Nn #1 { \use:n }
181   \exp_after:wN \exp_after:wN \exp_after:wN \__siunitx_print_math_aux:w
182   \cs:w \cs_to_str:N #1 \c_space_tl \cs_end:
183   \use@mathgroup ? { -2 } \q_stop #1
184 }
185 \cs_new_protected:Npn \__siunitx_print_math_aux:w #1 \use@mathgroup #2#3 #4 \q_stop #5
186 {
187   \int_compare:nNnTF \fam = {#3}
188   { \use_i_delimit_by_q_recursion_stop:nw { \__siunitx_print_math_aux:Nn #5 } }
189   { \__siunitx_print_math_aux:N }
190 }

```

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

```

191 \cs_new_protected:Npx \__siunitx_print_math_aux:Nn #1#2
192 {
193   \group_begin:
194     \tl_set:Nn \exp_not:N \l__siunitx_print_tmp_tl {#2}
195     \__siunitx_print_replace_font:N \exp_not:N \l__siunitx_print_tmp_tl
196     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
197       { \char_generate:nn { '\_ } { 8 } }
198       { \exp_not:N \__siunitx_print_math_sub:n }
199     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
200       { ^ }
201     { \exp_not:N \__siunitx_print_math_super:n }
202     #1 { \exp_not:N \tl_use:N \exp_not:N \l__siunitx_print_tmp_tl }
203   \group_end:
204 }
205 \cs_generate_variant:Nn \__siunitx_print_math_aux:Nn { c }
206 \cs_new_protected:Npx \__siunitx_print_math_sub:n #1
207 {
208   \char_generate:nn { '\_ } { 8 }
209   { \exp_not:N \__siunitx_print_math_script:n {#1} }
210 }
211 \cs_new_protected:Npn \__siunitx_print_math_super:n #1
212 { ^ { \__siunitx_print_math_script:n {#1} } }
213 \cs_new_protected:Npn \__siunitx_print_math_script:n #1
214 {
215   \group_begin:
216     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
217     \__siunitx_print_replace_font:N \l__siunitx_print_tmp_tl
218     \tl_use:N \l__siunitx_print_tmp_tl
219   \group_end:
220 }

```

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

```

221 \AtBeginDocument
222 {
223   \ifpackageloaded { tex4ht }
224   {
225     \cs_set_protected:Npn \__siunitx_print_math_auxii:n #1
226     {
227       \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
228       \exp_args:NNnx \tl_replace_all:Nnn \l__siunitx_print_tmp_tl
229         { ^ } { \token_to_str:N ^ }
230       \ensuremath { \exp_args:NV \__siunitx_print_math_auxiii:n \l__siunitx_print_tmp_tl }
231     }
232   }
233 }
234 }

```

(End definition for `\siunitx_print_math:n` and others. This function is documented on page 91.)

\siunitx_print_text:n

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

```

\__siunitx_print_text_replace:n
\__siunitx_print_text_replace:N
\__siunitx_print_text_replace:NNn
\__siunitx_print_text_sub:n
\__siunitx_print_text_super:n
\__siunitx_print_text_scripts:NNn
\__siunitx_print_text_scripts:
\__siunitx_print_text_scripts_one:NNn
\__siunitx_print_text_scripts_two:NNn
\__siunitx_print_text_scripts_two:nn
\__siunitx_print_text_scripts_two:n
\__siunitx_print_text_fraction:NNn

```

```

238 {
239   \bool_if:NT \l__siunitx_print_text_family_bool
240     { \fontfamily { \familydefault } }
241   \bool_if:NT \l__siunitx_print_text_series_bool
242     { \fontseries { \seriesdefault } }
243   \bool_if:NT \l__siunitx_print_text_shape_bool
244     { \fontshape { \shapedefault } }
245   \bool_lazy_any:nT
246     {
247       { \l__siunitx_print_text_family_bool }
248       { \l__siunitx_print_text_series_bool }
249       { \l__siunitx_print_text_shape_bool }
250     }
251     { \selectfont }
252   \tl_use:N \l__siunitx_print_text_font_tl
253   \exp_args:NnV \tl_if_head_eq_meaning:nNTF {#1} \l_siunitx_unit_fraction_tl
254     { \__siunitx_print_text_fraction:Nnn #1 }
255     { \__siunitx_print_text_replace:n {#1} }
256 }
257 }

```

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

```

258 \cs_new_protected:Npn \__siunitx_print_text_replace:n #1
259 {
260   \group_begin:
261     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
262     \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
263     \tl_use:N \l__siunitx_print_tmp_tl
264   \group_end:
265 }
266 \cs_new_protected:Npx \__siunitx_print_text_replace:N #1
267 {
268   \__siunitx_print_replace_font:N #1
269   \exp_not:N \__siunitx_print_text_replace:NNn #1
270     \exp_not:N \mathord { }
271     \exp_not:N \pm
272     { \exp_not:N \textpm }
273     \exp_not:N \mp
274     { \exp_not:n { \ensuremath { \mp } } } }
275 -
276   { \exp_not:N \textminus }
277   \exp_not:N \times
278   { \exp_not:N \texttimes }
279   \exp_not:N \cdot
280   { \exp_not:N \textperiodcentered }
281   \char_generate:nn { \_ } { 8 }
282   { \exp_not:N \__siunitx_print_text_sub:n }
283   ~
284   { \exp_not:N \__siunitx_print_text_super:n }
285   \exp_not:N \q_recursion_tail
286   { ? }
287   \exp_not:N \q_recursion_stop
288 }

```



```

289 \cs_new_protected:Npn \__siunitx_print_text_replace:NNn #1#2#3
290 {
291   \quark_if_recursion_tail_stop:N #2
292   \tl_replace_all:Nnn #1 {#2} {#3}
293   \__siunitx_print_text_replace:NNn #1
294 }

```

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

```

295 \sys_if_engine_xetex:T
296 {
297   \AtBeginDocument
298   {
299     \ifpackageloaded { bidi }
300     {
301       \cs_set_protected:Npn \__siunitx_print_text_replace:n #1
302       {
303         \group_begin:
304         \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
305         \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
306         \LRE { \tl_use:N \l__siunitx_print_tmp_tl }
307         \group_end:
308       }
309     }
310   { }
311 }
312 }

```

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.

```

313 \cs_new_protected:Npn \__siunitx_print_text_sub:n #1
314 {
315   \__siunitx_print_text_scripts:NnN
316   \textsubscript {#1} \__siunitx_print_text_super:n
317 }
318 \cs_new_protected:Npn \__siunitx_print_text_super:n #1
319 {
320   \__siunitx_print_text_scripts:NnN
321   \textsuperscript {#1} \__siunitx_print_text_sub:n
322 }
323 \cs_new_protected:Npn \__siunitx_print_text_scripts:NnN #1#2#3
324 {
325   \cs_set_protected:Npn \__siunitx_print_text_scripts:
326   {
327     \if_meaning:w \l_peek_token #3
328     \exp_after:wN \__siunitx_print_text_scripts_two:NnNn
329     \else:
330     \exp_after:wN \__siunitx_print_text_scripts_one:Nn
331     \fi:
332     #1 {#2}
333   }
334   \peek_after:Nw \__siunitx_print_text_scripts:
335 }
336 \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.

```

337 \cs_new_protected:Npn \__siunitx_print_text_scripts_one:Nn #1#2
338 {
339   \group_begin:
340     \tl_set:Nn \l__siunitx_print_tmp_tl {#2}
341     \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
342     \exp_args:NNV \group_end:
343     #1 \l__siunitx_print_tmp_tl
344 }

```

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

```

345 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:NnNn #1#2#3#4
346 {
347   \cs_if_eq:NNTF #1 \textsubscript
348     { \__siunitx_print_text_scripts_two:nn {#4} {#2} }
349     { \__siunitx_print_text_scripts_two:nn {#2} {#4} }
350 }
351 \cs_new_protected:Npx \__siunitx_print_text_scripts_two:nn #1#2
352 {
353   \group_begin:
354     \exp_not:N \m@th
355     \exp_not:N \ensuremath
356     {
357       ~ { \exp_not:N \__siunitx_print_text_scripts_two:n {#1} }
358       \char_generate:nn { '\_ } { 8 }
359       { \exp_not:N \__siunitx_print_text_scripts_two:n {#2} }
360     }
361   \group_end:
362 }
363 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:n #1
364 {
365   \mbox
366   {
367     \fontsize \sf@size \z@ \selectfont
368     \__siunitx_print_text_scripts_one:Nn \use:n {#1}
369   }
370 }

```

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

```

371 \cs_new_protected:Npn \__siunitx_print_text_fraction:Nnn #1#2#3
372 {
373   \ensuremath
374   {
375     #1
376     { \mbox { \__siunitx_print_text_replace:n {#2} } }
377     { \mbox { \__siunitx_print_text_replace:n {#3} } }
378   }
379 }

```

(End definition for `\siunitx_print_text:n` and others. This function is documented on page 91.)

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.

```
380 \keys_set:nn { siunitx }
381 {
382   color           = ,
383   mode            = math ,
384   number-color    = ,
385   number-mode     = math ,
386   propagate-math-font = false ,
387   reset-math-version = true ,
388   reset-text-shape = true ,
389   reset-text-series = true ,
390   reset-text-family = true ,
391   text-family-to-math = false ,
392   text-font-command = ,
393   text-series-to-math = false ,
394   unit-color       = ,
395   unit-mode        = math
396 }
```

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

```
397 \keys_set:nn { siunitx / series-version-mapping }
398 {
399   ul = light ,
400   el = light ,
401   l  = light ,
402   sl = light ,
403   m  = normal ,
404   sb = bold ,
405   b  = bold ,
406   eb = bold ,
407   ub = bold
408 }
409 </package>
```

Part VII

siunitx-quantity — Quantities

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

`\siunitx_quantity:nn`

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

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

`\siunitx_quantity_print:nn`

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

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

`allow-quantity-breaks`

`allow-quantity-breaks = true|false`

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

`prefix-mode`

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

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

`quantity-product`

`quantity-product = <tokens>`

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

`separate-uncertainty-units`

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

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

1 siunitx-quantity implementation

Start the DocStrip guards.

```
1 <*package>
```

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

```
2 <@@=siunitx_quantity>
```

1.1 Initial set-up

Scratch space.

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

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

1.2 Main formatting routine

Purely internal for the present.

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

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

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

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

```

\l_siunitx_quantity_number_tl
\l__siunitx_quantity_unit_tl

```

```

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

```

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

\siunitx_quantity:nn

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```
130 }
```

(End definition for `\siunitx_quantity:nn` and others. This function is documented on page 104.)

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

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

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

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

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

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

```
139 \cs_new_protected:Npn \siunitx_quantity_print:nn #1#2
140 {
141   \siunitx_print_number:n {#1}
142   \tl_if_blank:nF {#2}
143   {
144     \tl_use:N \l__siunitx_quantity_product_tl
145     \bool_if:NTF \l__siunitx_quantity_break_bool
146       { \penalty \binoppenalty }
147       { \nobreak }
148     \siunitx_print_unit:n {#2}
149   }
150 }
151 \cs_generate_variant:Nn \siunitx_quantity_print:nn { nV , VV , xV }
```

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

1.3 Standard settings for module options

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

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

1.4 Adjustments to units

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

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

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



```

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

```

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

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

```

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

```

(End definition for `\degree`. This function is documented on page [141](#).)

```

183 \endpackage

```

Part VIII

siunitx-symbol – Symbol-related settings

1 siunitx-symbol implementation

Start the DocStrip guards.

```
1 <*package>
```

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

```
2 <@@=siunitx-symbol>
```

`\l__siunitx_symbol_tmpa_tl` Scratch space.

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

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

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

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

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

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

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

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

`_siunitx_symbol_if_replace:NnT`

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

```

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

```

(End definition for `_siunitx_symbol_if_replace:NnT`.)

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

1.1 Bookmark definitions

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

```

129 \AtBeginDocument
130 {
131     \@ifpackageloaded { hyperref }
132     {
133         \exp_args:Nx \pdfstringdefDisableCommands
134         {
135             \cs_set_eq:NN \siunitx_print_text:n \exp_not:N \use:n
136             \siunitx_declare_unit:Nn \exp_not:N \ohm
137             { \_siunitx_symbol_non_latin:n { "03A9 } }
138         }
139     }
140     { }
141 }
142 \end{package}

```

Part IX

siunitx-table – Formatting numbers in tables

1 Numbers in tables

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

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

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

1.1 Key-value options

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

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

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

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

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

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

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

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

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

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

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

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

2 siunitx-table implementation

Start the DocStrip guards.

```
1 <*package>
```

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

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

Scratch space.

```
\l__siunitx_table_tmp_box
```

```
\l__siunitx_table_tmp_dim
```

```
\l__siunitx_table_tmp_tl
```

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

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

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

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

2.1 Interface functions

```
\l__siunitx_table_text_bool
```

Used to track that a cell is purely text.

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

(End definition for \l__siunitx_table_text_bool.)

```
\siunitx_cell_begin:w
```

```
\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 114.)

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
\__siunitx_table_collect_end:n entire content is not a brace group: there is more likely to be problematic content in the
\__siunitx_table_collect_end_aux:n case of a header.
\__siunitx_table_collect_end:w 110 \cs_new_protected:Npn \__siunitx_table_collect_end:
111 {
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 }
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   { \protected@edef }
134   \l__siunitx_table_collect_tl {#1}
135 }
136 \cs_new:Npn \__siunitx_table_collect_end_aux:n #1
137 { \exp_after:wN \__siunitx_table_collect_end:w #1 \q_stop }
138 \cs_new:Npn \__siunitx_table_collect_end:w #1 \q_stop
139 { \exp_not:n {#1} }

(End definition for \__siunitx_table_collect_end: and others.)

\__siunitx_table_split:nNNN Splitting into parts uses the fact that numbers cannot contain groups and that we can
\__siunitx_table_split_loop:NNN track where we are up to based on the content of the token lists.
\__siunitx_table_split_group:NNN 140 \cs_new_protected:Npn \__siunitx_table_split:nNNN #1#2#3#4
\__siunitx_table_split_token:NNN 141 {
142   \tl_clear:N #2

```

```

143 \tl_clear:N #3
144 \tl_clear:N #4
145 \__siunitx_table_split_loop:NNN #2#3#4 #1 \q_recursion_tail \q_recursion_stop
146 \__siunitx_table_split_tidy:N #2
147 \__siunitx_table_split_tidy:N #4
148 }
149 \cs_new_protected:Npn \__siunitx_table_split_loop:NNN #1#2#3
150 {
151 \peek_catcode_ignore_spaces:NTF \c_group_begin_token
152 { \__siunitx_table_split_group:NNNn #1#2#3 }
153 { \__siunitx_table_split_token:NNNN #1#2#3 }
154 }
155 \cs_new_protected:Npn \__siunitx_table_split_group:NNNn #1#2#3#4
156 {
157 \tl_if_empty:NTF #2
158 { \tl_put_right:Nn #1 { {#4} } }
159 { \tl_put_right:Nn #3 { {#4} } }
160 \__siunitx_table_split_loop:NNN #1#2#3
161 }
162 \cs_new_protected:Npn \__siunitx_table_split_token:NNNN #1#2#3#4
163 {
164 \quark_if_recursion_tail_stop:N #4
165 \tl_if_empty:NTF \l__siunitx_table_after_tl
166 {
167 \siunitx_if_number_token:NTF #4
168 { \tl_put_right:Nn #2 {#4} }
169 {
170 \tl_if_empty:NTF #2
171 { \tl_put_right:Nn #1 {#4} }
172 { \tl_put_right:Nn #3 {#4} }
173 }
174 }
175 { \tl_put_right:Nn #3 {#4} }
176 \__siunitx_table_split_loop:NNN #1#2#3
177 }

```

(End definition for __siunitx_table_split:nNNN and others.)

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

```

178 \cs_new_protected:Npn \__siunitx_table_split_tidy:N #1
179 {
180 \tl_if_empty:NF #1
181 { \__siunitx_table_split_tidy:NV #1 #1 }
182 }
183 \cs_new_protected:Npn \__siunitx_table_split_tidy:Nn #1#2
184 {
185 \str_if_eq:onT { \exp_after:wN { \use:n #2 } } {#2}
186 { \tl_set:Nn #1 { \use:n #2 } }
187 }
188 \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 $\text{\LaTeX} 2_{\epsilon}$ definition, cell material is centred by a construction of the (primitive) form

```
\hfil
#
\hfil
```

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

```
\hskip Opt plus 0.5fill
\kern Opt
#
\hskip Opt plus 0.5fill
```

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

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

```
189 \cs_new_protected:Npn \__siunitx_table_skip:n #1
190 {
191   \skip_horizontal:n {#1}
192   \tex_kern:D \c_zero_skip
193 }
```

(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
194 \keys_define:nn { siunitx }
195 {
196   table-column-width .dim_set:N =
197     \l_siunitx_table_column_width_dim ,
198   table-fixed-width .bool_set:N =
199     \l_siunitx_table_fixed_width_bool
200 }
```

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

`_siunitx_table_align_center:n`
`_siunitx_table_align_left:n`
`_siunitx_table_align_right:n`
`_siunitx_table_align_auxi: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.

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

```

212     {
213       \__siunitx_table_skip:n {#2}
214       #1
215       \__siunitx_table_skip:n { Opt~plus~1fill - #2 }
216     }
217   }
218 \AtBeginDocument
219 {
220   \ifpackageloaded { colortbl }
221   {
222     \cs_new_eq:NN
223       \__siunitx_table_align_auxii:nn
224       \__siunitx_table_align_auxi:nn
225     \cs_set_protected:Npn \__siunitx_table_align_auxi:nn #1#2
226     {
227       \__siunitx_table_skip:n{ Opt~plus~-0.5fill }
228       \__siunitx_table_align_auxii:nn {#1} {#2}
229       \__siunitx_table_skip:n { Opt~plus~-0.5fill }
230     }
231   }
232   { }
233 }

```

(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.

```

234 \keys_define:nn { siunitx }
235 {
236   table-text-alignment .choices:nn =
237     { center , left , right }
238     { \tl_set:Nn \l__siunitx_table_align_text_tl {#1} } ,
239 }
240 \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
241 \cs_new_protected:Npn \__siunitx_table_print_text:n #1
242 {
243   \bool_set_true:N \l__siunitx_table_text_bool
244   \use:c { __siunitx_table_align_ \l__siunitx_table_align_text_tl :n } {#1}
245 }
246 \cs_generate_variant:Nn \__siunitx_table_print_text:n { V }

```

(End definition for __siunitx_table_print_text:n.)

2.6 Number alignment: core ideas

<code>\l_siunitx_table_integer_box</code>	Boxes for the content before and after the decimal marker.
<code>\l_siunitx_table_decimal_box</code>	<pre> 247 \box_new:N \l__siunitx_table_integer_box 248 \box_new:N \l__siunitx_table_decimal_box (End definition for \l__siunitx_table_integer_box and \l__siunitx_table_decimal_box.) </pre>
<code>__siunitx_table_fil:</code>	Primitives renamed.
<code>__siunitx_table_fill:</code>	<pre> 249 \cs_new_eq:NN __siunitx_table_fil: \tex_hfil:D 250 \cs_new_eq:NN __siunitx_table_fill: \tex_hfill:D (End definition for __siunitx_table_fil: and __siunitx_table_fill:.) </pre>
<code>_siunitx_table_cleanup_decimal:w</code>	To remove the excess marker tokens in a decimal part.
	<pre> 251 \cs_new:Npn _siunitx_table_cleanup_decimal:w 252 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil 253 { #1#2#3#4#5#6#7 } (End definition for _siunitx_table_cleanup_decimal:w.) </pre>
<code>_siunitx_table_color_check:N</code>	Handle the fact that splitting a number can leave a negative color dangling.
<code>_siunitx_table_color_check:w</code>	
<code>_siunitx_table_color_check:Nnw</code>	<pre> 254 \cs_new_protected:Npn _siunitx_table_color_check:N #1 255 { \exp_after:wN _siunitx_table_color_check:w #1 \q_stop } 256 \cs_new_protected:Npn _siunitx_table_color_check:w #1 \q_nil #2 \q_stop 257 { 258 \tl_if_head_eq_meaning:nNT {#1} \color 259 { _siunitx_table_color_check:Nnw #1 \q_stop } 260 } 261 \cs_new_protected:Npn _siunitx_table_color_check:Nnw #1#2#3 \q_stop 262 { \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.) </pre>
<code>_siunitx_table_center_marker:</code>	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.
	<pre> 263 \cs_new_protected:Npn _siunitx_table_center_marker: 264 { 265 \hbox_set:Nn \l__siunitx_table_tmp_box 266 { \ensuremath { \mathord { \l_siunitx_number_output_decimal_tl } } } 267 \dim_compare:nNnTF 268 { \box_wd:N \l__siunitx_table_integer_box } 269 > 270 { 271 \box_wd:N \l__siunitx_table_decimal_box 272 - \box_wd:N \l__siunitx_table_tmp_box 273 } 274 { 275 \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box 276 { 277 \box_wd:N \l__siunitx_table_integer_box 278 + \box_wd:N \l__siunitx_table_tmp_box 279 } </pre>

```

280     {
281       \hbox_unpack:N \l__siunitx_table_decimal_box
282       \__siunitx_table_fil:
283     }
284   }
285   {
286     \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
287     {
288       \box_wd:N \l__siunitx_table_decimal_box
289       - \box_wd:N \l__siunitx_table_tmp_box
290     }
291     {
292       \__siunitx_table_fil:
293       \hbox_unpack:N \l__siunitx_table_integer_box
294     }
295   }
296 }

```

(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.

```

297 \keys_define:nn { siunitx }
298 {
299   table-alignment .meta:n =
300     { table-number-alignment = #1 , table-text-alignment = #1 },
301   table-alignment-mode .choices:nn =
302     { none , format , marker }
303     { \tl_set_eq:NN \l__siunitx_table_align_mode_tl \l_keys_choice_tl } ,
304   table-auto-round .bool_set:N =
305     \l__siunitx_table_auto_round_bool ,
306   table-format .code:n =
307     {
308       \__siunitx_table_split:nNNN {#1}
309       \l__siunitx_table_before_model_tl
310       \l__siunitx_table_model_tl
311       \l__siunitx_table_after_model_tl
312       \exp_args:NV \__siunitx_table_generate_model:n \l__siunitx_table_model_tl
313       \tl_set:Nn \l__siunitx_table_align_mode_tl { format }
314     } ,
315   table-number-alignment .choices:nn =
316     { center , left , right }
317     { \tl_set_eq:NN \l__siunitx_table_align_number_tl \l_keys_choice_tl }
318   }
319   \tl_new:N \l__siunitx_table_align_mode_tl
320   \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.

```

321 \tl_new:N \l__siunitx_table_format_tl
322 \tl_new:N \l__siunitx_table_before_model_tl
323 \tl_new:N \l__siunitx_table_model_tl
324 \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.

```

325 \cs_new_protected:Npn \__siunitx_table_generate_model:n #1
326 {
327   \group_begin:
328     \bool_set_true:N \l__siunitx_number_parse_bool
329     \keys_set:nn { siunitx } { retain-explicit-plus = true }
330     \siunitx_number_parse:nN {#1} \l__siunitx_table_format_tl
331     \exp_args:NNNV \group_end:
332     \tl_set:Nn \l__siunitx_table_format_tl \l__siunitx_table_format_tl
333     \tl_if_empty:NF \l__siunitx_table_format_tl
334     {
335       \exp_after:wN \__siunitx_table_generate_model:nnnnnnn
336       \l__siunitx_table_format_tl
337     }
338   }
339 \cs_new_protected:Npn \__siunitx_table_generate_model:nnnnnnn #1#2#3#4#5#6#7
340 {
341   \tl_set:Nx \l__siunitx_table_model_tl
342   {
343     \exp_not:n { {#1} {#2} }
344     { \prg_replicate:nn {#3} { 8 } }
345     { \prg_replicate:nn { 0 #4 } { 8 } }
346     {
347       \tl_if_blank:NF {#5}
348       {
349         \use:c { __siunitx_table_generate_model_ \tl_head:n {#5} :nnw }
350         {#4} #5
351       }
352     }
353     \exp_not:n { {#6} }
354     {
355       \int_compare:nNnTF {#7} = 0
356       { 0 }
357       { \prg_replicate:nn {#7} { 8 } }
358     }
359   }
360 }
361 \cs_new:Npn \__siunitx_table_generate_model_S:nnw #1#2#3
362 {
363   { S }
364   {
365     \exp_args:Nff \__siunitx_table_generate_model_S:nnn
366     { \tl_count:n {#1} } { \tl_count:n {#3} }
367     {#3}
368   }
369 }
370 \cs_new:Npn \__siunitx_table_generate_model_S:nnn #1#2#3
371 {

```

```

372 \prg_replicate:nn
373 {
374   \int_compare:nNnTF {#2} > {#1}
375   {
376     \str_range:nnn {#3} { 1 } {#1}
377     +
378     \str_range:nnn {#3} { 1 + #1 } {#2}
379   }
380   {#3}
381 }
382 { 8 }
383 }

```

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

2.7 Directly printing without collection

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

<pre> __siunitx_table_direct_begin: __siunitx_table_direct_begin:w __siunitx_table_direct_end: __siunitx_table_direct_marker: __siunitx_table_direct_marker_switch: __siunitx_table_direct_marker_end: __siunitx_table_direct_format: __siunitx_table_direct_format:nnnnnnn __siunitx_table_direct_format:w __siunitx_table_direct_format_switch: __siunitx_table_direct_format_end: __siunitx_table_direct_none: __siunitx_table_direct_none_end: </pre>	<pre> 384 \cs_new_protected:Npn __siunitx_table_direct_begin: 385 { __siunitx_table_direct_begin:w } 386 \cs_new_protected:Npn __siunitx_table_direct_begin:w #1 \ignorespaces 387 { 388 #1 389 \peek_catcode_ignore_spaces:NTF \c_group_begin_token 390 { __siunitx_table_print_text:n } 391 { 392 \m@th 393 \use:c { __siunitx_table_direct_ \l__siunitx_table_align_mode_tl : } 394 } 395 } 396 \cs_new_protected:Npn __siunitx_table_direct_end: 397 { \use:c { __siunitx_table_direct_ \l__siunitx_table_align_mode_tl _end: } } </pre>
--	---

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.

```

398 \cs_new_protected:Npn \__siunitx_table_direct_marker:
399 {
400   \hbox_set:Nn \l__siunitx_table_tmp_box
401   { \ensuremath { \mathord { \l__siunitx_number_output_decimal_tl } } }
402   \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
403   { \box_wd:N \l__siunitx_table_tmp_box }
404   { \__siunitx_table_fil: }
405   \hbox_set:Nw \l__siunitx_table_integer_box
406   \c_math_toggle_token
407   \tl_map_inline:Nn \l__siunitx_number_input_decimal_tl
408   {
409     \char_set_active_eq:NN ##1 \__siunitx_table_direct_marker_switch:
410     \char_set_mathcode:nn { '##1 } { "8000 }

```

```

411     }
412   }
413   \cs_new_protected:Npn \__siunitx_table_direct_marker_switch:
414   {
415     \c_math_toggle_token
416     \hbox_set_end:
417     \hbox_set:Nw \l__siunitx_table_decimal_box
418     \c_math_toggle_token
419     \l_siunitx_number_output_decimal_tl
420   }
421   \cs_new_protected:Npn \__siunitx_table_direct_marker_end:
422   {
423     \c_math_toggle_token
424     \hbox_set_end:
425     \__siunitx_table_center_marker:
426     \use:c { \__siunitx_table_align \l__siunitx_table_align_text_tl :n }
427     {
428       \box_use_drop:N \l__siunitx_table_integer_box
429       \box_use_drop:N \l__siunitx_table_decimal_box
430     }
431   }

```

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

```

432   \cs_new_protected:Npn \__siunitx_table_direct_format:
433   {
434     \tl_set:Nx \l__siunitx_table_tmp_tl
435     { \siunitx_number_output:NN \l__siunitx_table_model_tl \q_nil }
436     \exp_after:wN \__siunitx_table_direct_format_aux:w
437     \l__siunitx_table_tmp_tl \q_stop
438   }
439   \cs_new_protected:Npn \__siunitx_table_direct_format_aux:w
440   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_stop
441   {
442     \hbox_set:Nn \l__siunitx_table_tmp_box
443     { \ensuremath { \__siunitx_table_cleanup_decimal:w #4 } }
444     \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
445     { \box_wd:N \l__siunitx_table_tmp_box }
446     { \__siunitx_table_fil: }
447     \hbox_set:Nn \l__siunitx_table_tmp_box { \ensuremath { #1#2#3 } }
448     \hbox_set_to_wd:Nnw \l__siunitx_table_integer_box
449     { \box_wd:N \l__siunitx_table_tmp_box }
450     \c_math_toggle_token
451     \tl_map_inline:Nn \l_siunitx_number_input_decimal_tl
452     {
453       \char_set_active_eq:NN ##1 \__siunitx_table_direct_format_switch:
454       \char_set_mathcode:nn { '##1 } { "8000 }
455     }
456     \__siunitx_table_fill:
457   }
458   \cs_new_protected:Npn \__siunitx_table_direct_format_switch:
459   {
460     \c_math_toggle_token
461     \hbox_set_end:

```

```

462 \hbox_set_to_wd:Nnw \l__siunitx_table_decimal_box
463 { \box_wd:N \l__siunitx_table_decimal_box }
464 \c_math_toggle_token
465 \mathord { \l_siunitx_number_output_decimal_tl }
466 }
467 \cs_new_protected:Npn \__siunitx_table_direct_format_end:
468 {
469 \c_math_toggle_token
470 \__siunitx_table_fil:
471 \hbox_set_end:
472 \use:c { __siunitx_table_align \l__siunitx_table_align_number_tl :n }
473 {
474 \box_use_drop:N \l__siunitx_table_integer_box
475 \box_use_drop:N \l__siunitx_table_decimal_box
476 }
477 }

```

No parsing and no alignment is easy.

```

478 \cs_new_protected:Npn \__siunitx_table_direct_none: { \c_math_toggle_token }
479 \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
480 \box_new:N \l__siunitx_table_before_box
481 \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.

```

482 \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.

```

483 \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
484 \keys_define:nn { siunitx }
485 {
486 table-align-comparator .bool_set:N =
487 \l__siunitx_table_align_comparator_bool ,
488 table-align-exponent .bool_set:N =
489 \l__siunitx_table_align_exponent_bool ,
490 table-align-text-after .bool_set:N =
491 \l__siunitx_table_align_after_bool ,
492 table-align-text-before .bool_set:N =
493 \l__siunitx_table_align_before_bool ,
494 table-align-uncertainty .bool_set:N =

```

```

495     \l__siunitx_table_align_uncertainty_bool
496 }

```

(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:nnm
497 \cs_new_protected:Npn \__siunitx_table_print:nnn #1#2#3
498 { \use:c { __siunitx_table_print_ \l__siunitx_table_align_mode_tl :nnn } {#1} {#2} {#3} }
499 \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.

```

500 \cs_new_protected:Npn \__siunitx_table_print_marker:nnn #1#2#3
501 {
502   \hbox_set:Nn \l__siunitx_table_before_box {#1}
503   \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { Opt }
504   {
505     \box_clear:N \l__siunitx_table_before_box
506     #1
507   }
508   \hbox_set:Nn \l__siunitx_table_after_box {#3}
509   \dim_compare:nNnTF
510     { \box_wd:N \l__siunitx_table_after_box }
511     > { \box_wd:N \l__siunitx_table_before_box }
512     {
513       \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
514         { \box_wd:N \l__siunitx_table_after_box }
515       {
516         \__siunitx_table_fil:
517         \hbox_unpack:N \l__siunitx_table_before_box
518       }
519     }
520   {
521     \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
522       { \box_wd:N \l__siunitx_table_before_box }
523     {
524       \hbox_unpack:N \l__siunitx_table_after_box
525       \__siunitx_table_fil:
526     }
527   }
528   \box_use_drop:N \l__siunitx_table_before_box
529   \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
530   \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
531   \tl_set:Nx \l__siunitx_table_tmp_tl
532     { \siunitx_number_output:NN \l__siunitx_table_tmp_tl \q_nil }
533   \__siunitx_table_color_check:N \l__siunitx_table_tmp_tl
534   \exp_after:wN \__siunitx_table_print_marker:w
535     \l__siunitx_table_tmp_tl \q_stop
536   \box_use_drop:N \l__siunitx_table_after_box

```

```

537 }
538 \cs_new_protected:Npn \__siunitx_table_print_marker:w
539 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_stop
540 {
541   \hbox_set:Nn \l__siunitx_table_integer_box
542     { \siunitx_print_number:n { #1#2#3 } }
543   \hbox_set:Nn \l__siunitx_table_decimal_box
544     {
545       \siunitx_print_number:x
546       { \__siunitx_table_print_marker_aux:w #4 }
547     }
548   \__siunitx_table_center_marker:
549   \use:c { __siunitx_table_align_ \l__siunitx_table_align_text_tl :n }
550   {
551     \box_use_drop:N \l__siunitx_table_integer_box
552     \box_use_drop:N \l__siunitx_table_decimal_box
553   }
554 }
555 \cs_new:Npn \__siunitx_table_print_marker_aux:w
556 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
557 {
558   \exp_not:n {#1#2#3#4#5}
559   \tl_if_blank:nT {#1#2#3#4#5} { { } }
560   \exp_not:n {#6#7}
561 }

```

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.

```

562 \cs_new_protected:Npn \__siunitx_table_print_format:nnn #1#2#3
563 {
564   \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_before_model_tl }
565   \hbox_set:Nn \l__siunitx_table_before_box {#1}
566   \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { 0pt }
567   {
568     \box_clear:N \l__siunitx_table_before_box
569     #1
570   }
571   \dim_set:Nn \l__siunitx_table_before_dim { \box_wd:N \l__siunitx_table_tmp_box }
572   \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
573   \group_begin:
574     \bool_if:NT \l__siunitx_table_auto_round_bool
575     {
576       \exp_args:Nx \keys_set:nn { siunitx }
577       {
578         round-mode      = places ,
579         round-pad       = true  ,
580         round-precision =
581           \exp_after:wN \__siunitx_table_print_format:nnnnnn
582           \l__siunitx_table_format_tl
583       }

```

```

584     }
585     \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
586     \exp_args:NNNV \group_end:
587     \tl_set:Nn \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
588     \tl_set:Nx \l__siunitx_table_tmp_tl
589     {
590         \siunitx_number_output:NN \l__siunitx_table_model_tl \q_nil
591         \exp_not:N \q_mark
592         \siunitx_number_output:NN \l__siunitx_table_tmp_tl \q_nil
593     }
594     \exp_after:wN \__siunitx_table_print_format_auxi:w
595     \l__siunitx_table_tmp_tl \q_stop
596     \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_after_model_tl }
597     \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
598     { \box_wd:N \l__siunitx_table_tmp_box + \l__siunitx_table_carry_dim }
599     {
600         \bool_if:NT \l__siunitx_table_align_after_bool
601         { \skip_horizontal:n { \l__siunitx_table_carry_dim } }
602         #3
603         \__siunitx_table_fil:
604     }
605     \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
606     {
607         \box_use_drop:N \l__siunitx_table_before_box
608         \box_use_drop:N \l__siunitx_table_integer_box
609         \box_use_drop:N \l__siunitx_table_decimal_box
610         \box_use_drop:N \l__siunitx_table_after_box
611     }
612 }
613 \cs_new:Npn \__siunitx_table_print_format:nnnnnn #1#2#3#4#5#6#7
614 { 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).

```

615 \cs_new_protected:Npn \__siunitx_table_print_format_auxi:w
616 #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
617 {
618     \__siunitx_table_color_check:w #3 \q_nil \q_stop
619     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1}
620     \bool_if:NTF \l__siunitx_table_align_before_bool
621     {
622         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
623         { \box_wd:N \l__siunitx_table_tmp_box }
624         {
625             \__siunitx_table_fil:
626             \tl_if_blank:nF {#3}
627             { \siunitx_print_number:n {#3} }
628         }
629     }
630     {
631         \__siunitx_table_print_format_box:Nn \l__siunitx_table_integer_box {#3}
632         \dim_add:Nn \l__siunitx_table_before_dim
633         {

```

```

634         \box_wd:N \l__siunitx_table_tmp_box
635     - \box_wd:N \l__siunitx_table_integer_box
636     }
637 }
638 \__siunitx_table_print_format_auxii:w #2 \q_mark #4 \q_stop
639 }

```

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.

```

640 \cs_new_protected:Npn \__siunitx_table_print_format_auxii:w
641   #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
642   {
643     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
644     \bool_lazy_and:nnTF
645       { \l__siunitx_table_align_comparator_bool }
646       { \dim_compare_p:nNn { \box_wd:N \l__siunitx_table_integer_box } > { Opt } }
647       {
648         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
649         {
650           \box_wd:N \l__siunitx_table_integer_box
651           + \box_wd:N \l__siunitx_table_tmp_box
652         }
653         {
654           \hbox_unpack:N \l__siunitx_table_integer_box
655           \__siunitx_table_fil:
656           \siunitx_print_number:n {#4#5}
657         }
658       }
659     {
660       \bool_if:NTF \l__siunitx_table_align_before_bool
661       {
662         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
663         {
664           \box_wd:N \l__siunitx_table_integer_box
665           + \box_wd:N \l__siunitx_table_tmp_box
666         }
667         {
668           \__siunitx_table_fil:
669           \hbox_unpack:N \l__siunitx_table_integer_box
670           \siunitx_print_number:n {#4#5}
671         }
672       }
673     {
674       \dim_set:Nn \l__siunitx_table_tmp_dim
675       { \box_wd:N \l__siunitx_table_integer_box }
676       \hbox_set:Nn \l__siunitx_table_integer_box
677       {
678         \hbox_unpack:N \l__siunitx_table_integer_box

```



```

679         \siunitx_print_number:n {#4#5}
680     }
681     \dim_add:Nn \l__siunitx_table_before_dim
682     {
683         + \box_wd:N \l__siunitx_table_tmp_box
684         + \l__siunitx_table_tmp_dim
685         - \box_wd:N \l__siunitx_table_integer_box
686     }
687 }
688 }
689 \hbox_set_to_wd:Nnn \l__siunitx_table_before_box \l__siunitx_table_before_dim
690 {
691     \__siunitx_table_fil:
692     \hbox_unpack:N \l__siunitx_table_before_box
693 }
694 \__siunitx_table_print_format_auxiii:w #3 \q_mark #6 \q_stop
695 }

```

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.

```

696 \cs_new_protected:Npn \__siunitx_table_print_format_auxiii:w
697   #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
698   {
699     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
700     \__siunitx_table_print_format_box:Nn \l__siunitx_table_decimal_box {#4#5}
701     \dim_set:Nn \l__siunitx_table_carry_dim
702     {
703         \box_wd:N \l__siunitx_table_tmp_box
704         - \box_wd:N \l__siunitx_table_decimal_box
705     }
706     \__siunitx_table_print_format_auxiv:w #3 \q_mark #6 \q_stop
707 }

```

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.

```

708 \cs_new_protected:Npn \__siunitx_table_print_format_auxiv:w
709   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
710   {
711     \tl_if_blank:nTF {#1}
712     { \__siunitx_table_print_format_auxv:w }
713     { \__siunitx_table_print_format_auxvi:w }
714     #1#2 \q_mark #3#4 \q_stop
715 }
716 \cs_new_protected:Npn \__siunitx_table_print_format_auxv:w
717   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
718   #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
719   { \__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.

```

720 \cs_new_protected:Npn \__siunitx_table_print_format_auxvi:w
721   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark

```

```

722 #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
723 {
724   \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2#3 }
725   \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
726   \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #5#6#7 }
727   \__siunitx_table_print_format_after:N \l__siunitx_table_align_uncertainty_bool
728   \__siunitx_table_print_format_auxvii:w #4 \q_mark #8 \q_stop
729 }

```

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.

```

730 \cs_new_protected:Npn \__siunitx_table_print_format_auxvii:w
731 #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
732 {
733   \tl_if_blank:nF {#2}
734   {
735     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2 }
736     \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
737     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #3#4 }
738     \__siunitx_table_print_format_after:N \l__siunitx_table_align_exponent_bool
739   }
740 }

```

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

```

741 \cs_new_protected:Npn \__siunitx_table_print_format_box:Nn #1#2
742 {
743   \hbox_set:Nn #1
744   {
745     \tl_if_blank:nF {#2}
746     { \siunitx_print_number:n {#2} }
747   }
748 }

```

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

```

749 \cs_new_protected:Npn \__siunitx_table_print_format_after:N #1
750 {
751   \bool_if:NTF #1
752   {
753     \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
754     {
755       \box_wd:N \l__siunitx_table_decimal_box
756       + \l__siunitx_table_carry_dim
757       + \box_wd:N \l__siunitx_table_tmp_box
758     }
759     {
760       \hbox_unpack:N \l__siunitx_table_decimal_box
761       \__siunitx_table_fil:
762       \hbox_unpack:N \l__siunitx_table_tmp_box
763     }
764     \dim_set:Nn \l__siunitx_table_carry_dim
765     {
766       \l__siunitx_table_tmp_dim
767       - \box_wd:N \l__siunitx_table_tmp_box
768     }
769   }

```

```

769     }
770     {
771         \hbox_set:Nn \l__siunitx_table_decimal_box
772         {
773             \hbox_unpack:N \l__siunitx_table_decimal_box
774             \hbox_unpack:N \l__siunitx_table_tmp_box
775         }
776         \dim_add:Nn \l__siunitx_table_carry_dim
777         {
778             \l__siunitx_table_tmp_dim
779             - \box_wd:N \l__siunitx_table_tmp_box
780         }
781     }
782 }

```

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

```

783 \cs_new_protected:Npn \__siunitx_table_print_none:nnn #1#2#3
784 {
785     \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
786     {
787         #1
788         \siunitx_number_format:nN {#2} \l__siunitx_table_tmp_tl
789         \siunitx_print_number:V \l__siunitx_table_tmp_tl
790         #3
791     }
792 }

```

(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.

```

793 \keys_set:nn { siunitx }
794 {
795     table-align-comparator = true ,
796     table-align-exponent   = true ,
797     table-align-text-after  = true ,
798     table-align-text-before = true ,
799     table-align-uncertainty = true ,
800     table-alignment         = center ,
801     table-auto-round        = false ,
802     table-column-width      = 0pt ,
803     table-fixed-width       = false ,
804     table-format            = 2.2 ,
805     table-number-alignment  = center ,
806     table-text-alignment    = center ,

```

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

```

807     table-alignment-mode    = marker
808 }
809 \end{package}

```

Part X

siunitx-unit – Parsing and formatting units

This submodule is dedicated to formatting physical units. The main function, `\siunitx-unit_format:nN`, takes user input specify physical units and converts it into a formatted token list suitable for typesetting in math mode. While the formatter will deal correctly with “literal” user input, the key strength of the module is providing a method to describe physical units in a “symbolic” manner. The output format of these symbolic units can then be controlled by a number of key–value options made available by the module.

A small number of L^AT_EX 2_ε math mode commands are assumed to be available as part of the formatted output. The `\mathchoice` command (normally the T_EX primitive) is needed when using `per-mode = symbol-or-fraction`. The commands `\frac`, `\mathrm`, `\mbox`, `_` and `\,` are used by the standard module settings. For the display of colored (highlighted) and cancelled units, the commands `\textcolor` and `\cancel` are assumed to be available.

1 Formatting units

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

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

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

For example,

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

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

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

<code>\siunitx_unit_format_extract_prefixes:nnN</code>	<code>\siunitx_unit_format_extract_prefixes:nnN {<units>} <tl var> <fp var></code>
--	--

This function formats the $\langle units \rangle$ in the same way as described for `\siunitx_unit_format:nN`. When the input is given in symbolic form, any decimal unit prefixes will be extracted and the overall power of ten that these represent will be stored in the $\langle fp var \rangle$.

For example,

```
\siunitx_unit_format_extract_prefixes:nnN { \kilo \metre \per \second }
\l_tmpa_tl \l_tmpa_fp
```

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

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

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

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

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

For example,

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

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

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

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

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

For example,

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

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

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

2 Defining symbolic units

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

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

<code>\siunitx_declare_prefix:Nn</code>	<code>\siunitx_declare_prefix:Nn <prefix> {<symbol>}</code>
---	---

Defines a symbolic $\langle prefix \rangle$ (which should be a control sequence such as `\kilo`) to be converted by the parser to the $\langle symbol \rangle$. The latter should consist of literal content (e.g. `k`). In literal mode the $\langle symbol \rangle$ will be typeset directly. In contrast to `\siunitx_declare_prefix:Nnn`, there is no assumption about the mathematical nature of the $\langle prefix \rangle$, i.e. the prefix may represent a power of any base. As a result, no conversion of the $\langle prefix \rangle$ to a numerical power will be possible.

<code>\siunitx_declare_power:NNn</code>	<code>\siunitx_declare_power:NNn <pre-power> <post-power> {<value>}</code>
---	--

Defines *two* symbolic $\langle powers \rangle$ (which should be control sequences such as `\squared`) to be converted by the parser to the $\langle value \rangle$. The latter should be an integer or floating point number in the format defined for `l3fp`. Powers may precede a unit or be give after it: both forms are declared at once, as indicated by the argument naming. In literal mode, the $\langle value \rangle$ will be applied as a superscript to either the next token in the input (for the $\langle pre-power \rangle$) or appended to the previously-typeset material (for the $\langle post-power \rangle$).

<code>\siunitx_declare_qualifier:Nn</code>	<code>\siunitx_declare_qualifier:Nn <qualifier> {<meaning>}</code>
--	--

Defines a symbolic $\langle qualifier \rangle$ (which should be a control sequence such as `\catalyst`) to be converted by the parser to the $\langle meaning \rangle$. The latter should consist of literal content (e.g. `cat`). In literal mode the $\langle meaning \rangle$ will be typeset following a space after the unit to which it applies.

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

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

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

`\l_siunitx_unit_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>	

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

5.1 Key-value options

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

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

<hr/> <hr/> extract-mass-in-kilograms	extract-mass-in-kilograms = true false
	Determines whether prefix extraction treats kilograms as a base unit; when set false , grams are used. The standard setting is true .
<hr/> <hr/> forbid-literal-units	forbid-literal-units = true false
	Switch which determines if literal units are allowed when parsing is active; does not apply when parse-units is false .
<hr/> <hr/> fraction-command	fraction-command = $\langle command \rangle$
	Command used to create fractional output when per-mode is set to fraction . The standard setting is <code>\frac</code> .
<hr/> <hr/> inter-unit-product	inter-unit-product = $\langle separator \rangle$
	Inserted between unit combinations in parsed mode, and used to replace <code>.</code> and <code>~</code> in literal mode. The standard setting is <code>\,</code> .
<hr/> <hr/> parse-units	parse-units = true false
	Determines whether parsing of unit symbols is attempted or literal mode is used directly. The standard setting is true .
<hr/> <hr/> per-mode	per-mode = fraction power power-positive-first repeated-symbol symbol symbol-or-fraction
	Selects how the negative powers (<code>\per</code>) are formatted: a choice from the options 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 .
<hr/> <hr/> per-symbol	per-symbol = $\langle symbol \rangle$
	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> .
<hr/> <hr/> qualifier-mode	qualifier-mode = bracket combine phrase subscript
	Selects how qualifiers are formatted: a choice from the options bracket , combine , phrase and subscript . The option bracket wraps the qualifier in parenthesis, combine joins the qualifier with the unit directly, phrase joins the material using qualifier-phrase as a link, and subscript formats the qualifier as a subscript. The standard setting is subscript .
<hr/> <hr/> qualifier-phrase	qualifier-phrase = $\langle phrase \rangle$
	Defines the $\langle phrase \rangle$ used when qualifier-mode is set to phrase .

sticky-per `sticky-per = true|false`

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

unit-font-command `unit-font-command = <command>`

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 }
```

`\l__siunitx_unit_tmp_fp` Scratch space.

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

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

`\c__siunitx_unit_math_subscript_tl` Useful tokens with awkward category codes.

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

(End definition for `\c__siunitx_unit_math_subscript_tl`.)

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

```
9 \bool_new:N \l__siunitx_unit_parsing_bool
```

(End definition for `\l__siunitx_unit_parsing_bool`.)

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

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

(End definition for `\l__siunitx_unit_test_bool`.)

`__siunitx_unit_if_symbolic:nTF`

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

```

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

```

(End definition for `__siunitx_unit_if_symbolic:nTF`.)

6.2 Defining symbolic unit

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

`\l_siunitx_unit_symbolic_seq`

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

```

21 \seq_new:N \l_siunitx_unit_symbolic_seq

```

(End definition for `\l_siunitx_unit_symbolic_seq`. This variable is documented on page 139.)

`\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 139.)

`__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 138.)

`\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 138.)

`\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 138.)

`\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 138.)

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 139.)

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 142.)

\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 142.)

\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 142.)

\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 142.)

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 138.)

```
\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:nNN
\siunitx_unit_format_combine_exponent:nnN
\siunitx_unit_format_multiply:nnN
\siunitx_unit_format_multiply_extract_prefixes:nnNN
\siunitx_unit_format_multiply_combine_exponent:nnnN
\__siunitx_unit_format:nNN
\__siunitx_unit_format_aux:

```

Formatting parsed units can take place either with the prefixes printed or separated out into a power of ten. This variation is handled using two separate functions: as this submodule does not really deal with numbers, formatting the numeral part here would be tricky and it is better therefore to have a mechanism to return a simple numerical power. At the same time, most uses will 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 136.)

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 loosing 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 { \__siunitx_unit_format_literal_auxvii:nn }
278 {#1} #2 \q_recursion_stop
279 }
280 \cs_new_protected:Npx \__siunitx_unit_format_literal_auxvi:nN #1#2
281 {
282 \exp_not:N \quark_if_recursion_tail_stop_do:Nn #2
283 { \exp_not:N \__siunitx_unit_format_literal_add:n {#1} }
284 \exp_not:N \token_if_eq_meaning:NNTF #2 ^
285 { \exp_not:N \__siunitx_unit_format_literal_super:nn {#1} }
286 {
287 \exp_not:N \token_if_eq_meaning:NNTF
288 #2 \c__siunitx_unit_math_subscript_tl
289 { \exp_not:N \__siunitx_unit_format_literal_sub:nn {#1} }
290 { \exp_not:N \__siunitx_unit_format_literal_auxvii:nN {#1} #2 }
291 }
292 }

```

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

```

293 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxvii:nN #1#2
294 {
295 \str_if_eq:nnTF {#2} { \protect }
296 { \__siunitx_unit_format_literal_auxviii:nN {#1} }
297 { \__siunitx_unit_format_literal_auxvi:nN {#1#2} }
298 }
299 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxviii:nN #1#2
300 { \__siunitx_unit_format_literal_auxv:nw { #1 \protect #2 } }
301 \cs_new_protected:Npn \__siunitx_unit_format_literal_super:nn #1#2
302 {
303 \quark_if_recursion_tail_stop:n {#2}
304 \__siunitx_unit_format_literal_add:n {#1}
305 \tl_put_right:Nn \l__siunitx_unit_formatted_tl { ^ {#2} }
306 \__siunitx_unit_format_literal_auxvi:nN { }
307 }
308 \cs_new_protected:Npx \__siunitx_unit_format_literal_sub:nn #1#2
309 {
310 \exp_not:N \quark_if_recursion_tail_stop:n {#2}
311 \exp_not:N \__siunitx_unit_format_literal_add:n {#1}

```

```

312 \tl_put_right:Nx \exp_not:N \l__siunitx_unit_formatted_tl
313 {
314   \c__siunitx_unit_math_subscript_tl
315   {
316     \exp_not:N \exp_not:V
317     \exp_not:N \l_siunitx_unit_font_tl
318     { \exp_not:N \exp_not:n {#2} }
319   }
320 }
321 \exp_not:N \__siunitx_unit_format_literal_auxvi:nN { }
322 }
323 \cs_new_protected:Npn \__siunitx_unit_format_literal_add:n #1
324 {
325   \tl_put_right:Nx \l__siunitx_unit_formatted_tl
326   {
327     \tl_if_empty:NF \l__siunitx_unit_formatted_tl
328     { \exp_not:V \l__siunitx_unit_separator_tl }
329     \tl_if_empty:nF {#1}
330     { \exp_not:V \l_siunitx_unit_font_tl { \exp_not:n {#1} } }
331   }
332   \tl_clear:N \l__siunitx_unit_separator_tl
333 }
334 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxvii:nn #1#2
335 { \__siunitx_unit_format_literal_auxvi:nN { #1 {#2} } }
336 \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.

```

337 \cs_new_protected:Npn \siunitx_unit_pdfstring_context:
338 {
339   \bool_set_false:N \l__siunitx_unit_parsing_bool
340   \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
341   { \cs_set_eq:Nc ##1 { __siunitx_unit_ \token_to_str:N ##1 :w } }
342 }

```

(End definition for \siunitx_unit_pdfstring_context:. This function is documented on page 139.)

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**.

```

343 \keys_define:nn { siunitx }
344   {
345     sticky-per .bool_set:N = \l_siunitx_unit_sticky_per_bool
346   }

```

(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.

```

347 \prop_new:N \l_siunitx_unit_parsed_prop
348 \bool_new:N \l_siunitx_unit_per_bool
349 \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.

```

350 \cs_new_protected:Npn \_siunitx_unit_parse:n #1
351   {
352     \prop_clear:N \l_siunitx_unit_parsed_prop
353     \bool_set_true:N \l_siunitx_unit_parsing_bool
354     \bool_set_false:N \l_siunitx_unit_per_bool
355     \bool_set_false:N \l_siunitx_unit_test_bool
356     \int_zero:N \l_siunitx_unit_position_int
357     \siunitx_unit_options_apply:n {#1}
358     #1
359     \int_step_inline:nn \l_siunitx_unit_position_int
360       { \_siunitx_unit_parse_finalise:n {##1} }
361     \_siunitx_unit_parse_finalise:
362   }

```

(End definition for `_siunitx_unit_parse:n`.)

_siunitx_unit_parse_add:nnnn

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

```
363 \cs_new_protected:Npn \_siunitx_unit_parse_add:nnnn #1#2#3#4
364 {
365   \tl_set:Nx \l__siunitx_unit_tmp_tl { #1 - #2 }
366   \prop_if_in:NVTF \l__siunitx_unit_parsed_prop
367     \l__siunitx_unit_tmp_tl
368   {
369     \msg_error:nnxx { siunitx } { unit / duplicate-part }
370     { \exp_not:n {#1} } { \token_to_str:N #3 }
371   }
372   {
373     \prop_put:NVn \l__siunitx_unit_parsed_prop
374       \l__siunitx_unit_tmp_tl {#4}
375   }
376 }
```

(End definition for _siunitx_unit_parse_add:nnnn.)

_siunitx_unit_parse_prefix:Nn

_siunitx_unit_parse_power:nnN

_siunitx_unit_parse_qualifier:nn

_siunitx_unit_parse_special:n

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

```
377 \cs_new_protected:Npn \_siunitx_unit_parse_prefix:Nn #1#2
378 {
379   \int_set:Nn \l__siunitx_unit_tmp_int { \l__siunitx_unit_position_int + 1 }
380   \_siunitx_unit_parse_add:nnnn { prefix }
381     { \int_use:N \l__siunitx_unit_tmp_int } {#1} {#2}
382 }
383 \cs_new_protected:Npn \_siunitx_unit_parse_power:nnN #1#2#3
384 {
385   \tl_set:Nx \l__siunitx_unit_tmp_tl
386     { unit- \int_use:N \l__siunitx_unit_position_int }
387   \bool_lazy_or:nnTF
388     {#3}
389     {
390       \prop_if_in_p:NV
391         \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
392     }
393     {
394       \_siunitx_unit_parse_add:nnnn { power }
395       {
396         \int_eval:n
397           { \l__siunitx_unit_position_int \bool_if:NT #3 { + 1 } }
398       }
399       {#1} {#2}
400     }
401   {
402     \msg_error:nnxx { siunitx }
403       { unit / part-before-unit } { power } { \token_to_str:N #1 }
404   }
405 }
```



```

406 \cs_new_protected:Npn \__siunitx_unit_parse_qualifier:nn #1#2
407 {
408   \tl_set:Nx \l__siunitx_unit_tmp_tl
409   { unit- \int_use:N \l__siunitx_unit_position_int }
410   \prop_if_in:NVTF \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
411   {
412     \__siunitx_unit_parse_add:nnnn { qualifier }
413     { \int_use:N \l__siunitx_unit_position_int } {#1} {#2}
414   }
415   {
416     \msg_error:nnnn { siunitx }
417     { unit / part-before-unit } { qualifier } { \token_to_str:N #1 }
418   }
419 }

```

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

```

420 \cs_new_protected:Npn \__siunitx_unit_parse_special:n #1
421 {
422   \__siunitx_unit_parse_add:nnnn { special }
423   { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
424   {#1} {#1}
425 }

```

(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.

```

426 \cs_new_protected:Npn \__siunitx_unit_parse_unit:Nn #1#2
427 {
428   \int_incr:N \l__siunitx_unit_position_int
429   \tl_if_eq:nnT {#1} { \gram }
430   {
431     \__siunitx_unit_parse_add:nnnn { command }
432     { \int_use:N \l__siunitx_unit_position_int }
433     {#1} {#1}
434   }
435   \__siunitx_unit_parse_add:nnnn { unit }
436   { \int_use:N \l__siunitx_unit_position_int }
437   {#1} {#2}
438   \bool_if:NT \l__siunitx_unit_per_bool
439   {
440     \__siunitx_unit_parse_add:nnnn { per }
441     { \int_use:N \l__siunitx_unit_position_int }
442     { \per } { true }
443   }
444 }

```

(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.

```

445 \cs_new_protected:Npn \__siunitx_unit_parse_per:
446 {
447   \bool_if:NTF \l__siunitx_unit_sticky_per_bool
448   {
449     \bool_set_true:N \l__siunitx_unit_per_bool
450     \cs_set_protected:Npn \per
451     { \msg_error:nn { siunitx } { unit / duplicate-sticky-per } }
452   }
453   {
454     \__siunitx_unit_parse_add:nnnn
455     { per } { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
456     { \per } { true }
457   }
458 }

```

(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.

```

459 \cs_new_protected:Npn \__siunitx_unit_parse_finalise:n #1
460 {
461   \tl_set:Nx \l__siunitx_unit_tmp_tl { per- #1 }
462   \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
463   {
464     \prop_remove:NV \l__siunitx_unit_parsed_prop
465     \l__siunitx_unit_tmp_tl
466     \tl_set:Nx \l__siunitx_unit_tmp_tl { power- #1 }
467     \prop_get:NVNTF
468     \l__siunitx_unit_parsed_prop
469     \l__siunitx_unit_tmp_tl
470     \l__siunitx_unit_part_tl
471     {
472       \tl_set:Nx \l__siunitx_unit_part_tl
473       { \fp_eval:n { \l__siunitx_unit_part_tl * -1 } }
474       \fp_compare:nNnTF \l__siunitx_unit_part_tl = 1
475       {
476         \prop_remove:NV \l__siunitx_unit_parsed_prop
477         \l__siunitx_unit_tmp_tl
478       }
479       {
480         \prop_put:NVV \l__siunitx_unit_parsed_prop
481         \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
482       }
483     }
484     {
485       \prop_put:NVN \l__siunitx_unit_parsed_prop
486       \l__siunitx_unit_tmp_tl { -1 }
487     }
488   }
489 }

```

(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.

```

490 \cs_new_protected:Npn \_siunitx_unit_parse_finalise:
491 {
492   \clist_map_inline:nn { per , power , prefix }
493   {
494     \tl_set:Nx \l__siunitx_unit_tmp_tl
495     { ##1 - \int_eval:n { \l__siunitx_unit_position_int + 1 } }
496     \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
497     { \msg_error:nnn { siunitx } { unit / dangling-part } { ##1 } }
498   }
499 }

```

(End definition for _siunitx_unit_parse_finalise:.)

6.9 Formatting parsed units

Set up the options which apply to formatting.

```

\l_siunitx_unit_fraction_tl
\l__siunitx_unit_denominator_bracket_bool
\l__siunitx_unit_forbid_literal_bool
\l__siunitx_unit_parse_bool
\l__siunitx_unit_per_symbol_tl
\l__siunitx_unit_qualifier_mode_tl
\l__siunitx_unit_qualifier_phrase_tl

500 \keys_define:nn { siunitx }
501 {
502   bracket-unit-denominator .bool_set:N =
503   \l__siunitx_unit_denominator_bracket_bool ,
504   forbid-literal-units .bool_set:N =
505   \l__siunitx_unit_forbid_literal_bool ,
506   fraction-command .tl_set:N =
507   \l_siunitx_unit_fraction_tl ,
508   parse-units .bool_set:N =
509   \l__siunitx_unit_parse_bool ,
510   per-mode .choice: ,
511   per-mode / fraction .code:n =
512   {
513     \bool_set_false:N \l__siunitx_unit_autofrac_bool
514     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
515     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
516     \bool_set_true:N \l__siunitx_unit_two_part_bool
517   } ,
518   per-mode / power .code:n =
519   {
520     \bool_set_false:N \l__siunitx_unit_autofrac_bool
521     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
522     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
523     \bool_set_false:N \l__siunitx_unit_two_part_bool
524   } ,
525   per-mode / power-positive-first .code:n =
526   {
527     \bool_set_false:N \l__siunitx_unit_autofrac_bool
528     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
529     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
530     \bool_set_true:N \l__siunitx_unit_two_part_bool
531   } ,
532   per-mode / repeated-symbol .code:n =
533   {

```

```

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

```

(End definition for \l_siunitx_unit_fraction_tl and others. This variable is documented on page 139.)

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

```
561 \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.

```

562 \tl_new:N \l__siunitx_unit_bracket_open_tl
563 \tl_new:N \l__siunitx_unit_bracket_close_tl
564 \tl_set:Nn \l__siunitx_unit_bracket_open_tl { ( }
565 \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.

```
566 \bool_new:N \l__siunitx_unit_font_bool
```

(End definition for \l__siunitx_unit_font_bool.)

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

```

567 \bool_new:N \l__siunitx_unit_autofrac_bool
568 \bool_new:N \l__siunitx_unit_per_symbol_bool
569 \bool_new:N \l__siunitx_unit_powers_positive_bool
570 \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).

571 \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.

572 \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.

573 \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).

574 \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.

575 \fp_new:N \l__siunitx_unit_prefix_fp

(End definition for \l__siunitx_unit_prefix_fp.)

\l__siunitx_unit_multiple_fp For multiplying units.

576 \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

577 \tl_new:N \l__siunitx_unit_current_tl

578 \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).

579 \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.

580 \int_new:N \l__siunitx_unit_total_int

(End definition for \l__siunitx_unit_total_int.)

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

```

581 \cs_new_protected:Npn \_siunitx_unit_format_parsed:
582 {
583   \int_set_eq:NN \l__siunitx_unit_total_int \l__siunitx_unit_position_int
584   \tl_clear:N \l__siunitx_unit_denominator_tl
585   \tl_clear:N \l__siunitx_unit_formatted_tl
586   \fp_zero:N \l__siunitx_unit_prefix_fp
587   \int_zero:N \l__siunitx_unit_position_int
588   \fp_compare:nNnF \l__siunitx_unit_combine_exp_fp = \c_zero_fp
589     { \_siunitx_unit_format_combine_exp: }
590   \fp_compare:nNnF \l__siunitx_unit_multiple_fp = \c_one_fp
591     { \_siunitx_unit_format_multiply: }
592   \bool_lazy_and:nnT
593     { \l__siunitx_unit_prefix_exp_bool }
594     { \l__siunitx_unit_mass_kilogram_bool }
595     { \_siunitx_unit_format_mass_to_kilogram: }
596   \int_do_while:nNnn
597     \l__siunitx_unit_position_int < \l__siunitx_unit_total_int
598     {
599       \bool_set_false:N \l__siunitx_unit_bracket_bool
600       \tl_clear:N \l__siunitx_unit_current_tl
601       \bool_set_false:N \l__siunitx_unit_font_bool
602       \bool_set_true:N \l__siunitx_unit_numerator_bool
603       \int_incr:N \l__siunitx_unit_position_int
604       \clist_map_inline:nn { prefix , unit , qualifier , power , special }
605         { \_siunitx_unit_format_parsed_aux:n {##1} }
606       \_siunitx_unit_format_output:
607     }
608   \_siunitx_unit_format_finalise:
609 }
610 \cs_new_protected:Npn \_siunitx_unit_format_parsed_aux:n #1
611 {
612   \tl_set:Nx \l__siunitx_unit_tmp_tl
613     { #1 - \int_use:N \l__siunitx_unit_position_int }
614   \prop_get:NVNT \l__siunitx_unit_parsed_prop
615     \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
616   { \use:c { \_siunitx_unit_format_ #1 : } }
617 }

```

(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.

```

618 \cs_new_protected:Npn \_siunitx_unit_format_combine_exp:
619 {
620   \prop_get:NnNF \l__siunitx_unit_parsed_prop { power-1 } \l__siunitx_unit_tmp_tl
621     { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
622   \fp_set:Nn \l__siunitx_unit_tmp_fp
623     { \l__siunitx_unit_combine_exp_fp / \l__siunitx_unit_tmp_tl }
624   \prop_get:NnNTF \l__siunitx_unit_parsed_prop { prefix-1 } \l__siunitx_unit_tmp_tl
625     {
626       \prop_get:NVNF \l__siunitx_unit_prefixes_forward_prop
627         \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl

```

```

628     {
629         \prop_get:NnN \l__siunitx_unit_parsed_prop { prefix-1 } \l__siunitx_unit_tmp_tl
630         \msg_error:nnx { siunitx } { unit / non-numeric-exponent }
631         { \l__siunitx_unit_tmp_tl }
632         \tl_set:Nn \l__siunitx_unit_tmp_tl { 0 }
633     }
634 }
635 { \tl_set:Nn \l__siunitx_unit_tmp_tl { 0 } }
636 \tl_set:Nx \l__siunitx_unit_tmp_tl
637 { \fp_eval:n { \l__siunitx_unit_tmp_fp + \l__siunitx_unit_tmp_tl } }
638 \fp_compare:nNnTF \l__siunitx_unit_tmp_tl = \c_zero_fp
639 { \prop_remove:Nn \l__siunitx_unit_parsed_prop { prefix-1 } }
640 {
641     \prop_get:NVNTF \l__siunitx_unit_prefixes_reverse_prop
642     \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
643     { \prop_put:NnV \l__siunitx_unit_parsed_prop { prefix-1 } \l__siunitx_unit_tmp_tl }
644     {
645         \msg_error:nnx { siunitx } { unit / non-convertible-exponent }
646         { \l__siunitx_unit_tmp_tl }
647     }
648 }
649 }

```

(End definition for _siunitx_unit_format_combine_exp:.)

_siunitx_unit_format_multiply: A simple mapping.

```

650 \cs_new_protected:Npn \_siunitx_unit_format_multiply:
651 {
652     \int_step_inline:nn { \prop_count:N \l__siunitx_unit_parsed_prop }
653     {
654         \prop_get:NnNF \l__siunitx_unit_parsed_prop { power- ##1 } \l__siunitx_unit_tmp_tl
655         { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
656         \fp_set:Nn \l__siunitx_unit_tmp_fp
657         { \l__siunitx_unit_tmp_tl * \l__siunitx_unit_multiple_fp }
658         \fp_compare:nNnTF \l__siunitx_unit_tmp_fp = \c_one_fp
659         { \prop_remove:N \l__siunitx_unit_parsed_prop { power- ##1 } }
660         {
661             \prop_put:Nnx \l__siunitx_unit_parsed_prop { power- ##1 }
662             { \fp_use:N \l__siunitx_unit_tmp_fp }
663         }
664     }
665 }

```

(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.

```

666 \cs_new_protected:Npn \_siunitx_unit_format_mass_to_kilogram:
667 {
668     \int_step_inline:nn \l__siunitx_unit_total_int
669     {
670         \prop_if_in:NnT \l__siunitx_unit_parsed_prop { command- ##1 }

```

```

671     {
672         \prop_if_in:NnF \l__siunitx_unit_parsed_prop { prefix- ##1 }
673         {
674             \group_begin:
675             \bool_set_false:N \l__siunitx_unit_parsing_bool
676             \tl_set:Nx \l__siunitx_unit_tmp_tl { \kilo }
677             \exp_args:NNNV \group_end:
678             \tl_set:Nn \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
679             \prop_put:NnV \l__siunitx_unit_parsed_prop { prefix- ##1 }
680             \l__siunitx_unit_tmp_tl
681             \prop_get:NnNF \l__siunitx_unit_parsed_prop { power- ##1 }
682             \l__siunitx_unit_tmp_tl
683             { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
684             \fp_set:Nn \l__siunitx_unit_prefix_fp
685             { \l__siunitx_unit_prefix_fp - 3 * \l__siunitx_unit_tmp_tl }
686         }
687     }
688 }
689 }

```

(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.

```

690 \cs_new:Npn \_siunitx_unit_format_bracket:N #1
691 {
692     \bool_if:NTF \l__siunitx_unit_bracket_bool
693     {
694         \exp_not:V \l__siunitx_unit_bracket_open_tl
695         \exp_not:V #1
696         \exp_not:V \l__siunitx_unit_bracket_close_tl
697     }
698     { \exp_not:V #1 }
699 }

```

(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.

```

700 \cs_new_protected:Npn \_siunitx_unit_format_power:
701 {
702     \_siunitx_unit_format_font:
703     \exp_after:wN \_siunitx_unit_format_power_aux:wTF
704     \l__siunitx_unit_part_tl - \q_stop
705     { \_siunitx_unit_format_power_negative: }
706     { \_siunitx_unit_format_power_positive: }
707 }
708 \cs_new:Npn \_siunitx_unit_format_power_aux:wTF #1 - #2 \q_stop
709 { \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 $\neq 1$).


```

710 \cs_new_protected:Npn \__siunitx_unit_format_power_positive:
711 { \__siunitx_unit_format_power_superscript: }

```

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

```

712 \cs_new_protected:Npn \__siunitx_unit_format_power_negative:
713 {
714   \bool_set_false:N \l__siunitx_unit_numerator_bool
715   \bool_if:NTF \l__siunitx_unit_powers_positive_bool
716   {
717     \tl_set:Nx \l__siunitx_unit_part_tl
718     {
719       \exp_after:wN \__siunitx_unit_format_power_negative_aux:w
720       \l__siunitx_unit_part_tl \q_stop
721     }
722     \str_if_eq:VnF \l__siunitx_unit_part_tl { 1 }
723     { \__siunitx_unit_format_power_superscript: }
724   }
725   { \__siunitx_unit_format_power_superscript: }
726 }
727 \cs_new:Npn \__siunitx_unit_format_power_negative_aux:w - #1 \q_stop
728 { \exp_not:n {#1} }

```

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

```

729 \cs_new_protected:Npn \__siunitx_unit_format_power_superscript:
730 {
731   \exp_after:wN \__siunitx_unit_format_power_superscript:w
732   \l__siunitx_unit_part_tl . . \q_stop
733 }
734 \cs_new_protected:Npn \__siunitx_unit_format_power_superscript:w #1 . #2 . #3 \q_stop
735 {
736   \tl_if_blank:nTF {#2}
737   {
738     \tl_set:Nx \l__siunitx_unit_current_tl
739     {
740       \__siunitx_unit_format_bracket:N \l__siunitx_unit_current_tl
741       ^ { \exp_not:n {#1} }
742     }
743   }
744   {
745     \tl_set:Nx \l__siunitx_unit_tmp_tl
746     {
747       { }
748       \tl_if_head_eq_charcode:nNTF {#1} -
749       { { - } { \exp_not:o { \use_none:n #1 } } }
750       { { } { \exp_not:n {#1} } }
751       {#2}
752       { }
753       { }
754       { 0 }

```

```

755     }
756     \tl_set:Nx \l__siunitx_unit_current_tl
757     {
758         \__siunitx_unit_format_bracket:N \l__siunitx_unit_current_tl
759         ~ { \siunitx_number_output:N \l__siunitx_unit_tmp_tl }
760     }
761 }
762 \bool_set_false:N \l__siunitx_unit_bracket_bool
763 }

```

(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.

```

764 \cs_new_protected:Npn \__siunitx_unit_format_prefix:
765 {
766     \bool_if:NTF \l__siunitx_unit_prefix_exp_bool
767     { \__siunitx_unit_format_prefix_exp: }
768     { \__siunitx_unit_format_prefix_symbol: }
769 }
770 \cs_new_protected:Npn \__siunitx_unit_format_prefix_exp:
771 {
772     \prop_get:NVNTF \l__siunitx_unit_prefixes_forward_prop
773     \l__siunitx_unit_part_tl \l__siunitx_unit_part_tl
774     {
775         \bool_if:NT \l__siunitx_unit_mass_kilogram_bool
776         {
777             \tl_set:Nx \l__siunitx_unit_tmp_tl
778             { command- \int_use:N \l__siunitx_unit_position_int }
779             \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
780             { \__siunitx_unit_format_prefix_gram: }
781         }
782         \tl_set:Nx \l__siunitx_unit_tmp_tl
783         { power- \int_use:N \l__siunitx_unit_position_int }
784         \prop_get:NVNF \l__siunitx_unit_parsed_prop
785         \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
786         { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
787         \fp_add:Nn \l__siunitx_unit_prefix_fp
788         { \l__siunitx_unit_tmp_tl * \l__siunitx_unit_part_tl }
789     }
790     { \__siunitx_unit_format_prefix_symbol: }
791 }

```

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

```

792 \cs_new_protected:Npn \__siunitx_unit_format_prefix_gram:
793 {
794     \tl_set:Nx \l__siunitx_unit_part_tl
795     { \int_eval:n { \l__siunitx_unit_part_tl - 3 } }
796     \group_begin:
797     \bool_set_false:N \l__siunitx_unit_parsing_bool
798     \tl_set:Nx \l__siunitx_unit_current_tl { \kilo }
799     \exp_args:NNNV \group_end:

```

```

800 \tl_set:Nn \l__siunitx_unit_current_tl \l__siunitx_unit_current_tl
801 }
802 \cs_new_protected:Npn \__siunitx_unit_format_prefix_symbol:
803 { \tl_set_eq:NN \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl }

```

(End definition for __siunitx_unit_format_prefix: and others.)

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

```

\__siunitx_unit_format_qualifier:
\__siunitx_unit_format_qualifier_bracket:
\__siunitx_unit_format_qualifier_combine:
\__siunitx_unit_format_qualifier_phrase:
\__siunitx_unit_format_qualifier_subscript:
804 \cs_new_protected:Npn \__siunitx_unit_format_qualifier:
805 {
806   \use:c
807   {
808     __siunitx_unit_format_qualifier_
809     \l__siunitx_unit_qualifier_mode_tl :
810   }
811   \tl_put_right:NV \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
812 }
813 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_bracket:
814 {
815   \__siunitx_unit_format_font:
816   \tl_set:Nx \l__siunitx_unit_part_tl
817   {
818     \exp_not:V \l__siunitx_unit_bracket_open_tl
819     \exp_not:V \l__siunitx_unit_font_tl
820     { \exp_not:V \l__siunitx_unit_part_tl }
821     \exp_not:V \l__siunitx_unit_bracket_close_tl
822   }
823 }
824 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_combine: { }
825 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_phrase:
826 {
827   \__siunitx_unit_format_font:
828   \tl_set:Nx \l__siunitx_unit_part_tl
829   {
830     \exp_not:V \l__siunitx_unit_qualifier_phrase_tl
831     \exp_not:V \l__siunitx_unit_font_tl
832     { \exp_not:V \l__siunitx_unit_part_tl }
833   }
834 }
835 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_subscript:
836 {
837   \__siunitx_unit_format_font:
838   \tl_set:Nx \l__siunitx_unit_part_tl
839   {
840     \c__siunitx_unit_math_subscript_tl
841     {
842       \exp_not:V \l__siunitx_unit_font_tl
843       { \exp_not:V \l__siunitx_unit_part_tl }
844     }
845   }
846 }

```

(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.

```

847 \cs_new_protected:Npn \_siunitx_unit_format_special:
848 {
849   \tl_set:Nx \l__siunitx_unit_current_tl
850   {
851     \exp_not:V \l__siunitx_unit_part_tl
852     {
853       \bool_if:NTF \l__siunitx_unit_font_bool
854       { \use:n }
855       { \exp_not:V \l_siunitx_unit_font_tl }
856       { \exp_not:V \l__siunitx_unit_current_tl }
857     }
858   }
859   \bool_set_true:N \l__siunitx_unit_font_bool
860 }

```

(End definition for `_siunitx_unit_format_special:.`)

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

```

861 \cs_new_protected:Npn \_siunitx_unit_format_unit:
862 {
863   \tl_put_right:NV
864   \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
865 }

```

(End definition for `_siunitx_unit_format_unit:.`)

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

```

\_siunitx_unit_format_output_aux:nn
\_siunitx_unit_format_output_aux:nV
\_siunitx_unit_format_output_aux:nv
866 \cs_new_protected:Npn \_siunitx_unit_format_output:
867 {
868   \_siunitx_unit_format_font:
869   \bool_set_false:N \l__siunitx_unit_bracket_bool
870   \use:c
871   {
872     \_siunitx_unit_format_output_
873     \bool_if:NTF \l__siunitx_unit_numerator_bool
874     { aux: }
875     { denominator: }
876   }
877 }
878 \cs_new_protected:Npn \_siunitx_unit_format_output_aux:
879 {
880   \_siunitx_unit_format_output_aux:nV { formatted }
881   \l__siunitx_unit_product_tl
882 }

```

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.

```

883 \cs_new_protected:Npn \__siunitx_unit_format_output_denominator:
884 {
885   \bool_if:NTF \l__siunitx_unit_two_part_bool
886   {
887     \bool_lazy_and:nnT
888     { \l__siunitx_unit_denominator_bracket_bool }
889     { ! \tl_if_empty_p:N \l__siunitx_unit_denominator_tl }
890     { \bool_set_true:N \l__siunitx_unit_bracket_bool }
891     \__siunitx_unit_format_output_aux:nV { denominator }
892     \l__siunitx_unit_product_tl
893   }
894   {
895     \bool_lazy_and:nnT
896     { \l__siunitx_unit_per_symbol_bool }
897     { \tl_if_empty_p:N \l__siunitx_unit_formatted_tl }
898     { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
899     \__siunitx_unit_format_output_aux:nv { formatted }
900     {
901       \l__siunitx_unit_
902       \bool_if:NTF \l__siunitx_unit_per_symbol_bool
903       { per_symbol }
904       { product }
905       _tl
906     }
907   }
908 }
909 \cs_new_protected:Npn \__siunitx_unit_format_output_aux:nn #1#2
910 {
911   \tl_set:cx { \l__siunitx_unit_ #1 _tl }
912   {
913     \exp_not:v { \l__siunitx_unit_ #1 _tl }
914     \tl_if_empty:cF { \l__siunitx_unit_ #1 _tl }
915     { \exp_not:n {#2} }
916     \exp_not:V \l__siunitx_unit_current_tl
917   }
918 }
919 \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.

```

920 \cs_new_protected:Npn \__siunitx_unit_format_font:
921 {
922   \bool_if:NF \l__siunitx_unit_font_bool
923   {
924     \tl_set:Nx \l__siunitx_unit_current_tl
925     {

```

```

926         \exp_not:V \l_siunitx_unit_font_tl
927         { \exp_not:V \l__siunitx_unit_current_tl }
928     }
929     \bool_set_true:N \l__siunitx_unit_font_bool
930 }
931 }

```

(End definition for `__siunitx_unit_format_font:`)

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

```

\__siunitx_unit_format_finalise_autofrac:
\__siunitx_unit_format_finalise_fractional:
\__siunitx_unit_format_finalise_power:
932 \cs_new_protected:Npn \__siunitx_unit_format_finalise:
933 {
934     \tl_if_empty:NF \l__siunitx_unit_denominator_tl
935     {
936         \bool_if:NTF \l__siunitx_unit_powers_positive_bool
937         { \__siunitx_unit_format_finalise_fractional: }
938         { \__siunitx_unit_format_finalise_power: }
939     }
940 }

```

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.

```

941 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fractional:
942 {
943     \tl_if_empty:NT \l__siunitx_unit_formatted_tl
944     { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
945     \bool_if:NTF \l__siunitx_unit_autofrac_bool
946     { \__siunitx_unit_format_finalise_autofrac: }
947     {
948         \bool_if:NTF \l__siunitx_unit_per_symbol_bool
949         { \__siunitx_unit_format_finalise_symbol: }
950         { \__siunitx_unit_format_finalise_fraction: }
951     }
952 }

```

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.

```

953 \cs_new_protected:Npn \__siunitx_unit_format_finalise_autofrac:
954 {
955     \group_begin:
956     \__siunitx_unit_format_finalise_fraction:
957     \exp_args:NNNV \group_end:
958     \tl_set:Nn \l__siunitx_unit_tmp_tl \l__siunitx_unit_formatted_tl
959     \__siunitx_unit_format_finalise_symbol:
960     \tl_set:Nx \l__siunitx_unit_formatted_tl
961     {
962         \mathchoice
963         { \exp_not:V \l__siunitx_unit_tmp_tl }

```

```

964         { \exp_not:V \l__siunitx_unit_formatted_tl }
965         { \exp_not:V \l__siunitx_unit_formatted_tl }
966         { \exp_not:V \l__siunitx_unit_formatted_tl }
967     }
968 }

```

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

```

969 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fraction:
970 {
971     \tl_set:Nx \l__siunitx_unit_formatted_tl
972     {
973         \exp_not:V \l__siunitx_unit_fraction_tl
974         { \exp_not:V \l__siunitx_unit_formatted_tl }
975         { \exp_not:V \l__siunitx_unit_denominator_tl }
976     }
977 }
978 \cs_new_protected:Npn \__siunitx_unit_format_finalise_symbol:
979 {
980     \tl_set:Nx \l__siunitx_unit_formatted_tl
981     {
982         \exp_not:V \l__siunitx_unit_formatted_tl
983         \exp_not:V \l__siunitx_unit_per_symbol_tl
984         \__siunitx_unit_format_bracket:N \l__siunitx_unit_denominator_tl
985     }
986 }

```

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.

```

987 \cs_new_protected:Npn \__siunitx_unit_format_finalise_power:
988 {
989     \tl_if_empty:NTF \l__siunitx_unit_formatted_tl
990     {
991         \tl_set_eq:NN
992         \l__siunitx_unit_formatted_tl
993         \l__siunitx_unit_denominator_tl
994     }
995     {
996         \tl_set:Nx \l__siunitx_unit_formatted_tl
997         {
998             \exp_not:V \l__siunitx_unit_formatted_tl
999             \exp_not:V \l__siunitx_unit_product_tl
1000             \exp_not:V \l__siunitx_unit_denominator_tl
1001         }
1002     }
1003 }

```

(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.

```

1004 \bool_lazy_or:nnTF
1005 { \sys_if_engine luatex_p: }
1006 { \sys_if_engine xetex_p: }

```

```

1007 {
1008   \cs_new:Npn \__siunitx_unit_non_latin:n #1
1009     { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
1010 }
1011 {
1012   \cs_new:Npn \__siunitx_unit_non_latin:n #1
1013     {
1014       \exp_last_unbraced:Nf \__siunitx_unit_non_latin:nnnn
1015       { \char_to_utfviii_bytes:n {#1} }
1016     }
1017   \cs_new:Npn \__siunitx_unit_non_latin:nnnn #1#2#3#4
1018     {
1019       \exp_after:wN \exp_after:wN \exp_after:wN
1020       \exp_not:N \char_generate:nn {#1} { 13 }
1021       \exp_after:wN \exp_after:wN \exp_after:wN
1022       \exp_not:N \char_generate:nn {#2} { 13 }
1023     }
1024 }

```

(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 1025 \siunitx_declare_unit:Nn \kilogram { \kilo \gram }
\mole   1026 \siunitx_declare_unit:Nn \metre   { m }
\kelvin 1027 \siunitx_declare_unit:Nn \meter   { \metre }
\candela 1028 \siunitx_declare_unit:Nn \mole     { mol }
\second  1029 \siunitx_declare_unit:Nn \second   { s }
\ampere   1030 \siunitx_declare_unit:Nn \ampere    { A }
          1031 \siunitx_declare_unit:Nn \kelvin   { K }
          1032 \siunitx_declare_unit:Nn \candela { cd }

```

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

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

```

1033 \siunitx_declare_unit:Nn \gram { g }

```

(End definition for \gram. This function is documented on page 140.)

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

```

\zepto 1034 \siunitx_declare_prefix:Nnn \yocto { -24 } { y }
\atto   1035 \siunitx_declare_prefix:Nnn \zepto { -21 } { z }
\femto  1036 \siunitx_declare_prefix:Nnn \atto { -18 } { a }
\pico   1037 \siunitx_declare_prefix:Nnn \femto { -15 } { f }
\nano   1038 \siunitx_declare_prefix:Nnn \pico { -12 } { p }
\micro  1039 \siunitx_declare_prefix:Nnn \nano { -9 } { n }
\milli  1040 \siunitx_declare_prefix:Nnn \micro { -6 } { \__siunitx_unit_non_latin:n { "03BC } }
\centi  1041 \siunitx_declare_prefix:Nnn \milli { -3 } { m }
\deci   1042 \siunitx_declare_prefix:Nnn \centi { -2 } { c }
          1043 \siunitx_declare_prefix:Nnn \deci { -1 } { d }

```


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

```

\deca Now the large ones.
\deka 1044 \siunitx_declare_prefix:Nnn \deka { 1 } { da }
\hecto 1045 \siunitx_declare_prefix:Nnn \deka { 1 } { da }
\kilo 1046 \siunitx_declare_prefix:Nnn \hecto { 2 } { h }
\mega 1047 \siunitx_declare_prefix:Nnn \kilo { 3 } { k }
\giga 1048 \siunitx_declare_prefix:Nnn \mega { 6 } { M }
\tera 1049 \siunitx_declare_prefix:Nnn \giga { 9 } { G }
\peta 1050 \siunitx_declare_prefix:Nnn \tera { 12 } { T }
\exa 1051 \siunitx_declare_prefix:Nnn \peta { 15 } { P }
\zetta 1052 \siunitx_declare_prefix:Nnn \exa { 18 } { E }
\yotta 1053 \siunitx_declare_prefix:Nnn \zetta { 21 } { Z }
1054 \siunitx_declare_prefix:Nnn \yotta { 24 } { Y }

```

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

```

\becquerel Named derived units: first half of alphabet.
\degreeCelsius 1055 \siunitx_declare_unit:Nn \becquerel { Bq }
\coulomb 1056 \siunitx_declare_unit:Nx \degreeCelsius { \_siunitx_unit_non_latin:n { "00B0 } C }
\farad 1057 \siunitx_declare_unit:Nn \coulomb { C }
\gray 1058 \siunitx_declare_unit:Nn \farad { F }
\hertz 1059 \siunitx_declare_unit:Nn \gray { Gy }
\henry 1060 \siunitx_declare_unit:Nn \hertz { Hz }
\joule 1061 \siunitx_declare_unit:Nn \henry { H }
\katal 1062 \siunitx_declare_unit:Nn \joule { J }
\katal 1063 \siunitx_declare_unit:Nn \katal { kat }
\lumen 1064 \siunitx_declare_unit:Nn \lumen { lm }
\lux 1065 \siunitx_declare_unit:Nn \lux { lx }

```

(End definition for \becquerel and others. These functions are documented on page 141.)

```

\newton Named derived units: second half of alphabet.
\ohm 1066 \siunitx_declare_unit:Nn \newton { N }
\pascal 1067 \siunitx_declare_unit:Nx \ohm { \_siunitx_unit_non_latin:n { "03A9 } }
\radian 1068 \siunitx_declare_unit:Nn \pascal { Pa }
\siemens 1069 \siunitx_declare_unit:Nn \radian { rad }
\sievert 1070 \siunitx_declare_unit:Nn \siemens { S }
\steradian 1071 \siunitx_declare_unit:Nn \sievert { Sv }
\tesla 1072 \siunitx_declare_unit:Nn \steradian { sr }
\volt 1073 \siunitx_declare_unit:Nn \tesla { T }
\watt 1074 \siunitx_declare_unit:Nn \volt { V }
\weber 1075 \siunitx_declare_unit:Nn \watt { W }
1076 \siunitx_declare_unit:Nn \weber { Wb }

```

(End definition for \newton and others. These functions are documented on page 141.)

```

\astronomicalunit Non-SI, but accepted for general use. Once again there are two spellings, here for litre
\bel and with different output in this case.
\dalton 1077 \siunitx_declare_unit:Nn \astronomicalunit { au }
\day 1078 \siunitx_declare_unit:Nn \bel { B }
\decibel 1079 \siunitx_declare_unit:Nn \decibel { \deci \bel }
\electronvolt 1080 \siunitx_declare_unit:Nn \dalton { Da }
\hectare 1081 \siunitx_declare_unit:Nn \day { d }
\hour
\litre
\liter
\minute
\neper
\tonne

```

```

1082 \siunitx_declare_unit:Nn \electronvolt { eV }
1083 \siunitx_declare_unit:Nn \hectare { ha }
1084 \siunitx_declare_unit:Nn \hour { h }
1085 \siunitx_declare_unit:Nn \litre { L }
1086 \siunitx_declare_unit:Nn \liter { \litre }
1087 \siunitx_declare_unit:Nn \minute { min }
1088 \siunitx_declare_unit:Nn \neper { Np }
1089 \siunitx_declare_unit:Nn \tonne { t }

```

(End definition for \astronomicalunit and others. These functions are documented on page 141.)

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

```

\arcsecond 1090 \siunitx_declare_unit:Nx \arcminute { \_siunitx_unit_non_latin:n { "02B9 } }
\degree 1091 \siunitx_declare_unit:Nx \arcsecond { \_siunitx_unit_non_latin:n { "02BA } }
1092 \siunitx_declare_unit:Nx \degree { \_siunitx_unit_non_latin:n { "00B0 } }

```

(End definition for \arcminute, \arcsecond, and \degree. These functions are documented on page 141.)

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

```

1093 \siunitx_declare_unit:Nx \percent { \cs_to_str:N \% }

```

(End definition for \percent. This function is documented on page 141.)

\square Basic powers.

```

\squared 1094 \siunitx_declare_power:NNn \square \squared { 2 }
\cubic 1095 \siunitx_declare_power:NNn \cubic \cubed { 3 }
\cubed

```

(End definition for \square and others. These functions are documented on page 141.)

6.12 Messages

```

1096 \msg_new:nnnn { siunitx } { unit / dangling-part }
1097 { Found~#1~part~with~no~unit. }
1098 {
1099   Each~#1~part~must~be~associated~with~a~unit:~a~#1~part~was~found~
1100   but~no~following~unit~was~given.
1101 }
1102 \msg_new:nnnn { siunitx } { unit / duplicate-part }
1103 { Duplicate~#1~part:~#2. }
1104 {
1105   Each~unit~may~have~only~one~#1:\\
1106   the~additional~#1~part~'#2'~will~be~ignored.
1107 }
1108 \msg_new:nnnn { siunitx } { unit / duplicate-sticky-per }
1109 { Duplicate~\token_to_str:N \per. }
1110 {
1111   When~the~'sticky-per'~option~is~active,~only~one~
1112   \token_to_str:N \per \ may~appear~in~a~unit.
1113 }
1114 \msg_new:nnnn { siunitx } { unit / literal }
1115 { Literal~units~disabled. }
1116 {
1117   You~gave~the~literal~input~'#1'~
1118   but~literal~unit~output~is~disabled.

```

```

1119 }
1120 \msg_new:nnnn { siunitx } { unit / non-convertible-exponent }
1121 { Exponent~'#1'~cannot~be~converted~into~a~symbolic~prefix. }
1122 {
1123   The~exponent~'#1'~does~not~match~with~any~of~the~symbolic~prefixes~
1124   set~up.
1125 }
1126 \msg_new:nnnn { siunitx } { unit / non-numeric-exponent }
1127 { Prefix~'#1'~does~not~have~a~numerical~value. }
1128 {
1129   The~prefix~'#1'~needs~to~be~combined~with~a~number,~but~it~has~no
1130   numerical~value.
1131 }
1132 \msg_new:nnnn { siunitx } { unit / part-before-unit }
1133 { Found~#1~part~before~first~unit:~#2. }
1134 {
1135   The~#1~part~'#2'~must~follow~after~a~unit:~
1136   it~cannot~appear~before~any~units~and~will~therefore~be~ignored.
1137 }

```

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.

```

1138 \keys_set:nn { siunitx }
1139 {
1140   bracket-unit-denominator = true      ,
1141   forbid-literal-units     = false     ,
1142   fraction-command         = \frac     ,
1143   inter-unit-product       = \,        ,
1144   extract-mass-in-kilograms = true      ,
1145   parse-units              = true      ,
1146   per-mode                 = power      ,
1147   per-symbol               = /          ,
1148   qualifier-mode           = subscript ,
1149   qualifier-phrase         =           ,
1150   sticky-per               = false     ,
1151   unit-font-command        = \mathrm
1152 }
1153 \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 176.)

```
\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 176.)

\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 177.)

\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 177.)

\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 177.)

\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 176.)

\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 177.)

\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 176.)

\K Temperatures.

```

68 \siunitx_declare_unit:Nn \K { \kelvin }

```

(End definition for \K. This function is documented on page 176.)

\dB

```

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

```

(End definition for \dB. This function is documented on page 178.)

\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 178.)

\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 177.)

`\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 178.)

`\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 178.)

`\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 176.)

```
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 182.)

```
\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 182.)

```
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. Where the `soulpos` package is loaded *after* `siunitx`, the commands `\hl` and `\ul` will be created only after the hook is used. The `soul` package creates those using `\newcommand`, so we have to avoid an issue.

```

37 \cs_new_protected:Npn \siunitx_command_create:
38 {
39   \bool_if:NT \l__siunitx_command_create_bool
40   {
41     \__siunitx_command_create:
42     \@ifpackageloaded { soulpos }
43     {
44       \@ifpackageloaded { soul }
45       { }
46       {
47         \cs_undefine:N \hl
48         \cs_undefine:N \ul
49       }
50     }
51   { }
52 }
```

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

```

53 \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
54 {
55   \cs_if_free:NT ##1
56   { \cs_set_protected:Npn ##1 { \ERROR } }
57 }
58 }
59 \AtBeginDocument { \siunitx_command_create: }
60 \cs_new_protected:Npn \__siunitx_command_create:
61 {
62   \bool_if:NT \l__siunitx_command_xspace_bool
63   { \RequirePackage { xspace } }
64   \bool_if:NT \l__siunitx_command_overwrite_bool
65   {
66     \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
67     { \cs_undefine:N ##1 }
68   }
69   \cs_set_protected:Npx \__siunitx_command_create:N ##1
70   {
71     \ProvideDocumentCommand ##1 { \bool_if:NT \l__siunitx_command_optarg_bool { o } }
72     {
73       \mode_leave_vertical:
74       \group_begin:
75       \bool_if:NTF \l__siunitx_command_optarg_bool
76       { \exp_not:N \IfNoValueTF {####1} }
77       { \use_i:nn }
78       {
79         \siunitx_unit_options_apply:n {##1}
80         \bool_if:NT \l__siunitx_command_prespace_bool { \exp_not:N \ }

```

```

81         \siunitx_unit_format:nN {##1}
82         \exp_not:N \l__siunitx_command_tmp_tl
83         \siunitx_print_unit:V
84         \exp_not:N \l__siunitx_command_tmp_tl
85     }
86     { \siunitx_quantity:nn {####1} {##1} }
87 \group_end:
88 \bool_if:NT \l__siunitx_command_xspace_bool { \exp_not:N \xspace }
89 }
90 }
91 \seq_map_function:NN \l_siunitx_unit_seq \__siunitx_command_create:N
92 }
93 \cs_new_protected:Npn \__siunitx_command_create:N #1 { }

```

(End definition for `\siunitx_command_create:`, `__siunitx_command_create:`, and `__siunitx_command_create:N`. This function is documented on page [183](#).)

2.3 Standard settings for module options

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

```

94 \keys_set:nn { siunitx }
95 {
96     free-standing-units      = false ,
97     overwrite-commands      = false ,
98     space-before-unit       = false ,
99     unit-optional-argument  = false ,
100    use-xspace                = false
101 }
102 \</package>

```

Part XIV

siunitx-emulation – Emulation

1 siunitx-emulation implementation

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

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

_siunitx_option_deprecated:nn

Abstract out a simple wrapper.

_siunitx_option_deprecated:nnn

_siunitx_option_deprecated:nnV

```
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

Abstract out a simple wrapper.

_siunitx_option_removed:V

```
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     prefixes-as-symbols .choice: ,
601     prefixes-as-symbols / false . code:n =
602     {
603         \_siunitx_option_deprecated:nn
604         { prefixes-as-symbols~==false }
605         { prefix-mode~==extract-exponent }
606     } ,
607     prefixes-as-symbols / true . code:n =
608     {
609         \_siunitx_option_deprecated:nn
610         { prefixes-as-symbols~==true }
611         { prefix-mode~==input }
612     } ,
613     prefixes-as-symbols .default:n = true
614 }
615 </options>

```

1.10 Preamble commands

```

616 <*interfaces>

```

`\DeclareBinaryPrefix` We simply drop #3.

```

617 \NewDocumentCommand \DeclareBinaryPrefix { +m m m }
618 {
619     \siunitx_declare_prefix:Nn #1 {#2}
620 }

```

(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.

```

621 \NewDocumentCommand \DeclareSIPrePower { +m m }
622 {
623     \siunitx_declare_power:NNn #1 \_siunitx_tmp:w {#2}
624     \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
625 }
626 \NewDocumentCommand \DeclareSIPostPower { +m m }
627 {
628     \siunitx_declare_power:NNn \_siunitx_tmp:w #1 {#2}
629     \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
630 }

```

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

1.11 Document commands

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

```

631 \NewDocumentCommand \si { O { } m }
632 {
633   \mode_leave_vertical:
634   \group_begin:
635     \keys_set:nn { siunitx } {#1}
636     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
637     \siunitx_print_unit:V \l__siunitx_tmp_tl
638   \group_end:
639 }

```

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

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

```

640 \NewDocumentCommand \SI { O { } m o m }
641 {
642   \mode_leave_vertical:
643   \group_begin:
644     \keys_set:nn { siunitx } {#1}
645     \IfNoValueF {#3}
646     {
647       \siunitx_unit_format:nN {#3} \l__siunitx_tmp_tl
648       \siunitx_print_unit:V \l__siunitx_tmp_tl
649       \nobreak
650     }
651     \siunitx_quantity:nn {#2} {#4}
652   \group_end:
653 }

```

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

`\SIlist` Straight copies.

`\SIrange`

```

654 \NewDocumentCommand \SIlist
655 { O { } > { \SplitList { ; } } m > { \TrimSpaces } m }
656 {
657   \mode_leave_vertical:
658   \group_begin:
659     \siunitx_unit_options_apply:n {#3}
660     \keys_set:nn { siunitx } {#1}
661     \siunitx_quantity_list:nn {#2} {#3}
662   \group_end:
663 }
664 \NewDocumentCommand \SIrange { O { } m m > { \TrimSpaces } m }
665 {
666   \mode_leave_vertical:
667   \group_begin:
668     \siunitx_unit_options_apply:n {#4}
669     \keys_set:nn { siunitx } {#1}
670     \siunitx_quantity_range:nnn {#2} {#3} {#4}
671   \group_end:
672 }

```

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

1.12 Symbol commands

673 <@@=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.

```
674 \bool_lazy_or:nnTF
675   { \sys_if_engine luatex_p: }
676   { \sys_if_engine xetex_p: }
677   {
678     \cs_new:Npn \_siunitx_emulation_non_latin:n #1
679       { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
680   }
681   {
682     \cs_new:Npn \_siunitx_emulation_non_latin:n #1
683       {
684         \exp_last_unbraced:Nf \_siunitx_emulation_non_latin:nnnn
685         { \char_to_utfviii_bytes:n {#1} }
686       }
687     \cs_new:Npn \_siunitx_emulation_non_latin:nnnn #1#2#3#4
688       {
689         \exp_after:wN \exp_after:wN \exp_after:wN
690         \exp_not:N \char_generate:nn {#1} { 13 }
691         \exp_after:wN \exp_after:wN \exp_after:wN
692         \exp_not:N \char_generate:nn {#2} { 13 }
693       }
694   }
```

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

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

The same setup as elsewhere but localised to the emulation module

```
695 \AtBeginDocument
696   {
697     \cs_new_protected:Npn \SIUnitSymbolArcminute
698       { \ensuremath { { } ^\circ } }
699     \cs_new_protected:Npn \SIUnitSymbolArcsecond
700       { \ensuremath { { } ^\circ } }
701     \ifpackageloaded { fontspec }
702       {
703         \cs_new_protected:Npx \SIUnitSymbolAngstrom
704           { \_siunitx_emulation_non_latin:n { "00C5 } }
705         \cs_new_protected:Npx \SIUnitSymbolDegree
706           { \_siunitx_emulation_non_latin:n { "00B0 } }
707         \cs_new_protected:Npx \SIUnitSymbolCelsius
708           { \_siunitx_emulation_non_latin:n { "00B0 } C }
709       }
710     {
711       \cs_new_protected:Npx \SIUnitSymbolAngstrom
712         {
713           \siunitx_print_text:n
714           { \_siunitx_emulation_non_latin:n { "00C5 } }
715         }
716       \cs_new_protected:Npx \SIUnitSymbolCelsius
717         {
718           \siunitx_print_text:n
719           { \_siunitx_emulation_non_latin:n { "00B0 } } C
720         }
721     }
```

```

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

```

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

`\celsius` Deprecated but should work.

```

756 \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.

```

757 \msg_new:nnnn { siunitx } { unit-deprecated }
758 { Unit-macro~#1~has-been-deprecated-in-this-release. }
759 {
760   The~BIPM~have~removed~this~unit~from~the~SI~Brochure.~
761   You~should~define~it~yourself~using~\token_to_str:N \DeclareSIUnit
762   in~your~source.
763 }
764 \cs_gset_protected:Npn \_siunitx_emulation_tmp:w #1#2
765 {
766   \quark_if_recursion_tail_stop:N #1

```

```

767 \bool_new:c { g__siunitx_emulation_unit_warning_ \token_to_str:N #1 _bool }
768 \siunitx_declare_unit:Nx #1
769 {
770   \exp_not:N \bool_if:NF
771   \exp_not:c { g__siunitx_emulation_unit_warning_ \token_to_str:N #1 _bool }
772   {
773     \exp_not:N \bool_gset_true:N
774     \exp_not:c { g__siunitx_emulation_unit_warning_ \token_to_str:N #1 _bool }
775     \msg_warning:nnn { siunitx } { unit-deprecated }
776     { \token_to_str:N #1 }
777   }
778   #2
779 }
780 \__siunitx_emulation_tmp:w
781 }
782 \__siunitx_emulation_tmp:w
783 \atomicmassunit { u }
784 \bar { bar }
785 \barn { b }
786 \bohr
787 { \text { \ensuremath { a } } } \char_generate:nn { '\_ } { 8 } { 0 } }
788 \clight
789 { \text { \ensuremath { c } } } \char_generate:nn { '\_ } { 8 } { 0 } }
790 \electronmass
791 {
792   \text { \ensuremath { m } } }
793   \char_generate:nn { '\_ } { 8 } { \exp_not:N \mathrm { e } } }
794 }
795 \elementarycharge { \text { \ensuremath { e } } } }
796 \hartree
797 {
798   \text { \ensuremath { E } } }
799   \char_generate:nn { '\_ } { 8 } { \exp_not:N \mathrm { h } } }
800 }
801 \knot { kn }
802 \mmHg { mmHg }
803 \nauticalmile { M }
804 \planckbar
805 { \text { \ensuremath { \exp_not:N \hbar } } } }
806 \q_recursion_tail { }
807 \q_recursion_stop
808 \ifpackageloaded { fontspec }
809 {
810   \__siunitx_emulation_tmp:w \angstrom { \__siunitx_emulation_non_latin:n { "00C5 } }
811 }
812 {
813   \__siunitx_emulation_tmp:w \angstrom
814   { \exp_not:N \text { \__siunitx_emulation_non_latin:n { "00C5 } } } }
815 }
816 \q_recursion_tail { }
817 \q_recursion_stop
818 </interfaces>
819 </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

$\%$	1093
\backslash	<i>104</i> , <i>136</i> , 9, 15, 21, 27, 156, 1143, 1978, 1984
$\backslash-$	101
$\backslash\backslash$	6, 84, 1105
$\backslash_$	8, 197, 208, 281, 358, 787, 789, 793, 799
$\backslash\sim$	212
$\backslash\sqcup$	48, 50, 54, 56, 60, 62, 80, 114, 538, 540, 541, 542, 546, 548, 551, 557, 559, 563, 565, 568, 574, 576, 583, 585, 1112

A

$\backslash A$	<i>176</i> , <u>2</u>
allow-quantity-breaks	<i>104</i>
\backslash ampere	<i>140</i> , 2, 3, 4, 5, 6, 7, <u>1025</u>
\backslash ang	6, <u>100</u> , 277, 313
angle-mode	12
angle-symbol-degree	12
angle-symbol-minute	12
angle-symbol-over-decimal	12
angle-symbol-second	12
\backslash angstrom	810, 813
\backslash approx	1980
arc-separator	12
\backslash arcminute	<i>141</i> , 51, 53, 286, 353, <u>1090</u>
\backslash arcsecond	<i>141</i> , 56, 58, 288, 358, <u>1090</u>
\backslash as	<i>176</i> , <u>89</u>
\backslash astronomicalunit	<i>141</i> , <u>1077</u>
\backslash AtBeginDocument	5, 17, 37, 49, 59, 73, 129, 218, 221, 244, 297, 318, 695
\backslash atomicmassunit	783
\backslash atto	<i>140</i> , 90, <u>1034</u>

B

\backslash bar	784
\backslash barn	785
\backslash becquerel	<i>141</i> , <u>1055</u>
\backslash begin	203
\backslash bel	<i>141</i> , 69, <u>1077</u>
\backslash bfseries	91
\backslash binoppenalty	146
\backslash bit	182, <u>10</u>
\backslash bohr	786
\backslash boldmath	92
bool commands:	
\backslash bool_gset_true:N	773

\backslash bool_if:NTF	10, 16, 17, 18, 32, 34, 39, 47, 50, 61, 62, 64, 71, 75, 80, 88, 91, 96, 101, 106, 108, 110, 116, 119, 121, 126, 144, 145, 149, 158, 164, 164, 182, 183, 186, 192, 209, 233, 239, 241, 243, 253, 255, 256, 265, 376, 380, 397, 407, 415, 437, 438, 447, 475, 476, 480, 493, 574, 582, 600, 620, 660, 692, 715, 751, 766, 770, 775, 853, 873, 885, 902, 922, 936, 945, 948, 995, 1005, 1282, 1467, 1641, 1647, 1663, 1698, 1871, 1892
\backslash bool_lazy_all:nTF	1604
\backslash bool_lazy_all_p:n	1093, 1111
\backslash bool_lazy_and:nnTF	111, 137, 217, 268, 283, 431, 515, 592, 644, 675, 699, 887, 895, 1375, 1626
\backslash bool_lazy_any:nTF	245
\backslash bool_lazy_or:nnTF	10, 105, 120, 159, 171, 257, 317, 387, 389, 443, 543, 674, 731, 747, 1004, 1090, 1108, 1406, 1677, 1838, 1854, 1867
\backslash bool_lazy_or_p:nn	1610
\backslash bool_new:N	3, 6, 9, 10, 10, 11, 12, 18, 19, 20, 21, 22, 23, 43, 44, 76, 88, 348, 446, 561, 566, 567, 568, 569, 570, 571, 574, 662, 767, 1507, 1569, 1570
\backslash bool_set_false:N	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, 339, 354, 355, 457, 513, 514, 520, 521, 522, 523, 527, 528, 529, 534, 537, 541, 599, 601, 651, 675, 714, 762, 797, 869, 1528, 1532, 1537, 1538
\backslash bool_set_true:N	11, 14, 17, 24, 30, 30, 34, 35, 52, 56, 62, 63, 69, 76, 102, 141, 163, 208, 209, 234, 243, 254, 261, 328, 353, 449, 455, 515, 516, 530, 535, 536, 542, 543, 544, 548, 549, 550, 551, 602, 649, 859, 890, 929, 1522, 1523, 1527, 1533, 1904, 1905
\backslash c_false_bool	47, 121, 387, 449, 453, 458, 466, 490, 496, 534, 549, 591, 614

<code>\c_true_bool</code>	44, 118, 372, 387, 447, 463, 490, 496, 547, 618
box commands:	
<code>\box_clear:N</code>	505, 568
<code>\box_new:N</code> 3, 170, 171, 247, 248, 480, 481	
<code>\box_use:N</code>	249
<code>\box_use_drop:N</code>	244, 246, 428, 429, 474, 475, 528, 536, 551, 552, 607, 608, 609, 610
<code>\box_wd:N</code>	228, 229, 263, 268, 271, 272, 277, 278, 288, 289, 403, 445, 449, 463, 503, 510, 511, 514, 522, 566, 571, 598, 623, 634, 635, 646, 650, 651, 664, 665, 675, 683, 685, 703, 704, 725, 736, 755, 757, 767, 779
bracket-ambiguous-numbers	40
bracket-negative-numbers	40
bracket-unit-denominator	142
<code>\byte</code>	182, 10
C	
<code>\cancel</code>	136, 142, 148, 107
<code>\candela</code>	140, 1025
<code>\cdot</code>	8, 33, 279
<code>\celsius</code>	756
<code>\centi</code>	140, 65, 1034
char commands:	
<code>\char_generate:nn</code>	8, 15, 26, 28, 164, 175, 177, 197, 208, 281, 358, 679, 690, 692, 787, 789, 793, 799, 1009, 1020, 1022
<code>\char_set_active_eq:NN</code>	222, 224, 226, 409, 453
<code>\char_set_catcode_active:N</code>	101
<code>\char_set_catcode_active:n</code>	212
<code>\char_set_mathcode:nn</code>	410, 454
<code>\char_to_utfviii_bytes:n</code>	21, 170, 685, 1015
<code>\char_value_catcode:n</code>	15, 164, 679, 1009
<code>\clight</code>	788
clist commands:	
<code>\clist_map_break:</code>	119
<code>\clist_map_function:nN</code>	34, 39
<code>\clist_map_inline:nn</code> 19, 28, 95, 97, 107, 123, 165, 174, 198, 253, 492, 604	
<code>\cm</code>	176, 60
<code>\color</code>	38, 258, 1629
color	92
<code>\complexnum</code>	181
<code>\complexqty</code>	181
compound-exponents	21
compound-final-separator	21
compound-pair-separator	21
compound-separator	21
compound-separator-mode	21
compound-units	21
<code>\coulomb</code>	141, 1055
<code>\cr</code>	117, 29
cs commands:	
<code>\cs:w</code>	117, 182, 458, 714, 1019
<code>\cs_end:</code>	117, 182, 459, 717, 1022
<code>\cs_generate_variant:Nn</code>	2, 3, 3, 4, 5, 5, 6, 21, 27, 61, 62, 65, 72, 87, 114, 123, 144, 151, 181, 188, 193, 199, 205, 207, 246, 302, 368, 373, 380, 497, 499, 788, 843, 871, 919, 1032, 1194, 1197, 1208, 1695, 1813, 1828
<code>\cs_gset_protected:Npn</code>	764
<code>\cs_if_eq:NNTF</code>	347
<code>\cs_if_exist:NTF</code>	99, 113, 116, 223, 250, 333, 741
<code>\cs_if_free:NTF</code>	7, 55
<code>\cs_new:Npn</code> 14, 18, 23, 67, 111, 121, 136, 138, 163, 167, 168, 172, 209, 210, 229, 240, 246, 247, 251, 344, 346, 361, 361, 363, 369, 370, 381, 383, 384, 386, 387, 389, 466, 468, 471, 474, 477, 480, 483, 555, 613, 671, 678, 682, 687, 690, 708, 725, 727, 731, 738, 740, 746, 766, 778, 789, 795, 801, 807, 819, 826, 844, 866, 872, 873, 880, 900, 905, 919, 929, 942, 950, 963, 977, 983, 988, 1001, 1008, 1012, 1013, 1017, 1025, 1027, 1033, 1042, 1058, 1072, 1083, 1104, 1122, 1152, 1179, 1186, 1193, 1195, 1198, 1209, 1219, 1226, 1233, 1234, 1235, 1242, 1244, 1250, 1255, 1269, 1277, 1286, 1301, 1309, 1327, 1343, 1344, 1362, 1371, 1373, 1395, 1404, 1421, 1434, 1440, 1449, 1473, 1483, 1485, 1490, 1495, 1497, 1572, 1574, 1576, 1578, 1580, 1585, 1590, 1602, 1617, 1624, 1631, 1637, 1655, 1661, 1667, 1675, 1682, 1696, 1707, 1716, 1718, 1720, 1722, 1728, 1738, 1743, 1763, 1772, 1774, 1793, 1814, 1829, 1834, 1836, 1845, 1851, 1865, 1882, 1888, 1898, 1933, 1939, 1944	
<code>\cs_new:Npx</code>	245, 391, 400
<code>\cs_new_eq:NN</code>	222, 249, 250, 764
<code>\cs_new_protected:Npn</code>	7, 11, 14, 14, 16, 22, 23, 24, 25, 27, 28, 37, 37, 39, 40, 40, 45, 45, 49, 50, 55, 59, 60, 60, 61, 62, 63, 64, 66, 67, 68, 70, 70, 73, 77, 78, 80,

exp commands:	
\exp_after:wN	365, 366, 367, 370, 392, 394, 395,
25, 27, 68, 80, 104, 115, 135, 137,	396, 397, 397, 402, 435, 436, 437,
140, 144, 148, 149, 153, 168, 170,	439, 467, 470, 473, 474, 476, 479,
174, 176, 181, 185, 205, 231, 235,	481, 482, 482, 485, 489, 490, 491,
245, 255, 328, 330, 335, 358, 371,	530, 558, 560, 589, 610, 679, 682,
372, 382, 436, 459, 460, 461, 463,	694, 695, 696, 698, 703, 728, 733,
482, 507, 534, 581, 594, 595, 669,	736, 739, 741, 748, 749, 750, 756,
689, 691, 691, 703, 716, 719, 722,	774, 798, 818, 819, 820, 821, 829,
731, 961, 998, 1008, 1019, 1021,	830, 831, 832, 834, 842, 843, 851,
1021, 1264, 1389, 1470, 1583, 1941	855, 856, 872, 882, 902, 909, 913,
\exp_args:Nc	913, 914, 915, 915, 916, 923, 926,
\exp_args:Ne	927, 933, 937, 938, 939, 954, 963,
\exp_args:Nf	964, 965, 965, 966, 968, 971, 973,
727, 742, 1067, 1078, 1324, 1436, 1442	974, 975, 975, 982, 983, 990, 998,
\exp_args:Nff	999, 1000, 1002, 1014, 1026, 1292,
\exp_args:NNc	1295, 1299, 1333, 1341, 1343, 1379,
\exp_args:Nmno	1384, 1385, 1386, 1393, 1402, 1409,
\exp_args:NNNV	1413, 1416, 1418, 1475, 1477, 1480,
25, 130, 239, 331, 586, 677, 799, 957	1481, 1492, 1500, 1501, 1619, 1621,
\exp_args:NNnx	1629, 1634, 1635, 1653, 1659, 1664,
\exp_args:NNV	1665, 1670, 1672, 1684, 1690, 1692,
\exp_args:NnV	1702, 1705, 1724, 1733, 1734, 1747,
\exp_args:NV	1748, 1752, 1756, 1773, 1778, 1780,
73, 114, 211, 230, 312, 1222, 1488	1786, 1787, 1789, 1835, 1853, 1863,
\exp_args:Nv	1874, 1875, 1877, 1878, 1884, 1885
\exp_args:Nx	\expandafter
\exp_args:Nxx	\ExplFileDate
\exp_last_unbraced:Nf	exponent-base
20, 169, 684, 1014, 1160, 1423	exponent-mode
\exp_not:N	exponent-product
26, 28, 36, 54, 59, 76, 77, 80, 82, 84,	expression
88, 98, 100, 101, 102, 105, 109,	extract-mass-in-kilograms
111, 124, 135, 136, 175, 177, 193,	
194, 195, 195, 196, 198, 199, 200,	F
201, 201, 202, 202, 203, 209, 217,	\F
217, 218, 218, 220, 222, 223, 224,	\fam
225, 226, 227, 269, 270, 271, 272,	\familydefault
273, 276, 277, 278, 278, 279, 280,	\farad
281, 282, 282, 283, 284, 284, 285,	\femto
285, 287, 287, 288, 289, 289, 290,	\fF
310, 311, 312, 316, 317, 318, 321,	\fg
336, 337, 338, 354, 355, 357, 359,	\fi
393, 396, 403, 404, 591, 690, 692,	540, 542, 546, 548, 551, 557, 559,
735, 740, 742, 743, 744, 751, 753,	563, 565, 568, 574, 576, 583, 585, 753
770, 771, 773, 774, 793, 799, 805,	fi commands:
814, 1020, 1022, 1629, 1688, 1731, 1872	\fi:
\exp_not:n	file commands:
110, 111, 112, 113, 114, 114,	\file_if_exist:nTF
130, 139, 161, 169, 201, 215, 216,	fill-angle-degrees
221, 228, 242, 274, 279, 280, 287,	fill-angle-minutes
288, 288, 289, 290, 291, 295, 316,	fill-angle-seconds
317, 318, 324, 328, 330, 343, 347,	fixed-exponent
348, 353, 354, 356, 357, 361, 362,	\fmol
	\fmtversion

<code>\fontfamily</code>	91, 240	264, 279, 307, 331, 342, 361, 420,
<code>\fontseries</code>	91, 242	427, 469, 476, 516, 523, 586, 638,
<code>\fontshape</code>	91, 244	652, 662, 671, 677, 799, 957, 1909, 1913
<code>\fontsize</code>	367	
<code>forbid-literal-units</code>	143	
fp commands:		
<code>\fp_add:Nn</code>	787	
<code>\fp_compare:nNnTF</code>		
.....	474, 588, 590, 638, 658	
<code>\fp_eval:n</code>	48,	
.....	64, 73, 74, 76, 122, 473, 637, 1493	
<code>\fp_new:N</code>	4, 4, 573, 575, 576	
<code>\fp_set:Nn</code>		
.....	135, 136, 143, 149, 150, 157,	
.....	165, 172, 173, 202, 249, 622, 656, 684	
<code>\fp_use:N</code>	203, 662	
<code>\fp_zero:N</code>	135, 142, 156, 164, 182, 586	
<code>\c_one_fp</code>	136, 143, 150, 590, 658	
<code>\l_tmpa_fp</code>	137	
<code>\c_zero_fp</code>	588, 638	
<code>\frac</code>	136, 1142	
<code>fraction-command</code>	143	
<code>free-standing-units</code>	183	
<code>\fs</code>	176, 89	
G		
<code>\g</code>	176, 35	
<code>\ge</code>	38, 98, 1980	
<code>\geq</code>	1980	
<code>\GetTranslation</code>	49, 55, 61	
<code>\GeV</code>	177, 42	
<code>\gg</code>	38, 97, 1980	
<code>\GHz</code>	176, 8	
<code>\gibi</code>	182, 2	
<code>\giga</code>	140, 12, 47, 56, 85, 1044	
<code>\GPa</code>	178, 82	
<code>\gram</code>	140, 155, 35,	
.....	36, 37, 38, 39, 40, 41, 429, 1025, 1033	
<code>\gray</code>	141, 1055	
group commands:		
<code>\group_begin:</code>		
.....	139, 13, 16, 26, 33, 42, 74,	
.....	74, 87, 93, 94, 100, 103, 111, 117,	
.....	120, 131, 140, 149, 159, 167, 176,	
.....	178, 184, 192, 193, 201, 211, 215,	
.....	215, 252, 260, 303, 327, 339, 353,	
.....	417, 424, 466, 473, 513, 520, 573,	
.....	634, 643, 658, 667, 674, 796, 955, 1903	
<code>\c_group_begin_token</code> ...	41, 151, 389	
<code>\group_end:</code>	139,	
.....	16, 25, 41, 45, 59, 82, 87, 91, 93, 98,	
.....	106, 107, 112, 115, 125, 130, 135,	
.....	138, 143, 153, 162, 171, 179, 187,	
.....	196, 203, 205, 206, 219, 239, 244,	
.....	264, 279, 307, 331, 342, 361, 420,	
.....	427, 469, 476, 516, 523, 586, 638,	
.....	652, 662, 671, 677, 799, 957, 1909, 1913	
<code>group-digits</code>	41	
<code>group-minimum-digits</code>	41	
<code>group-separator</code>	41	
<code>\GW</code>	177, 42	
H		
<code>\H</code>	75, 334, 337, 338	
<code>\hartree</code>	796	
<code>\hbar</code>	805	
hbox commands:		
<code>\hbox_overlap_right:n</code>	243, 245	
<code>\hbox_set:Nn</code>	214,	
.....	219, 265, 400, 442, 447, 502, 508,	
.....	541, 543, 564, 565, 596, 676, 743, 771	
<code>\hbox_set:Nw</code>	405, 417	
<code>\hbox_set_end:</code>	416, 424, 461, 471	
<code>\hbox_set_to_wd:Nnn</code>		
.....	240, 264, 275, 286, 402, 444,	
.....	513, 521, 597, 622, 648, 662, 689, 753	
<code>\hbox_set_to_wd:Nnw</code>	448, 462	
<code>\hbox_to_wd:nn</code>	210	
<code>\hbox_unpack:N</code>	281, 293, 517, 524,	
.....	654, 669, 678, 692, 760, 762, 773, 774	
<code>\hbox_unpack_drop:N</code>	268	
<code>\hectare</code>	141, 1077	
<code>\hecto</code>	140, 27, 31, 1044	
<code>\henry</code>	141, 75, 76, 77, 1055	
<code>\hertz</code>	141, 8, 9, 10, 11, 12, 13, 1055	
<code>\highlight</code>	142, 148, 107	
<code>\hL</code>	177, 27	
<code>\hl</code>	177, 185, 27, 47	
<code>\hour</code>	141, 58, 1077	
<code>\Hz</code>	176, 8	
I		
if commands:		
<code>\if_false:</code>	28, 34	
<code>\if_meaning:w</code>	327	
<code>\IfFormatAtLeastTF</code>	42, 59	
<code>\ifmmode</code>	48, 50, 54, 56, 60, 62, 98,	
.....	540, 542, 546, 548, 551, 557, 559,	
.....	563, 565, 568, 574, 576, 583, 585, 740	
<code>\IfNoValueF</code>	645	
<code>\IfNoValueTF</code>	76, 76	
<code>\ignorespaces</code>	114, 37, 204, 386	
<code>input-close-uncertainty</code>	41	
<code>input-comparators</code>	41	
<code>input-decimal-markers</code>	41	
<code>input-digits</code>	41	
<code>input-exponent-markers</code>	41	
<code>input-open-uncertainty</code>	41	

\mbox	91, 136, 365, 376, 377	\mV	177, 21
\mdseries	92	\MW	177, 42
\mebi	182, 2	\mW	177, 42
\mega	140, 11, 46, 55, 81, 84, 88, 1044	N	
\MessageBreak	18	\N	177, 78
\meter	140, 172, 1025	\nA	176, 2
\metre	140, 141, 154, 172, 60, 61, 62, 63, 64, 65, 66, 67, 1025	\nano	140, 4, 17, 23, 37, 62, 73, 93, 1034
\MeV	177, 42	\nauticalmile	803
\mEV	177, 42	negative-color	42
\mg	176, 35	\neper	141, 1077
\mH	75	\newcolumnstype	228, 255
MHz	176, 8	\NewDocumentCommand	62, 66,
\mHz	176, 8		70, 74, 91, 100, 108, 117, 127, 137,
\micro	140, 5, 18, 24, 30, 34, 38, 43, 49, 63, 74, 77, 94, 114, 116, 1034		145, 155, 164, 173, 181, 189, 198,
\milli	140, 6, 9, 19, 25, 29, 33, 39, 44, 50, 53, 64, 76, 79, 86, 95, 1034		208, 617, 621, 626, 631, 640, 654, 664
minimum-decimal-digits	42	\newton	141, 78, 79, 80, 81, 1066
minimum-integer-digits	42	\nF	178, 70
\minute	141, 1077	\ng	176, 35
\mJ	177, 42	\nm	176, 60
\mL	177, 27	\nmol	177, 14
\ml	177, 27	\nobreak	147, 194, 277, 649
\mm	176, 60	\ns	176, 89
\mmHg	802	\num	135, 100, 278, 313
\mmol	177, 14	number-angle-product	13
\MN	177, 78	number-color	92
\mN	177, 78	number-mode	92
mode	92	\numlist	127, 280, 314
mode commands:		\numproduct	127, 294
\mode_if_math:TF	82, 135, 207	\numrange	173, 296, 315
\mode_leave_vertical:	73, 93, 102, 110, 119, 130, 139, 148, 158, 166, 175, 183, 191, 200, 633, 642, 657, 666	\nV	177, 21
\Mohm	178, 86	O	
\mohm	178, 86	\of	142, 113
\mol	177, 14	\ohm	141, 86, 87, 88, 94, 96, 136, 1066
\mole	140, 14, 15, 16, 17, 18, 19, 20, 1025	\Omega	101, 743
\mp	38, 93, 273, 274, 1986	output-close-uncertainty	42
\MPa	178, 82	output-decimal-marker	42
\ms	176, 89	output-open-uncertainty	42
msg commands:		\over	144
\msg_error:nn	451	overwrite-commands	183
\msg_error:nnn	31, 160, 193, 497, 630, 645	P	
\msg_error:nnnn	369, 402, 416	\Pa	178, 82
\msg_info:nnnn	13, 18, 234, 404, 421, 438, 445, 453, 536	\pA	176, 2
\msg_new:nnn	4, 9, 35, 82, 242	\PackageError	15
\msg_new:nnnn	25, 757, 1096, 1102, 1108, 1114, 1120, 1126, 1132, 1957	parse-numbers	42
\msg_warning:nn	87	parse-units	143
\msg_warning:nnn	24, 31, 226, 253, 775	\pascal	141, 82, 83, 84, 85, 1066
		\pdfstringdefDisableCommands	133, 322, 330
		\pebi	182, 2
		peek commands:	
		\peek_after:Nw	334

82, 83, 84, 85, 86, 86, 87, 88, 89, 90,
 91, 92, 93, 94, 95, 96, 136, 756, 768,
 1025, 1026, 1027, 1028, 1029, 1030,
 1031, 1032, 1033, 1055, 1056, 1057,
 1058, 1059, 1060, 1061, 1062, 1063,
 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
 \siunitx_declare_unit:Nnn
 138, 139, 58, 66, 70, 78, 180
 \siunitx_format_number:nN 12
 \siunitx_if_number:nTF 40, 1900
 \siunitx_if_number_p:n 40
 \siunitx_if_number_token:NTF ...
 40, 167, 1917
 \siunitx_if_number_token_p:N . 1917
 \l_siunitx_list_separator_final_-
 tl 20, 285, 292, 307, 401, 435
 \l_siunitx_list_separator_pair_-
 tl 20, 283, 290, 305, 401, 437
 \l_siunitx_list_separator_tl ...
 20, 284, 291, 306, 401, 439
 \siunitx_number_adjust_exponent:Nn
 39, 84, 232, 1483
 \siunitx_number_adjust_exponent:nn
 39, 1483
 \l_siunitx_number_bracket_-
 ambiguous_bool
 63, 260, 1507, 1512, 1607
 \l_siunitx_number_comparator_tl . 40
 \l_siunitx_number_exponent_tl ... 40
 \siunitx_number_format:nN
 39, 14, 113, 173, 788
 \l_siunitx_number_input_comparator_-
 tl 29, 250, 1923
 \l_siunitx_number_input_decimal_-
 tl 40,
 28, 40, 390, 405, 407, 451, 561, 1921
 \l_siunitx_number_input_exponent_-
 tl 29, 261, 269, 270, 1925
 \l_siunitx_number_input_sign_tl .
 29, 309, 417, 512, 1928
 \siunitx_number_list:nn . 20, 142, 415
 \siunitx_number_normalize_-
 symbols:N 40, 77, 126
 \siunitx_number_output:N
 39, 22, 125, 759, 1572
 \siunitx_number_output:n ... 39, 1572
 \siunitx_number_output:NN
 38, 39, 52, 67, 119,
 129, 134, 169, 435, 532, 590, 592, 1572
 \siunitx_number_output:nN .. 39, 1572
 \l_siunitx_number_output_-
 decimal_tl 40, 266,
 401, 419, 465, 1508, 1550, 1687, 1690
 \siunitx_number_parse:nN
 . 38, 39, 104, 114, 19, 50, 64, 107,
 108, 152, 166, 330, 529, 572, 668, 1906
 \l_siunitx_number_parse_bool ...
 39, 40, 9,
 10, 10, 17, 18, 50, 61, 96, 110, 328, 1905
 \siunitx_number_process:N 39
 \siunitx_number_process:NN
 39, 20, 72, 87, 91, 110,
 167, 210, 233, 238, 243, 530, 585, 685
 \siunitx_number_product:n 20, 161, 464
 \siunitx_number_range:nn 20, 178, 511
 \l_siunitx_number_sign_tl 40
 \siunitx_prin_number:n 92
 \siunitx_print....:n 28, 91–93
 \siunitx_print_match:n 91, 80
 \siunitx_print_math:n 91, 83, 99
 \siunitx_print_number:n 91,
 60, 90, 114, 125, 141, 210, 217,
 251, 272, 273, 274, 274, 276, 378,
 542, 545, 627, 656, 670, 679, 746, 789
 \siunitx_print_text:n
 91, 68, 84, 88, 103,
 118, 135, 235, 713, 718, 723, 729, 745
 \siunitx_print_unit:n .. 91, 104,
 47, 60, 83, 124, 148, 224, 280, 637, 648
 \siunitx_quantity_print:nn 139
 \siunitx_quantity:nn
 104, 40, 86, 97, 651
 \siunitx_quantity_list:nn
 20, 134, 415, 661
 \l_siunitx_quantity_prefix_mode_-
 tl 9, 65, 184, 188, 263
 \siunitx_quantity_print:nn
 104, 55,
 104, 108, 121, 123, 127, 139, 151, 372
 \siunitx_quantity_product:nn ...
 20, 152, 464
 \siunitx_quantity_range:nnn
 20, 170, 511, 670
 \l_siunitx_range_phrase_tl
 21, 297, 299, 310, 499, 530
 \l_siunitx_unit_font_tl
 138, 88, 91,
 122, 317, 330, 819, 831, 842, 855, 926
 \siunitx_unit_format:nN 104,
 136, 137, 149, 38, 46, 54, 79, 81,
 92, 123, 132, 223, 239, 279, 636, 647
 \siunitx_unit_format_combine_-
 exponent:nnN 137, 75, 132, 195

\siunitx_unit_format_extract_-	\l__siunitx_angle_force_arc_bool
prefixes:nNN 137 , 80 , 132 , 218 6 , 47
\siunitx_unit_format_multiply:nN	\l__siunitx_angle_force_decimal_-
..... 137 , 132 , 244	bool 6 , 61
\siunitx_unit_format_multiply_-	\l__siunitx_angle_marker_box ...
combine_exponent:nnnN 137 , 132 , 203	170 , 214 , 228 , 232 , 238 , 240 , 244 , 249
\siunitx_unit_format_multiply_-	\l__siunitx_angle_minutes_tl 87 , 136
extract_prefixes:nnNN 137 , 132 , 224	\l__siunitx_angle_product_tl 6 , 278
\l_siunitx_unit_fraction_tl	\l__siunitx_angle_seconds_tl 87 , 137
..... 139 , 253 , 500 , 973	\l__siunitx_angle_separator_tl 6 , 195
\siunitx_unit_options_apply:n ...	__siunitx_angle_sign:nnnnnnn ... 91
..... 138 , 139 , 43 , 79 , 89 , 95 ,	\l__siunitx_angle_sign_tl
121 , 132 , 150 , 168 , 193 , 357 , 659 , 668 90 , 101 , 105 , 114 , 163
\siunitx_unit_pdfstring_context:	\l__siunitx_angle_symbol_degree_-
..... 139 , 332 , 337	tl 6 , 175
\siunitx_unit_power_set:NnN ... 148	\l__siunitx_angle_symbol_minute_-
\l_siunitx_unit_seq .. 139 , 22 , 75 , 91	tl 6 , 177
\l_siunitx_unit_symbolic_seq ...	\l__siunitx_angle_symbol_second_-
139 , 21 , 29 , 53 , 66 , 179 , 340 , 624 , 629	tl 6 , 179
siunitx internal commands:	\l__siunitx_angle_tmp_bool
__siunitx_angle:n 277 , 344 3 , 208 , 209 , 256
__siunitx_angle:nnn 105 , 210	\l__siunitx_angle_tmp_dim
__siunitx_angle:w 344 3 , 241 , 263 , 265
__siunitx_angle_arc_convert:n .. 45	\l__siunitx_angle_tmp_tl
__siunitx_angle_arc_print:nnn 3 , 152 , 153 , 159 , 223 , 224 , 279 , 280
..... 53 , 134 , 172	\l__siunitx_angle_unit_box
__siunitx_angle_arc_print_- 170 , 219 , 229 , 233 , 237 , 246
auxi:nnn 172	__siunitx_bookmark_cmd:Nn 270
__siunitx_angle_arc_print_-	__siunitx_bookmark_cmd:Nnn
auxii:nw 187 , 204	. 270 , 276 , 277 , 278 , 279 , 280 , 287 ,
__siunitx_angle_arc_print_-	294 , 295 , 296 , 298 , 300 , 301 , 302 , 309
auxii:w 172	\c__siunitx_bookmark_seq ... 311 , 324
__siunitx_angle_arc_print_-	\l__siunitx_column_type_tl .. 46 , 260
auxiii:n 172	__siunitx_command_create: 37
__siunitx_angle_arc_print_-	__siunitx_command_create:N 37
auxiv:NN 172	\l__siunitx_command_create_bool .
__siunitx_angle_arc_print_- 4 , 39
auxv:w 172	\l__siunitx_command_optarg_bool .
__siunitx_angle_arc_print_- 4 , 71 , 75
auxvi:n 172	\l__siunitx_command_overwrite_-
__siunitx_angle_arc_sign:nn ... 91	bool 4 , 64
__siunitx_angle_arc_sign:nnn 66 , 91	\l__siunitx_command_prespace_-
\l_siunitx_angle_astronomy_bool	bool 4 , 80
..... 6 , 186	\l__siunitx_command_tmp_tl . 3 , 82 , 84
\l__siunitx_angle_degrees_tl ...	\l__siunitx_command_xspace_bool .
..... 50 , 51 , 52 , 54 , 87 , 135 4 , 62 , 88
__siunitx_angle_extract_-	\l__siunitx_compound_bracket_-
sign:nnnnnnnn 91	close_tl 14 , 274 , 290 , 396
\l__siunitx_angle_fill_degrees_-	\l__siunitx_compound_bracket_-
bool 6	open_tl 14 , 272 , 288 , 394
\l__siunitx_angle_fill_minutes_-	\l__siunitx_compound_count_int ..
bool 6 11 , 329 , 352 , 357
\l__siunitx_angle_fill_seconds_-	\l__siunitx_compound_end_tl
bool 6 9 , 331 , 362

\l__siunitx_compound_exp_- bracket_bool	\l__siunitx_compound_separator_- final_tl
18, 254, 284	18, 355
\l__siunitx_compound_exp_- combine_bool	\l__siunitx_compound_separator_- pair_tl
18, 112, 209, 234, 256	18, 320
\l__siunitx_compound_exp_tl	\l__siunitx_compound_separator_- text_bool
8, 148, 264, 270, 285, 291, 295	18, 376
__siunitx_compound_extract_- exp:nN	\l__siunitx_compound_separator_- tl
180	18, 350, 368
__siunitx_compound_extract_- exp:nnnnnnN	\l__siunitx_compound_start_tl
180	9, 330, 347
__siunitx_compound_extract_- exponents:	\l__siunitx_compound_tmp_fp
114, 131	4, 212, 213, 230, 232
__siunitx_compound_extract_- exponents_auxi:w	\l__siunitx_compound_tmp_seq
131	4, 95, 127, 147, 174, 178, 300, 311, 318, 323, 334
__siunitx_compound_extract_- exponents_auxii:nw	\l__siunitx_compound_tmp_tl
131	4, 107, 110, 118, 120, 121, 124, 127, 133, 136, 166, 167, 168, 169, 170, 171, 173, 174, 210, 231, 232, 233, 238, 243
__siunitx_compound_extract_- exponents_auxiii:nnnnnnn	__siunitx_compound_uncert_- bracket:N
131	121, 171, 381
\l__siunitx_compound_first_tl	__siunitx_compound_uncert_- bracket:nnw
7, 110, 119, 125, 134, 150, 210, 212, 233, 238, 243	381
__siunitx_compound_format:n	__siunitx_compound_uncert_- bracket:w
85	381
__siunitx_compound_format:nn	\l__siunitx_compound_unit_bool
85, 262	12, 88, 108, 116, 164, 261
__siunitx_compound_format:nnn	\l__siunitx_compound_unit_- bracket_bool
85	18, 253, 258, 269
__siunitx_compound_format_- combine-exponent:n	\l__siunitx_compound_unit_power_- bool
180	22, 57, 63, 69, 75, 81, 182
__siunitx_compound_format_- combine-exponent:nn	\l__siunitx_compound_unit_- repeat_bool
180	18, 255, 259, 265
__siunitx_compound_format_- combine-exponent_aux:n	\l__siunitx_compound_unit_tl
180	12, 213, 230, 239, 244, 372
__siunitx_compound_format_- extract-exponent:n	__siunitx_compound_unparsed:n
180	103, 162
__siunitx_compound_format_- extract-exponent_aux:n	__siunitx_declare_column:Nnn
180	221
__siunitx_compound_format_- input:n	__siunitx_emulation_non_latin:n
180	674, 704, 706, 708, 714, 719, 724, 734, 750, 810, 814
__siunitx_compound_format_- input:nn	__siunitx_emulation_non_- latin:nnnn
241	674
__siunitx_compound_format_- units:nn	__siunitx_emulation_tmp:w
109, 180	764, 780, 782, 810, 813
__siunitx_compound_parsed:n	__siunitx_list_aux:
129, 162	415
__siunitx_compound_print:N	__siunitx_list_count:n
90, 266, 273, 276, 281	361
__siunitx_compound_print:nnN	__siunitx_list_count:w
281	361
__siunitx_compound_print:nnnN	\l__siunitx_list_exp_tl
281	401, 433
__siunitx_compound_print_aux:n	\l__siunitx_list_units_tl
281	401, 441
__siunitx_compound_print_aux:nn	__siunitx_list_use:nnnnn
281	282, 289, 304, 361
__siunitx_compound_print_- quantity:n	__siunitx_list_use_aux:nnnnn
266, 277, 281	361
__siunitx_compound_print_- separator:n	
281	

_siunitx_list_use_auxi:nw	_siunitx_number_exponent_-
..... 374, 375, 384	engineering:nnNw 710
_siunitx_list_use_auxi:w 361	_siunitx_number_exponent_-
_siunitx_list_use_auxii:nnw .. 361	engineering_0:nnnn 710
_siunitx_list_use_auxiii:nnw . 361	_siunitx_number_exponent_-
_siunitx_load_check: 25	engineering_1:nnnn 710
_siunitx_load_check:n ... 28, 36, 40	_siunitx_number_exponent_-
_siunitx_number_adjust_exp:nn 1483	engineering_2:nnnn 710
_siunitx_number_adjust_-	_siunitx_number_exponent_-
exp:nnnnnnnn 1483	engineering_aux:nnnnnn 710
_siunitx_number_adjust_exp:nNw	_siunitx_number_exponent_-
..... 1483	engineering_uncert:nn 710
\l_siunitx_number_arg_tl ... 50,	_siunitx_number_exponent_-
53, 69, 119, 125, 126, 127, 246, 253,	engineering_uncert_S:nnn 710
256, 272, 287, 299, 373, 508, 514, 522	_siunitx_number_exponent_-
\l_siunitx_number_bracket_-	finalise:n 710, 1243
close_tl 1503, 1621, 1672	_siunitx_number_exponent_-
\l_siunitx_number_bracket_-	fixed:nnnnnn 710
negative_bool 1509, 1647	_siunitx_number_exponent_-
\l_siunitx_number_bracket_open_-	fixed:nnnnnn 710
tl 1503, 1619, 1670	\l_siunitx_number_exponent_-
\l_siunitx_number_comparator_tl	fixed_int 630, 728, 736
..... 70, 252, 255, 356	_siunitx_number_exponent_-
_siunitx_number_digits:NN 694, 958	input:nnnnnn 710
_siunitx_number_digits:Nn ... 958	\l_siunitx_number_exponent_-
_siunitx_number_digits:nn ... 958	mode_tl 630, 715, 719, 1203
_siunitx_number_digits:nnnnnn 958	\l_siunitx_number_exponent_-
_siunitx_number_digits_S:n .. 958	product_tl 1509, 1874
_siunitx_number_digits_-	_siunitx_number_exponent_-
uncert:nn 974, 983	scientific:nnnnnn 710
_siunitx_number_digits_uncert_-	_siunitx_number_exponent_-
S:n 988	scientific:nnnnnn 710
_siunitx_number_drop_exponent:NN	_siunitx_number_exponent_-
..... 692, 993	scientific:nnnw 710
_siunitx_number_drop_exponent:nnnnnn	_siunitx_number_exponent_-
..... 993	shift:nnn 710, 1237
\l_siunitx_number_drop_exponent_-	_siunitx_number_exponent_-
bool 630, 995	shift_down:nnn 710
_siunitx_number_drop_uncertainty:NN	_siunitx_number_exponent_-
..... 690, 1003	shift_down:nnnw 710
_siunitx_number_drop_uncertainty:nnnnnn	_siunitx_number_exponent_-
..... 1003	shift_down:nw 710
\l_siunitx_number_drop_uncertainty_-	_siunitx_number_exponent_-
bool 630, 1005	shift_uncert:nw 710
\l_siunitx_number_drop_zero_-	_siunitx_number_exponent_-
decimal_bool 630, 1467	shift_uncert_S:nnnn 710
\l_siunitx_number_explicit_-	_siunitx_number_exponent_-
plus_bool 29, 318, 517	shift_up:nnn 710
_siunitx_number_exponent:NN ...	_siunitx_number_exponent_-
..... 706, 710	shift_up:nnw 710
\l_siunitx_number_exponent_-	_siunitx_number_exponent_-
base_tl 1509, 1878	shift_up_aux:nnn 710
_siunitx_number_exponent_-	
engineering:nnnnnn 710	

\l__siunitx_number_exponent_tl ..	\c__siunitx_number_normalize_tl .
..... 71,	77
263, 297, 312, 320, 321, 332, 342, 359	__siunitx_number_output:Nn ..
_siunitx_number_exponent_-	1572
uncert:n	_siunitx_number_output:nn ..
710	1572
_siunitx_number_expression:n ..	_siunitx_number_output:nnnnnnn
..... 29, 122 1572
\l__siunitx_number_expression_-	_siunitx_number_output_-
bool	bracket:nn
29, 121	1572
\l__siunitx_number_flex_tl 47, 72,	_siunitx_number_output_-
135, 144, 148, 170, 178, 464, 466, 502	bracket:w
\l__siunitx_number_group_-	1572
decimal_bool	_siunitx_number_output_color:n
1509 1592, 1624
\l__siunitx_number_group_-	_siunitx_number_output_-
integer_bool	comparator:nn
1509	1572
\l__siunitx_number_group_-	_siunitx_number_output_-
minimum_int	decimal:nn
1509, 1701	1572
\l__siunitx_number_group_-	_siunitx_number_output_-
separator_tl 1509, 1730, 1733, 1758	decimal_aux:n
_siunitx_number_if_token_-	1572
auxi:NN	_siunitx_number_output_-
1917	decimal_loop:NNNN
_siunitx_number_if_token_-	1572
auxii:NN	_siunitx_number_output_-
1917	digits:nn
_siunitx_number_if_token_-	1572
auxiii:NN	_siunitx_number_output_end: .
1917	1572
\l__siunitx_number_implicit_-	\l__siunitx_number_output_exp_-
plus_bool	marker_tl
1509, 1641, 1892	1509, 1858, 1885
\l__siunitx_number_input_digit_-	_siunitx_number_output_-
tl	exponent:nnnn
29,	1572
236, 340, 384, 402, 486, 541, 1924	_siunitx_number_output_-
\l__siunitx_number_input_ignore_-	exponent_auxi:nnnn
tl	1572
29, 1926	_siunitx_number_output_-
\l__siunitx_number_input_tl	exponent_auxiii:nn
..... 74, 125, 161	1572
\l__siunitx_number_input_uncert_-	_siunitx_number_output_-
close_tl ... 29, 530, 555, 568, 1922	integer:nnn
\l__siunitx_number_input_uncert_-	1572
open_tl	_siunitx_number_output_-
29, 412, 1927	integer_aux:n
\l__siunitx_number_input_uncert_-	1572
sign_tl	_siunitx_number_output_-
29, 149, 1929	integer_aux_0:n
\l__siunitx_number_min_decimal_-	1572
int	_siunitx_number_output_-
630, 972, 991	integer_aux_1:n
\l__siunitx_number_min_integer_-	1572
int	_siunitx_number_output_-
630, 967	integer_aux_2:n
\l__siunitx_number_negative_-	1572
color_tl	_siunitx_number_output_-
1509, 1628, 1629	integer_first:nnNN
_siunitx_number_normalize_-	1572
aux:nN	_siunitx_number_output_-
77	integer_loop:NNNN
_siunitx_number_normalize_-	1572
aux:NnN	_siunitx_number_output_sign:N
80, 85, 89	1572
_siunitx_number_normalize_-	_siunitx_number_output_sign:nN
minus:N 1572
79, 102	_siunitx_number_output_-
_siunitx_number_normalize_-	sign:nnn
sign:N	1572
77	_siunitx_number_output_sign_-
	brackets:w
	1572

_siunitx_number_output_sign_- color:w	1572	_siunitx_number_parse_exponent_- auxii:nn	259
\l_siunitx_number_output_- uncert_close_tl	1509, 1789	_siunitx_number_parse_exponent_- auxiii:Nw	259
\l_siunitx_number_output_- uncert_open_tl	1509, 1787	_siunitx_number_parse_exponent_- auxiv:nn	259
_siunitx_number_output_uncert_- S:nnnw	1572	_siunitx_number_parse_exponent_- check:N	259
_siunitx_number_output_uncert_- S:nnw	1572	_siunitx_number_parse_exponent_- cleanup:N	259
_siunitx_number_output_uncert_- S_aux:nnn	1572	_siunitx_number_parse_exponent_- cleanup:wN	345, 347
_siunitx_number_output_uncert_- S_aux:nnnw	1797, 1814, 1821, 1828	_siunitx_number_parse_exponent_- zero_test:N	259
_siunitx_number_output_uncert_- S_aux:nnw	1819, 1829	_siunitx_number_parse_finalise:	164, 350
_siunitx_number_output_uncert_- S_compact-marker:nn	1572	_siunitx_number_parse_finalise:nw	350
_siunitx_number_output_uncert_- S_compact:nn	1572	_siunitx_number_parse_loop:	265, 305, 368
_siunitx_number_output_uncert_- S_full:nn	1572	_siunitx_number_parse_loop_- after_decimal:NNN	368
_siunitx_number_output_- uncertainty:nnn	1572	_siunitx_number_parse_loop_- break:wN	368, 529, 531, 540, 563, 573, 600, 622, 628
_siunitx_number_output_- uncertainty_unaligned:n ...	1572	_siunitx_number_parse_loop_- first:N	368
\l_siunitx_number_outputted_tl	8, 21, 24, 26	_siunitx_number_parse_loop_- first:NNN	371, 376, 465
_siunitx_number_parse:nN	108	_siunitx_number_parse_loop_- main:NNNNN	368
_siunitx_number_parse_check:	129, 133	_siunitx_number_parse_loop_- main_decimal:NN	368
_siunitx_number_parse_combine_- uncert:	151, 166	_siunitx_number_parse_loop_- main_digit:NNNNN	368
_siunitx_number_parse_combine_- uncert_auxi:nnnnnnnn	166	_siunitx_number_parse_loop_- main_end:NN	368
_siunitx_number_parse_combine_- uncert_auxii:nnnnn	166	_siunitx_number_parse_loop_- main_sign:NNN	368
_siunitx_number_parse_combine_- uncert_auxiii:nnnnnn	166	_siunitx_number_parse_loop_- main_store:NNN	368, 590
_siunitx_number_parse_combine_- uncert_auxiv:nnnn	166	_siunitx_number_parse_loop_- main_uncert:NNN	368
_siunitx_number_parse_combine_- uncert_auxv:w	166	_siunitx_number_parse_loop_- root_swap:NNwNN	368
_siunitx_number_parse_combine_- uncert_auxvi:w	166	_siunitx_number_parse_sign:	257, 505
_siunitx_number_parse_comparator:	128, 243	_siunitx_number_parse_sign_- aux:Nw	505
_siunitx_number_parse_comparator_- aux:Nw	243	_siunitx_number_parse_uncert:NN	459, 526
_siunitx_number_parse_exponent:	259, 524	_siunitx_number_parse_uncert:NNNN	526
_siunitx_number_parse_exponent_- auxi:w	259		

_siunitx_number_parse_uncert_- after:N	526	_siunitx_number_round_engineering:nn	1027
_siunitx_number_parse_uncert_- auxi:NN	526	_siunitx_number_round_engineering:nnN	1027
_siunitx_number_parse_uncert_- auxii:N	526	_siunitx_number_round_engineering:NNNNn	1027
_siunitx_number_parse_uncert_- auxii:NN	526	_siunitx_number_round_figures:nnnnnnn	1286
_siunitx_number_parse_uncert_- auxiii:N	557, 570, 575	_siunitx_number_round_figures_- count:nnN	1286
_siunitx_number_parse_uncert_- marker:N	526	_siunitx_number_round_figures_- count:nnnN	1286
_siunitx_number_parse_uncert_- marker:nnnN	595, 596	_siunitx_number_round_final:nn	1027
_siunitx_number_parse_uncert_- marker:nNw	601, 603	_siunitx_number_round_final_- decimal:nnw	1027
\l_siunitx_number_parsed_tl	48, 19, 20, 22, 73, 118, 131, 140, 152, 154, 156, 170, 177, 212, 214, 264, 301, 304, 313, 323, 349, 352, 354, 357, 372, 503, 518, 519, 521, 523, 1906, 1907	_siunitx_number_round_final_- integer:nnw	1027
\l_siunitx_number_partial_tl	75, 370, 432, 433, 436, 446, 470, 471, 474, 478, 488, 546, 577, 588, 589, 599, 610, 611	_siunitx_number_round_final_- output:nn	1027
_siunitx_number_process:nnnnnnnNN	685	_siunitx_number_round_final_- shift:nn	1027
_siunitx_number_round:NN	707, 1015	_siunitx_number_round_final_- shift:Nw	1027
_siunitx_number_round:nnn	1027, 1315, 1354, 1364, 1425, 1453, 1459	_siunitx_number_round_fixed:nn	1027
_siunitx_number_round_auxi:nnnN	1027	\l_siunitx_number_round_half_- even_bool	630, 1095, 1113
_siunitx_number_round_auxii:nnnN	1027	_siunitx_number_round_if_- half:N	1260
_siunitx_number_round_auxiii:nnnN	1027	_siunitx_number_round_if_- half:n	1260
_siunitx_number_round_auxiv:nnN	1027	_siunitx_number_round_if_half_- p:n	1097, 1115, 1260
_siunitx_number_round_auxiv:nnnN	1048, 1062, 1072, 1078	_siunitx_number_round_input:nn	1027
_siunitx_number_round_auxv:nnN	1027	\l_siunitx_number_round_min_tl	665, 673, 1382, 1390
_siunitx_number_round_auxvi:nN	1027	\l_siunitx_number_round_mode_tl	630, 1020, 1200, 1246
_siunitx_number_round_auxvi:nnN	1087	_siunitx_number_round_none:nnnnnnn	1015
_siunitx_number_round_auxvi:nnnN	1081, 1104	_siunitx_number_round_pad:nnn	1277, 1320, 1349
_siunitx_number_round_auxvii:nnN	1027	\l_siunitx_number_round_pad_- bool	630, 1282
_siunitx_number_round_auxviii:nnN	1027	_siunitx_number_round_places:nnnnnnn	1327
		_siunitx_number_round_places_- decimal:nn	1327
		_siunitx_number_round_places_- end:nn	1242, 1327
		_siunitx_number_round_places_- finalise:n	1327

_siunitx_number_round_places_- finalise:nnnnn 1327	_siunitx_number_zero_decimal:NN 693 , 1465
_siunitx_number_round_places_- finalise:nnnnnnn 1327	_siunitx_number_zero_decimal:nnnnnnn 1465
_siunitx_number_round_places_- integer:nn 1327	\l_siunitx_number_zero_exponent_- bool 1509 , 1611 , 1855
\l_siunitx_number_round_- precision_int 630 , 1223 , 1290 , 1313 , 1316 , 1321 , 1334 , 1347 , 1350 , 1357 , 1367 , 1408 , 1428	\l_siunitx_number_zero_uncert_- bool 29 , 233 , 582
_siunitx_number_round_scientific:nn 1235	_siunitx_option_deprecated:nn 11 , 76 , 82 , 88 , 108 , 122 , 131 , 137 , 151 , 157 , 222 , 228 , 241 , 247 , 268 , 274 , 286 , 292 , 301 , 310 , 316 , 352 , 358 , 364 , 370 , 516 , 522 , 530 , 545 , 563 , 569 , 577 , 583 , 589 , 603 , 609
_siunitx_number_round_scientificc:nn 1027	_siunitx_option_deprecated:nnn .. 11 , 42 , 50 , 58 , 66 , 97 , 323 , 333 , 341 , 377 , 388 , 396 , 413 , 507 , 554 , 595
_siunitx_number_round_truncate:n 1027	_siunitx_option_removed:n 22 , 36 , 172 , 196 , 216 , 261 , 280 , 282 , 305 , 330 , 348 , 513 , 527
_siunitx_number_round_truncate:nnN 1027	_siunitx_option_table_comparator:nnnnnnn 451
_siunitx_number_round_truncate_- direct:n 1027 , 1462	_siunitx_option_table_comparator:nnnnnnnn 466
_siunitx_number_round_uncertainty:nnn 1404	_siunitx_option_table_figures-decimal:nnnnnnnn 451
_siunitx_number_round_uncertainty:nnnnn 1404	_siunitx_option_table_figures-exponent:nnnnnnnn 451
_siunitx_number_round_uncertainty:nnnnnn 1404	_siunitx_option_table_figures-integer:nnnnnnnn 451
_siunitx_number_round_uncertainty_- aux:nnnnn 1404	_siunitx_option_table_figures-uncertainty:nnnnnnnn 451
_siunitx_number_round_uncertainty_- aux:nnnnnn 1404	_siunitx_option_table_format:n 451 , 489 , 491 , 493 , 495 , 497 , 499 , 501
_siunitx_number_set_round_- min:n 653 , 666	_siunitx_option_table_sign-exponent:nnnnnnnn 451
_siunitx_number_set_round_- min:nnnnnnn 666	_siunitx_option_table_sign-mantissa:nnnnnnnn 451
\l_siunitx_number_tight_bool 1509 , 1663 , 1871	_siunitx_print_aux:nn 60
\l_siunitx_number_tmp_tl 7 , 147 , 150 , 268 , 273 , 279 , 280 , 288 , 289 , 668 , 669	_siunitx_print_convert_- series:n 99
_siunitx_number_token_auxi:NN 1920 , 1933 , 1937	_siunitx_print_extract_- series:Nw 99
_siunitx_number_token_auxii:NN 1936 , 1939	_siunitx_print_math_aux:N 99
_siunitx_number_token_auxiii:NN 1941 , 1944 , 1955	_siunitx_print_math_aux:Nn ... 99
\l_siunitx_number_uncert_mode_- tl 1509 , 1606 , 1776 , 1788	_siunitx_print_math_aux:w 99
\l_siunitx_number_uncert_- separator_tl 1509 , 1786	_siunitx_print_math_auxi:n ... 99
\l_siunitx_number_unity_- mantissa_bool ... 1509 , 1678 , 1868	_siunitx_print_math_auxii:n ... 99
\l_siunitx_number_valid_tl .. 1899	_siunitx_print_math_auxiii:n .. 99
\l_siunitx_number_validate_bool 76 , 158 , 1904	_siunitx_print_math_auxiv:n ... 99
	_siunitx_print_math_auxv:n ... 99
	\l_siunitx_print_math_family_- bool 7 , 149

<code>\l__siunitx_print_math_font_bool</code>	7 , 164
<code>__siunitx_print_math_script:n</code>	99
<code>__siunitx_print_math_sub:n</code>	99
<code>__siunitx_print_math_super:n</code>	99
<code>__siunitx_print_math_text:n</code>	99
<code>__siunitx_print_math_version:nn</code>	99
<code>\l__siunitx_print_math_version-bool</code>	7 , 126
<code>\l__siunitx_print_math_weight-bool</code>	7 , 101
<code>\l__siunitx_print_number_color-tl</code>	7
<code>\l__siunitx_print_number_mode_tl</code>	7
<code>__siunitx_print_replace_font:N</code>	86 , 195 , 217 , 268
<code>\l__siunitx_print_text_family-bool</code>	28 , 239 , 247
<code>\l__siunitx_print_text_font_tl</code>	7 , 252
<code>__siunitx_print_text_fraction:Nnn</code>	235
<code>__siunitx_print_text_replace:N</code>	235
<code>__siunitx_print_text_replace:n</code>	235
<code>__siunitx_print_text_replace:NNn</code>	235
<code>__siunitx_print_text_scripts:</code>	235
<code>__siunitx_print_text_scripts:NnN</code>	235
<code>__siunitx_print_text_scripts-one:Nn</code>	330 , 337 , 368
<code>__siunitx_print_text_scripts-one:NnN</code>	235
<code>__siunitx_print_text_scripts-two:n</code>	235
<code>__siunitx_print_text_scripts-two:nn</code>	235
<code>__siunitx_print_text_scripts-two:NnNn</code>	235
<code>\l__siunitx_print_text_series-bool</code>	30 , 241 , 248
<code>\l__siunitx_print_text_shape-bool</code>	32 , 243 , 249
<code>__siunitx_print_text_sub:n</code>	235
<code>__siunitx_print_text_super:n</code>	235
<code>\l__siunitx_print_tmp_tl</code>	6 , 103 , 105 , 107 , 194 , 195 , 196 , 199 , 202 , 216 , 217 , 218 , 227 , 228 , 230 , 261 , 262 , 263 , 304 , 305 , 306 , 340 , 341 , 343
<code>\l__siunitx_print_unit_color_tl</code>	7
<code>\l__siunitx_print_unit_mode_tl</code>	7
<code>\l__siunitx_print_version_b_tl</code>	48
<code>\l__siunitx_print_version_eb_tl</code>	48
<code>\l__siunitx_print_version_el_tl</code>	48
<code>\l__siunitx_print_version_l_tl</code>	48
<code>\l__siunitx_print_version_m_tl</code>	48
<code>\l__siunitx_print_version_sb_tl</code>	48
<code>\l__siunitx_print_version_sl_tl</code>	48
<code>\l__siunitx_print_version_ub_tl</code>	48
<code>\l__siunitx_print_version_ul_tl</code>	48
<code>\c__siunitx_print_weight_b_tl</code>	97
<code>\c__siunitx_print_weight_c_tl</code>	95
<code>\c__siunitx_print_weight_ecl_tl</code>	95
<code>\c__siunitx_print_weight_ex_tl</code>	95
<code>\c__siunitx_print_weight_l_tl</code>	97
<code>\c__siunitx_print_weight_m_tl</code>	97
<code>\c__siunitx_print_weight_sc_tl</code>	95
<code>\c__siunitx_print_weight_sx_tl</code>	95
<code>\c__siunitx_print_weight_uc_tl</code>	95
<code>\c__siunitx_print_weight_ux_tl</code>	95
<code>\c__siunitx_print_weight_x_tl</code>	95
<code>__siunitx_product_aux:</code>	464
<code>__siunitx_product_aux:n</code>	464
<code>\l__siunitx_product_exp_tl</code>	445 , 488
<code>\l__siunitx_product_phrase_bool</code>	445 , 480 , 493
<code>\l__siunitx_product_phrase_tl</code>	445 , 481
<code>\l__siunitx_product_symbol_tl</code>	445 , 482
<code>\l__siunitx_product_units_tl</code>	445 , 494
<code>\l__siunitx_quantity_bracket-close_tl</code>	5 , 113
<code>\l__siunitx_quantity_bracket-open_tl</code>	5 , 111
<code>\l__siunitx_quantity_break_bool</code>	9 , 145
<code>__siunitx_quantity_extract-exp:nNN</code>	73 , 131
<code>__siunitx_quantity_extract-exp:nnnnnnnnNN</code>	131
<code>__siunitx_quantity_non_latin:n</code>	159 , 181
<code>__siunitx_quantity_non_latin:nnnn</code>	159
<code>\l__siunitx_quantity_number_tl</code>	38 , 53 , 56 , 64 , 66 , 67 , 68 , 72 , 74 , 82 , 85 , 87 , 91
<code>__siunitx_quantity_parsed:nn</code>	40
<code>__siunitx_quantity_parsed-aux:nnnn</code>	40
<code>__siunitx_quantity_parsed-aux:nnnw</code>	40
<code>__siunitx_quantity_parsed-aux:w</code>	40
<code>__siunitx_quantity_parsed-combine-exponent:n</code>	40
<code>__siunitx_quantity_parsed-input:n</code>	40

\l__siunitx_quantity_product_tl .	\l__siunitx_table_before_box ...
..... 9, 144	. 480, 502, 503, 505, 511, 513, 517,
\l__siunitx_quantity_tmp_fp	522, 528, 565, 566, 568, 607, 689, 692
..... 3, 74, 76, 81, 85	\l__siunitx_table_before_dim ...
\l__siunitx_quantity_tmp_tl 3 482, 571, 632, 681, 689
\l__siunitx_quantity_uncert_-	\l__siunitx_table_before_model_-
bracket_bool 9, 106	tl 309, 322, 448, 564
\l__siunitx_quantity_uncert_-	\l__siunitx_table_before_tl
repeat_bool 9, 119 107, 116, 120, 123
\l__siunitx_quantity_unit_tl ...	\l__siunitx_table_carry_dim
..... 38, 46, 47, 54, 483, 598, 601, 701, 756, 764, 776
56, 76, 81, 92, 104, 116, 122, 124, 127	__siunitx_table_center_marker: .
__siunitx_range_aux: 511 263, 425, 548
\l__siunitx_range_exp_tl ... 499, 529	__siunitx_table_cleanup_-
\l__siunitx_range_units_tl . 499, 532	decimal:w 251, 443
__siunitx_symbol_if_replace:Nn . 31	__siunitx_table_collect_begin: .
__siunitx_symbol_if_replace:NnTF 11, 24
..... 31, 51, 56, 61, 94, 114	__siunitx_table_collect_begin:N 126
__siunitx_symbol_non_latin:n ...	__siunitx_table_collect_begin:w 24
..... 10, 34, 69, 75, 89, 108, 123, 137	__siunitx_table_collect_end: 19, 110
__siunitx_symbol_non_latin:nnnn 10	__siunitx_table_collect_end:n . 110
\l__siunitx_symbol_tmpa_tl 3, 34, 35, 36, 39, 75, 76, 77, 80	__siunitx_table_collect_end:w . 110
\l__siunitx_symbol_tmpb_tl 3, 38, 39, 79, 80	__siunitx_table_collect_end_-
\l__siunitx_table_after_box 480,	aux:n 110
508, 510, 514, 521, 524, 536, 597, 610	__siunitx_table_collect_group:n 39
\l__siunitx_table_after_model_tl	__siunitx_table_collect_loop: ..
..... 311, 324, 441, 596 31, 38, 39
\l__siunitx_table_after_tl 107, 118, 125, 165	__siunitx_table_collect_relax:N 39
\l__siunitx_table_align_after_-	__siunitx_table_collect_-
bool 484, 600	search:NnTF 39
__siunitx_table_align_auxi:nn . 201	__siunitx_table_collect_search_-
__siunitx_table_align_auxii:nn 201	aux:Nn 39
\l__siunitx_table_align_before_-	\l__siunitx_table_collect_tl ...
bool 484, 620, 660 23, 27, 47, 64, 113, 115, 134
__siunitx_table_align_center:n 201	__siunitx_table_collect_token:N 39
\l__siunitx_table_align_comparator_-	__siunitx_table_collect_token_-
bool 484, 645	aux:N 39
\l__siunitx_table_align_exponent_-	__siunitx_table_color_check:N ..
bool 484, 738 254, 533
__siunitx_table_align_left:n . 201	__siunitx_table_color_check:Nnw 254
\l__siunitx_table_align_mode_tl .	__siunitx_table_color_check:w ..
..... 297, 393, 397, 498 254, 618
\l__siunitx_table_align_number_-	\l__siunitx_table_column_width_-
tl 297, 472, 605, 785	dim 194, 210
__siunitx_table_align_right:n . 201	\l__siunitx_table_decimal_box ...
\l__siunitx_table_align_text_tl .	. 247, 271, 275, 281, 288, 402, 417,
..... 234, 244, 426, 549	429, 444, 462, 463, 475, 543, 552,
\l__siunitx_table_align_uncertainty_-	609, 700, 704, 753, 755, 760, 771, 773
bool 484, 727	__siunitx_table_direct_begin: ..
\l__siunitx_table_auto_round_- 12, 384
bool 297, 574	__siunitx_table_direct_begin:w 384
	__siunitx_table_direct_end: 20, 384
	__siunitx_table_direct_format: 384

_siunitx_table_direct_format:nnnnnn	_siunitx_table_print_format_-
..... 384	auxiii:w 497
_siunitx_table_direct_format:w 384	_siunitx_table_print_format_-
_siunitx_table_direct_format_-	auxiv:w 497
aux:w 436, 439	_siunitx_table_print_format_-
_siunitx_table_direct_format_-	auxv:w 497
end: 384	_siunitx_table_print_format_-
_siunitx_table_direct_format_-	auxvi:w 497
switch: 384	_siunitx_table_print_format_-
_siunitx_table_direct_marker: 384	auxvii:w 497
_siunitx_table_direct_marker_-	_siunitx_table_print_format_-
end: 384	box:Nn 497
_siunitx_table_direct_marker_-	_siunitx_table_print_marker:nnn
switch: 384 497
_siunitx_table_direct_none: .. 384	_siunitx_table_print_marker:w 497
_siunitx_table_direct_none_-	_siunitx_table_print_marker_-
end: 384	aux:w 497
_siunitx_table_fil:	_siunitx_table_print_none:nnn 497
.... 249, 282, 292, 404, 446, 470,	_siunitx_table_print_text:n ...
516, 525, 603, 625, 655, 668, 691, 761 120, 241, 390
_siunitx_table_fill: 249, 456	_siunitx_table_skip:n
\l_siunitx_table_fixed_width_- 189, 213, 215, 227, 229
bool 194, 209	_siunitx_table_split:nNNN
\l_siunitx_table_format_tl 321, 114, 140, 308
330, 332, 333, 336, 456, 460, 464, 582	_siunitx_table_split_group:NNNn
_siunitx_table_generate_- 140
model:n 312, 325	_siunitx_table_split_loop:NNN 140
_siunitx_table_generate_-	_siunitx_table_split_tidy:N ...
model:nnnnnn 325, 463 146, 147, 178
_siunitx_table_generate_model_-	_siunitx_table_split_tidy:Nn . 178
S:nnn 325	_siunitx_table_split_token:NNNN
_siunitx_table_generate_model_- 140
S:nnw 325	\l_siunitx_table_text_bool
\l_siunitx_table_integer_box 6, 9, 16, 243
..... 247, 268, 277, 286,	\l_siunitx_table_tmp_box
293, 405, 428, 448, 474, 541, 551, 3, 265, 272, 278, 289, 400,
608, 622, 631, 635, 646, 648, 650,	403, 442, 445, 447, 449, 564, 571,
654, 662, 664, 669, 675, 676, 678, 685	596, 598, 619, 623, 634, 643, 651,
\l_siunitx_table_model_tl	665, 683, 699, 703, 724, 725, 726,
..... 310, 312, 321, 341, 435, 590	735, 736, 737, 757, 762, 767, 774, 779
\l_siunitx_table_number_tl	\l_siunitx_table_tmp_dim
..... 107, 117, 119, 124	.. 132, 3, 674, 684, 725, 736, 766, 778
_siunitx_table_print:nnn . 122, 497	\l_siunitx_table_tmp_tl . 3, 434,
_siunitx_table_print_format:nnn	437, 529, 530, 531, 532, 533, 535,
..... 497	572, 585, 587, 588, 592, 595, 788, 789
_siunitx_table_print_format:nnnnnn	_siunitx_tmp:w
..... 497 229, 231, 623, 624, 628, 629
_siunitx_table_print_format_-	\l_siunitx_tmp_tl 43, 113,
after:N 497	114, 123, 124, 186, 636, 637, 647, 648
_siunitx_table_print_format_-	\l_siunitx_unit_autofrac_bool ..
auxi:w 497	513, 520, 527, 534, 541, 548, 567, 945
_siunitx_table_print_format_-	\l_siunitx_unit_bracket_bool ...
auxii:w 497 561, 599, 692, 762, 869, 890

<code>\l_siunitx_unit_bracket_close_tl</code>	562 , 696 , 821
<code>\l_siunitx_unit_bracket_open_tl</code>	562 , 694 , 818
<code>\l_siunitx_unit_combine_exp_fp</code>	135 , 142 , 149 , 156 , 164 , 172 , 573 , 588 , 623
<code>\l_siunitx_unit_current_tl</code>	577 , 600 , 738 , 740 , 756 , 758 , 798 , 800 , 803 , 811 , 849 , 856 , 864 , 916 , 924 , 927
<code>\l_siunitx_unit_denominator_bracket_bool</code>	500 , 888
<code>\l_siunitx_unit_denominator_tl</code>	579 , 584 , 889 , 934 , 975 , 984 , 993 , 1000
<code>\l_siunitx_unit_font_bool</code>	566 , 601 , 853 , 859 , 922 , 929
<code>\l_siunitx_unit_forbid_literal_bool</code>	192 , 500
<code>_siunitx_unit_format:nNN</code>	132
<code>_siunitx_unit_format_aux:</code> ...	132
<code>_siunitx_unit_format_bracket:N</code>	690 , 740 , 758 , 984
<code>_siunitx_unit_format_combine_exp:</code>	589 , 618
<code>_siunitx_unit_format_finalise:</code>	608 , 932
<code>_siunitx_unit_format_finalise_autofrac:</code>	932
<code>_siunitx_unit_format_finalise_fraction:</code>	950 , 956 , 969
<code>_siunitx_unit_format_finalise_fractional:</code>	932
<code>_siunitx_unit_format_finalise_power:</code>	932
<code>_siunitx_unit_format_finalise_symbol:</code>	949 , 959 , 978
<code>_siunitx_unit_format_font:</code>	702 , 815 , 827 , 837 , 868 , 920
<code>_siunitx_unit_format_literal:n</code>	194 , 197 , 211
<code>_siunitx_unit_format_literal_add:n</code>	211
<code>_siunitx_unit_format_literal_auxi:w</code>	211
<code>_siunitx_unit_format_literal_auxii:n</code>	251 , 255
<code>_siunitx_unit_format_literal_auxii:w</code>	211
<code>_siunitx_unit_format_literal_auxiii:w</code>	211
<code>_siunitx_unit_format_literal_auxiv:n</code>	211
<code>_siunitx_unit_format_literal_auxv:nw</code>	211
<code>_siunitx_unit_format_literal_auxvi:nN</code>	211
<code>_siunitx_unit_format_literal_auxvii:nN</code>	211
<code>_siunitx_unit_format_literal_auxvii:nn</code>	211
<code>_siunitx_unit_format_literal_auxviii:nN</code>	211
<code>_siunitx_unit_format_literal_sub:nn</code>	211
<code>_siunitx_unit_format_literal_subscript:</code>	211
<code>_siunitx_unit_format_literal_super:nn</code>	211
<code>_siunitx_unit_format_literal_superscript:</code>	211
<code>_siunitx_unit_format_literal_tilde:</code>	211
<code>_siunitx_unit_format_mass_to_kilogram:</code>	595 , 666
<code>_siunitx_unit_format_multiply:</code>	591 , 650
<code>_siunitx_unit_format_output:</code> ..	606 , 866
<code>_siunitx_unit_format_output_aux:</code>	866
<code>_siunitx_unit_format_output_aux:nn</code>	866
<code>_siunitx_unit_format_output_denominator:</code>	866
<code>_siunitx_unit_format_parsed:</code> ..	189 , 581
<code>_siunitx_unit_format_parsed_aux:n</code>	581
<code>_siunitx_unit_format_power:</code> ..	700
<code>_siunitx_unit_format_power_aux:wTF</code>	700
<code>_siunitx_unit_format_power_negative:</code>	700
<code>_siunitx_unit_format_power_negative_aux:w</code>	700
<code>_siunitx_unit_format_power_positive:</code>	700
<code>_siunitx_unit_format_power_superscript:w</code>	731 , 734
<code>_siunitx_unit_format_power_superscript:</code>	700
<code>_siunitx_unit_format_prefix:</code> ..	764
<code>_siunitx_unit_format_prefix_exp:</code>	764
<code>_siunitx_unit_format_prefix_gram:</code>	764
<code>_siunitx_unit_format_prefix_symbol:</code>	764

_siunitx_unit_format_qualifier:	_siunitx_unit_parse_special:n
..... 804 109, 112, 377
_siunitx_unit_format_qualifier_-	_siunitx_unit_parse_unit:Nn 81, 426
bracket: 804	\l_siunitx_unit_parsed_prop ...
_siunitx_unit_format_qualifier_- 154, 188,
combine: 804	347, 352, 366, 373, 391, 410, 462,
_siunitx_unit_format_qualifier_-	464, 468, 476, 480, 485, 496, 614,
phrase: 804	620, 624, 629, 639, 643, 652, 654,
_siunitx_unit_format_qualifier_-	659, 661, 670, 672, 679, 681, 779, 784
subscript: 804	\l_siunitx_unit_parsing_bool ...
_siunitx_unit_format_special: 847 9, 34, 216, 339, 353, 675, 797
_siunitx_unit_format_unit: .. 861	\l_siunitx_unit_part_tl
\l_siunitx_unit_formatted_tl 470, 472, 473, 474,
161, 131, 181, 201, 232, 240, 241,	481, 577, 615, 704, 717, 720, 722,
305, 312, 325, 327, 585, 897, 898,	732, 773, 788, 794, 795, 803, 811,
943, 944, 958, 960, 964, 965, 966,	816, 820, 828, 832, 838, 843, 851, 864
971, 974, 980, 982, 989, 992, 996, 998	\l_siunitx_unit_per_bool
_siunitx_unit_if_symbolic:n ... 11 157, 347, 354, 438, 449
_siunitx_unit_if_symbolic:nTF .	\l_siunitx_unit_per_symbol_bool
..... 11, 79, 185 514, 521,
_siunitx_unit_literal_power:nn	528, 535, 542, 549, 567, 896, 902, 948
..... 43, 117, 209	\l_siunitx_unit_per_symbol_tl ..
_siunitx_unit_literal_special:nN 500, 983
..... 111, 210	\l_siunitx_unit_position_int ...
\l_siunitx_unit_mass_kilogram_- 161,
bool 122, 594, 775	347, 356, 359, 379, 386, 397, 409,
\c_siunitx_unit_math_subscript_-	413, 423, 428, 432, 436, 441, 455,
tl 7, 221, 245, 288, 314, 840	495, 583, 587, 597, 603, 613, 778, 783
\l_siunitx_unit_multiple_fp 136,	\l_siunitx_unit_powers_positive_-
143, 150, 157, 165, 173, 576, 590, 657	bool 515,
_siunitx_unit_non_latin:n 1004,	522, 529, 536, 543, 550, 567, 715, 936
1040, 1056, 1067, 1090, 1091, 1092	\l_siunitx_unit_prefix_exp_bool
_siunitx_unit_non_latin:nnnn 1004 134,
\l_siunitx_unit_numerator_bool .	141, 148, 155, 163, 171, 574, 593, 766
..... 168, 571, 602, 714, 873	\l_siunitx_unit_prefix_fp
\l_siunitx_unit_options_bool 182, 203, 575, 586, 684, 685, 787
..... 88, 91, 102	\l_siunitx_unit_prefixes_-
_siunitx_unit_parse:n 187, 350	forward_prop 49, 626, 772
_siunitx_unit_parse_add:nnnn ..	\l_siunitx_unit_prefixes_-
..... 363,	reverse_prop 49, 641
380, 394, 412, 422, 431, 435, 440, 454	\l_siunitx_unit_product_tl
\l_siunitx_unit_parse_bool 183, 500 122, 252, 881, 892, 999
_siunitx_unit_parse_finalise: .	\l_siunitx_unit_qualifier_mode_-
..... 361, 490	tl 500, 572, 809
_siunitx_unit_parse_finalise:n	\l_siunitx_unit_qualifier_-
..... 360, 459	phrase_tl 500, 830
_siunitx_unit_parse_per: . 106, 445	\l_siunitx_unit_separator_tl .. 211
_siunitx_unit_parse_power:nnN .	_siunitx_unit_set_symbolic:Nnn
..... 44, 47, 118, 121, 377 23, 42, 45, 51, 66, 76, 104
_siunitx_unit_parse_prefix:Nn .	_siunitx_unit_set_symbolic:Nnnn
..... 53, 377 23
_siunitx_unit_parse_qualifier:nn	_siunitx_unit_set_symbolic:Npnn
..... 68, 115, 377 23, 107, 110, 113, 116, 119

<code>\l_siunitx_unit_sticky_per_bool</code>	<code>\sys_if_engine_xetex_p:</code>
..... 157, 343, 447	12, 107, 122, 161, 676, 733, 749, 1006
<code>\l_siunitx_unit_test_bool</code>	
..... 10, 14, 32, 355	
<code>\l_siunitx_unit_tmp_fp</code> .. 4, 137,	
151, 158, 174, 622, 637, 656, 658, 662	
<code>\l_siunitx_unit_tmp_int</code> . 4, 379, 381	
<code>\l_siunitx_unit_tmp_tl</code>	
..... 4, 15, 17, 217, 218, 220,	
230, 231, 233, 236, 365, 367, 374,	
385, 391, 408, 410, 461, 462, 465,	
466, 469, 477, 481, 486, 494, 496,	
612, 615, 620, 621, 623, 624, 627,	
629, 631, 632, 635, 636, 637, 638,	
642, 643, 646, 654, 655, 657, 676,	
678, 680, 682, 683, 685, 745, 759,	
777, 779, 782, 785, 786, 788, 958, 963	
<code>\l_siunitx_unit_total_int</code>	
..... 580, 583, 597, 668	
<code>\l_siunitx_unit_two_part_bool</code> ..	
516, 523, 530, 537, 544, 551, 567, 885	
skip commands:	
<code>\skip_horizontal:N</code>	
250	
<code>\skip_horizontal:n</code> 191, 225, 601	
<code>\c_zero_skip</code>	
192	
<code>\sp</code>	
165	
<code>\space</code>	
48, 50, 54, 56,	
60, 62, 540, 542, 546, 548, 551, 557,	
559, 563, 565, 568, 574, 576, 583, 585	
space-before-unit	
183	
<code>\SplitArgument</code>	
100	
<code>\SplitList</code>	
128, 137, 146, 156, 655	
<code>\square</code>	
141, 1094	
<code>\squared</code>	
142, 1094	
<code>\steradian</code>	
141, 1066	
sticky-per	
144	
str commands:	
<code>\str_case_e:nnTF</code>	
151	
<code>\str_if_eq:nnTF</code>	
39,	
69, 80, 130, 132, 185, 236, 263,	
295, 299, 333, 424, 609, 719, 722,	
751, 768, 813, 823, 853, 955, 1126,	
1200, 1246, 1385, 1398, 1411, 1476,	
1501, 1645, 1687, 1730, 1776, 1947	
<code>\str_if_eq_p:nn</code>	
319, 444,	
544, 676, 678, 700, 702, 1376, 1378,	
1606, 1612, 1627, 1679, 1856, 1869	
<code>\str_range:nnn</code>	
376, 378	
<code>\symoperators</code>	
91, 173	
sys commands:	
<code>\sys_if_engine_luatex_p:</code>	
11, 106, 121, 160, 675, 732, 748, 1005	
<code>\sys_if_engine_xetex:TF</code>	
295	
	T
	<code>table-align-comparator</code>
	114
	<code>table-align-exponent</code>
	114
	<code>table-align-text-after</code>
	114
	<code>table-align-text-before</code>
	114
	<code>table-align-uncertainty</code>
	114
	<code>table-alignment</code>
	115
	<code>table-alignment-mode</code>
	115
	<code>table-auto-round</code>
	115
	<code>table-column-width</code>
	115
	<code>table-fixed-width</code>
	115
	<code>table-format</code>
	115
	<code>table-number-alignment</code>
	115
	<code>table-text-alignment</code>
	115
	<code>\tablenum</code>
	198
	<code>\tabularnewline</code>
	55, 57, 84, 86
	<code>\tebi</code>
	182, 2
	<code>\tera</code>
	140, 13, 57, 1044
	<code>\tesla</code>
	141, 1066
	<code>\TeV</code>
	177, 42
	TeX and L^AT_EX 2_ε commands:
	<code>\@ifl@t@r</code>
	12, 42
	<code>\@ifpackagelater</code>
	1
	<code>\@ifpackageloaded</code>
	30, 42, 44, 63, 75, 80, 83,
	131, 220, 223, 246, 299, 320, 701, 808
	<code>\@ifundefined</code>
	9
	<code>\@maybe@unskip</code>
	81
	<code>\@temptokena</code>
	234, 236
	<code>\f@family</code>
	151
	<code>\f@series</code>
	104
	<code>\FB@fg</code>
	333
	<code>\m@th</code>
	354, 392
	<code>\math@version</code>
	132
	<code>\NC@do</code>
	229, 230
	<code>\NC@find</code>
	239
	<code>\NC@list</code>
	7, 230, 231
	<code>\newcommand</code>
	185
	<code>\protected@edef</code>
	151, 15, 35, 76, 119, 133, 230
	<code>\sf@size</code>
	367
	<code>\tab@setcr</code>
	82
	<code>\use@mathgroup</code>
	183, 185
	<code>\z@</code>
	367
	tex commands:
	<code>\tex_cr:D</code>
	32
	<code>\tex_hfil:D</code>
	247, 249
	<code>\tex_hfill:D</code>
	250
	<code>\tex_hss:D</code>
	267, 269
	<code>\tex_kern:D</code>
	192

`\text` 91, 101, 49, 55, 61,
136, 237, 541, 547, 558, 564, 575,
584, 787, 789, 792, 795, 798, 805, 814
`text-family-to-math` 93
`text-font-command` 93
`text-series-to-math` 93
`\textcenteredperiod` 91
`\textcolor` .. 91–93, 136, 142, 70, 111, 112
`\textminus` 91, 276
`\textmu` 124, 735
`\textohm` 109, 751
`\textperiodcentered` 280
`\textpm` 91, 272
`\textsubscript` 91, 316, 347
`\textsuperscript` 91, 321
`\texttimes` 91, 278
`\the` 231, 236
`\THz` 176, 8
`tight-spacing` 42
`\times` ... 38, 14, 20, 26, 32, 277, 578, 1973
tl commands:
`\c_empty_tl` 55, 56, 1740
`\c_space_tl`
.... 182, 273, 327, 352, 357, 396, 404
`\tl_clear:N` 27,
101, 112, 118, 142, 143, 144, 149,
152, 154, 177, 178, 181, 232, 255,
264, 301, 313, 323, 332, 349, 370,
478, 502, 503, 523, 584, 585, 600, 688
`\tl_clear_new:N` 83
`\tl_const:Nn` 7, 91, 96, 98
`\tl_count:N` 599, 611
`\tl_count:n`
. 113, 175, 182, 185, 366, 599, 607,
680, 703, 743, 980, 981, 1211, 1347,
1350, 1357, 1367, 1379, 1428, 1437,
1478, 1701, 1712, 1782, 1840, 1848
`\tl_head:n` 99, 214,
269, 349, 1091, 1109, 1262, 1411, 1768
`\tl_head:w` 913, 937
`\tl_if_blank:nTF` 3, 17, 44,
103, 142, 142, 158, 184, 192, 211,
300, 322, 347, 348, 350, 352, 355,
357, 363, 385, 403, 482, 559, 626,
711, 733, 736, 745, 791, 803, 809,
831, 847, 907, 921, 931, 944, 974,
1046, 1156, 1288, 1329, 1444, 1451,
1633, 1639, 1657, 1685, 1765, 1890
`\tl_if_blank_p:n` 4, 139, 143,
218, 219, 390, 391, 1407, 1608, 1839
`\tl_if_empty:NTF` 68,
88, 105, 105, 109, 119, 125, 127,
135, 156, 157, 165, 170, 180, 233,
256, 261, 264, 304, 312, 327, 333,
352, 470, 522, 577, 687, 914, 934,
943, 989, 1382, 1487, 1582, 1858, 1907
`\tl_if_empty:nTF` ... 84, 329, 709, 1587
`\tl_if_empty_p:N`
..... 270, 285, 432, 889, 897, 1628
`\tl_if_eq:nnTF` 429
`\tl_if_exist:NTF` 95
`\tl_if_head_eq_charcode:nNTF` .. 748
`\tl_if_head_eq_meaning:nNTF` 253, 258
`\tl_if_head_is_N_type:nTF` 275
`\tl_if_in:NnTF` ... 5, 56, 149, 250,
309, 340, 384, 390, 402, 405, 412,
417, 486, 512, 530, 541, 555, 561, 568
`\tl_if_novalue:nTF` 212, 215
`\tl_if_single_token:nTF` 93
`\tl_map_function:nN` 103, 129
`\tl_map_inline:Nn` .. 260, 270, 407, 451
`\tl_map_inline:nn` 54
`\tl_new:N` 3, 3, 3,
4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8,
8, 9, 9, 9, 10, 13, 14, 15, 23, 28, 38,
39, 43, 68, 69, 70, 71, 72, 73, 74, 75,
87, 88, 89, 90, 107, 108, 109, 131,
240, 319, 320, 321, 322, 323, 324,
336, 401, 402, 445, 447, 499, 500,
562, 563, 572, 577, 578, 579, 663,
664, 665, 1503, 1504, 1508, 1571, 1899
`\tl_put_right:Nn` 47,
57, 64, 158, 159, 168, 171, 172, 175,
236, 305, 312, 325, 342, 381, 392,
434, 446, 472, 488, 546, 579, 811, 863
`\tl_replace_all:Nnn`
..... 3, 5, 5, 6, 88, 90,
104, 196, 199, 218, 220, 228, 272, 292
`\tl_reverse:n` .. 798, 1183, 1184, 1190
`\tl_set:Nn`
.... 61, 151, 170, 7, 8, 16, 17, 21,
24, 26, 34, 51, 53, 53, 66, 75, 82, 85,
103, 111, 118, 120, 124, 127, 131,
132, 133, 136, 146, 147, 148, 161,
163, 168, 170, 186, 194, 200, 212,
216, 217, 227, 231, 238, 240, 252,
253, 261, 263, 264, 268, 297, 299,
304, 313, 320, 321, 330, 331, 332,
332, 340, 341, 354, 365, 385, 408,
433, 434, 441, 448, 456, 461, 464,
466, 471, 472, 494, 514, 518, 519,
521, 531, 564, 565, 587, 588, 588,
612, 621, 632, 635, 636, 655, 673,
676, 678, 683, 712, 717, 721, 738,
745, 756, 777, 782, 786, 794, 798,
800, 816, 828, 838, 849, 898, 911,
924, 944, 958, 960, 960, 971, 980,
996, 997, 1007, 1017, 1469, 1505, 1506

<code>\tl_set_eq:NN</code>	18, 20, 44, 125, 159, 252, 303, 317, 407, 413, 452, 462, 505, 509, 557, 640, 656, 803, 991, 1010, 1565
<code>\tl_tail:n</code>	100, 914, 938
<code>\tl_use:N</code>	75, 122, 144, 202, 218, 252, 263, 297, 299, 306, 310
<code>\l_tmpa_tl</code>	136, 137
token commands:	
<code>\c_math_toggle_token</code> ...	406, 415, 418, 423, 450, 460, 464, 469, 478, 479
<code>\token_if_eq_charcode_p:NN</code> ...	516
<code>\token_if_eq_meaning:NNTF</code>	103, 284, 287, 1499
<code>\token_to_str:N</code>	30, 83, 85, 85, 95, 98, 105, 180, 219, 221, 229, 338, 341, 370, 403, 417, 761, 767, 771, 774, 776, 1109, 1112
<code>\tonne</code>	141, 1077
<code>\tothe</code>	142, 116
<code>\TrimSpaces</code>	91, 117, 128, 146, 156, 164, 655, 664
<code>\ttdefault</code>	155
U	
<code>\uA</code>	176, 2
<code>\uF</code>	178, 70
<code>\uG</code>	176, 35
<code>\uH</code>	75
<code>\uJ</code>	177, 42
<code>\uL</code>	177, 27
<code>\uI</code>	177, 185, 27, 48
<code>\uM</code>	176, 60
<code>\uMol</code>	177, 14
<code>uncertainty-mode</code>	43
<code>uncertainty-separator</code>	43
<code>\unit</code>	200, 100, 279, 313
<code>unit-color</code>	93
<code>unit-font-command</code>	144
<code>unit-mode</code>	93
<code>unit-optional-argument</code>	183
<code>\unskip</code>	54, 83
<code>\upOmega</code>	99, 100, 741, 742
<code>\upshape</code>	92
<code>\us</code>	176, 89
use commands:	
<code>\use:N</code>	65, 72, 184, 188, 244, 349, 393, 397, 426, 472, 498, 549, 605, 616, 785, 806, 859, 870, 883, 946, 986, 1202, 1703, 1709, 1768, 1788
<code>\use:n</code>	69, 70, 92, 135, 137, 174, 180, 185, 186, 211, 258, 275, 284, 368, 854, 1689, 1732, 1873
<code>\use_i:nn</code>	77, 859, 946
<code>\use_i:nnnn</code>	148
<code>\use_i_delimit_by_q_recursion_-</code> stop:nw	188, 1230, 1275, 1949, 1951
<code>\use_i_delimit_by_q_stop:nw</code> ...	104
<code>\use_ii:nn</code>	1415
<code>\use_iv:nnnn</code>	140, 144
<code>\use_none:n</code>	482, 749, 1264, 1750
<code>\use_none:nn</code>	1754
<code>\use_none:nnnn</code>	88
<code>use-xspace</code>	183
<code>\uV</code>	177, 21
<code>\uW</code>	177, 42
V	
<code>\V</code>	177, 21
<code>\volt</code>	141, 21, 22, 23, 24, 25, 26, 1066
W	
<code>\W</code>	177, 42
<code>\watt</code> ..	141, 42, 43, 44, 45, 46, 47, 58, 1066
<code>\weber</code>	141, 1066
X	
<code>\xspace</code>	88
Y	
<code>\yobi</code>	182, 2
<code>\yocto</code>	140, 1034
<code>\yotta</code>	140, 1044
Z	
<code>\zebi</code>	182, 2
<code>\zepto</code>	140, 1034
<code>\zetta</code>	140, 1044