siunitx — A comprehensive (SI) units package*

Joseph Wright†

Released 2008/09/03

Abstract

Typesetting values with units requires care to ensure that the combined mathematical meaning of the value plus unit combination is clear. In particular, the SI units system lays down a consistent set of units with rules on how these are to be used. However, different countries and publishers have differing conventions on the exact appearance of numbers (and units).

The siunitx provides a set of tools for authors to typeset numbers and units in a consistent way. The package has an extended set of configuration options which make it possible to follow varying typographic conventions with the same input syntax. The package includes automated processing of numbers and units, and the ability to control tabular alignment of numbers.

A number of LaTeX packages have been developed in the past for formatting units: SIunits, SIstyle, unitsdef, units, fancyunits and fancynum. Support for users of all of these packages is available as emulation modules in siunitx. In addition, siunitx can carry out many of the functions of the dcolumn, rccol and numprint packages.

Contents

*This file describes version v1.0l, last revised 2008/09/03.

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

Part I
Introduction

The correct application of units of measurement is very important in technical applications. For this reason, carefully-crafted definitions of a coherent units system have been laid down by the *Conférence Générale des Poids et Mesures*[1] (CGPM): this has resulted in the *Système International d'Unités*[2] (SI). At the same time, typographic conventions for correctly displaying both numbers and units exist to ensure that no loss of meaning occurs in printed matter.

siunitx aims to provide a unified method for LaTeX users to typeset units and values correctly and easily. The design philosophy of siunitx is to follow the agreed rules by default, but to allow variation through option settings. In this way, users can use siunitx to follow the requirements of publishers, co-authors, universities, *etc.* without needing to alter the input at all.

siunitx is intended as a complete replacement for SIunits, SIstyle, unitsdef, units, fancyunits and fancynum. As such, emulation modes are provided for all of these packages. Where possible, conventions from the existing solutions have been used here. For example, the macros `\num`, `\ang` and `\SI` act in a very similar fashion to those in existing packages.

[1] General Conference on Weights and Measures.
[2] International System of Units.

Part II
Using the siunitx package

1 For the impatient

siunitx provides the user macros:

- `\SI[`⟨*options*⟩`]{`⟨*value*⟩`}[`⟨*pre-unit*⟩`]{`⟨*unit*⟩`}`
- `\si[`⟨*options*⟩`]{`⟨*unit*⟩`}`
- `\num[`⟨*options*⟩`]{`⟨*number*⟩`}`
- `\ang[`⟨*options*⟩`]{`⟨*angle*⟩`}`
- `\sisetup{`⟨*options*⟩`}`

plus the `S` and `s` column types for decimal alignments and units in tables. These macros are designed for typesetting units and values with control of appearance and with intelligent processing.

By default, all text is typeset in the current upright, serif maths font. This can be changes by setting the appropriate package options: `obeyall` will use the current font for typesetting.

The package includes a "unit processor", which allows the use of named units or literal values. Named units are processed to correctly include powers.

10 g
23.4 g cm^3
1×10^{34}
1°2′3″
16.7 m s^{-1}
30 × 10^3 Hz
1.2 mm × 3.56 mm × 9.2 mm
−4.5 cm
J mol^{-1} K^{-1}
$\frac{\text{J}}{\text{mol K}}$
1.2346
9.8000

Heading
1.3
134.2
3.56
74.7

```
\SI{10}{\gram}\\
\SI{23.4}{g.cm^3}\\
\num{1e34}\\
\ang{1;2;3}\\
\emph{\SI{16,7}{\metre\per\second}}\\
\textbf{\SI{30e3}{\Hz}}\\
\SI{1.2 x 3.56 x 9.2}{\milli\metre}\\
\sisetup{obeyall}
\textbf{\SI{-4.5}{\cm}}\\
\si{\joule\per\mole\per\kelvin}\\
\si[per=frac]{\joule\per\mole\per\kelvin}\\
\num[dp=4]{1.23456}\\
\num[dp=4]{9.8}\\
\begin{tabular}{S[tabformat=3.2]}
  \toprule
  {Heading}\\
  \midrule
  1.3 \\
  134.2 \\
  3.56 \\
  74,7 \\
  \bottomrule
\end{tabular}
```

2 Requirements

siunitx requires a reasonably up to date TEX system. The package requires ε-TEX-extensions, which should be available on most systems.[3] The following packages are also needed:

- array and xspace: from the tools bundle, which should be available to everyone;
- xkeyval: this processes the option handling, and needs to be at least v2.5;
- amstext: from the $\mathcal{AMS}$TEX support bundle (the $\mathcal{AMS}$ fonts are also needed to provide the default upright μ).

Hopefully most people using the package will have access to all of those items.

To use the `fraction=sfrac` option, the xfrac package is needed. This needs various experimental LATEX3 packages. As a result, siunitx does not load xfrac. If you want to use `fraction=sfrac`, *you* need to load xfrac in your preamble before siunitx.[4] If the package is not loaded, `fraction=sfrac` falls back on a nicefrac-like method. The interested user should look at the xfrac documentation for reasons this might not be ideal.[5]

3 Loading the package

siunitx is loaded by the usual LATEX method.

```
\usepackage[<options>]{siunitx}
```

As is shown in the example, the package can be loaded with one or more options, using the key–value system. The full range of package options are described in Section 10; some options are described in the along with the appropriate user macros. Most of the user macros accept the same key–value settings as an optional argument.

4 Numbers

`\num` Numbers are automatically formatted by the `\num` macro. This takes one optional and one mandatory argument: `\num[`⟨*options*⟩`]{`⟨*number*⟩`}`. The contents of ⟨*number*⟩ are automatically formatted, in a similar method to that used by numprint. The formatter removes "hard" spaces (`\,` and `~`), automatically identifies exponents (by default marked using `e` or `d`) and adds the appropriate spacing of large numbers. A leading zero is added before a decimal marker, if needed: both "`.`" and "`,`" are recognised as decimal marker.

1 123 1234 12 345
0.1 0.123 0.1234 0.123 45
1×10^{10} 3.45×10^{-4} -10^{10}

```
\num{1}     \num{123}      \num{1234}   \num{12345}\\
\num{0.1}   \num{0.123}    \num{0,1234} \num{.12345}\\
\num{1e10}  \num{3.45d-4}  \num{-e10}
```

[3] If you have an old LATEX try "`elatex`" rather than "`latex`".

[4] This document has been compiled in this way. You have to load xfrac first as otherwise very nasty things happen with xkeyval. MiKTEX users should note that the packaged versions of expl3, template and xparse will not work with xfrac: download copies from CTAN!

[5] On the other hand, some fractional units will look really bad with `\sfrac`. Use this option with caution.

Various error-checking systems are built into the package, so that if ⟨*number*⟩ does not contain any numeric characters, a warning is issued. Isolated signs are also detected. The package recognises `(` and `)` as "extra" characters, which can be used to indicate the error in a number.[6] The `seperr` causes this data to be given as a separate error value. If the number also contains an exponent, then brackets are re-added after the separation to ensure that meaning is not lost.

$1.234(5) = 1.234 \pm 0.005$
$1.234(5) \times 10^{6} = (1.234 \pm 0.005) \times 10^{6}$

```
$\num{1.234(5)} = \num[seperr]{1.234(5)}$\\
$\num{1.234(5)e6} = \num[seperr]{1.234(5)e6}$
```

The same applies to the unit and value macro `\SI`, described later, for example the rest mass of an electron [1]:

$m_\mathrm{e} = 9.109\,389\,7(54) \times 10^{-31}\,\mathrm{kg}$
$m_\mathrm{e} = (9.109\,389\,7 \pm 0.000\,005\,4) \times 10^{-31}\,\mathrm{kg}$

```
$ m_{\mathrm{e}}
  = \SI{9.1093897(54)e-31}{\kg} $\\
$ m_{\mathrm{e}}
  = \SI[seperr]{9.1093897(54)e-31}{\kg} $
```

A number of effects are available as options. These are fully explained in Section 10. Some of the more useful options are illustrated here. By default, the output of the package is typeset in maths mode. However, the use of the current text font can be forced.[7]

1 234 567 890 1 234 567 890

```
\num{1234567890} \num[mode=text]{1234567890}
```

siunitx can automatically add zeros and signs to numbers. This can be altered as desired.

1 1.0
2 +2
3×10^{4} $+3 \times 10^{4}$ $+3 \times 10^{+4}$
0.5 .5

```
\num{1.}   \num[padnumber=all]{1.}\\
\num{2}    \num[addsign=all]{2}\\
\num{3e4}  \num[addsign=mant]{3e4} \num[addsign=all]{3e4}\\
\num{.5}   \num[padnumber=none]{.5}
```

The separation of digits can be turned on and off, and the output changed.

1234 1 234
12 345 12,345
12345

```
\num{1234}  \num[sepfour=true]{1234}\\
\num{12345} \num[digitsep=comma]{12345}\\
\num[digitsep=none]{12345}
```

The formatting of exponents is also customisable.

1×10^{10} $1 \cdot 10^{10}$
2×10^{20} 2×5^{20}
3×10^{30} $3{\times}10^{30}$

```
\num{1e10} \num[expproduct=cdot]{1e10}\\
\num{2e20} \num[expbase=5]{2e20}\\
\num{3e30} \num[expproduct=tighttimes]{3e30}
```

siunitx can automatically add colour to negative numbers, which is often useful for highlighting purposes. This is turned on with the `colourneg` option; the colour used is set by `negcolour`. Both of these are available with the US spellings: `colorneg` and `negcolor`.

-1
-2
3×10^{-3}
-4

```
\num{-1}\\
\sisetup{colourneg} \num{-2}\\
\num{3e-3}\\
\num[negcolour=blue]{-4}
```

[6] This is common in chemical crystallography, for example.

[7] This document is typeset using lowercase numbers in text mode, which emphasises the effect here.

siunitx can automatically zero-fill and round to a fixed number of decimal places. This is controlled by two options `fixdp` and `dp`. The later is an integer which specifies how many places to fix to; setting this option automatically sets `fixdp` to `true`. The place-fixing system will only alter pure numbers: for example, any error component will result in the input being left unchanged.

```
\num{1.23456}\\
\sisetup{dp=2}
\num{1.23456}\\
\num{9.8}\\
\num{-10.432}\\
\num{44.3221(2)}
```

1.234 56
1.23
9.80
−10.43
44.3221(2)

5 Tabular material

5.1 Aligning numbers

Centring numbers in tabular content is handled by a new column type, the `S` column. This is based closely on the dcolumn method for centring numbers in columns, but adds the functionality of the `\num` macro.[8]

By default, the decimal marker of the number is placed at the centre of the column, which then resizes to accommodate the width of the contents (Table 1). This behaviour is set by the `tabnumalign=centredecimal` option. By setting the `tabnumalign` option to `centre`, the centre of the space reserved for the number is placed at the centre of the column. The space reserved is stored in `tabformat`, which is of the form ⟨*before*⟩⟨*dec*⟩⟨*after*⟩, where ⟨*before*⟩ is the number of characters before the decimal marker and ⟨*after*⟩ is the number after. Thus in the example, `tabformat=2.4` provides space for two digits before the decimal marker and four after. `tabnumalign` can also be set to `left` and `right`, with the expected results.

```
\begin{table}
  \caption{Behaviour of \texttt{S} column type
    \label{tab:default}}
  \centering
  \begin{tabular}{%
    S%
    S[tabnumalign=centre,tabformat=2.4]%
    S[tabnumalign=right,tabformat=2.4]%
    S[tabnumalign=centre,tabformat=2.4,decimalsymbol=comma]}
  \toprule
  {Some Values} & {Some Values} & {Some Values} & {Some Values} \\
  \midrule
   2.3456  &  2.3456  &  2.3456 &  2.3456 \\
  34.2345  & 34.2345  & 34.2345 & 34.2345 \\
  56.7835  & 56.7835  & 56.7835 & 56.7835 \\
  90.473   & 90.473   & 90.473  & 90.473  \\
  \bottomrule
  \end{tabular}
\end{table}
```

[8] The approach used is actually a combination of dcolumn for centring the material and numprint for processing it. It will therefore give rather different results than the `n` and `N` column types in numprint.

Table 1: Behaviour of `S` column type

Some Values	Some Values	Some Values	Some Values
2.3456	2.3456	2.3456	2,3456
34.2345	34.2345	34.2345	34,2345
56.7835	56.7835	56.7835	56,7835
90.473	90.473	90.473	90,473

The `tabformat` setting can also be used to reserve space for numbers containing exponents. This is given in the same format as above, but with a mantissa and exponent part (Table 2). Notice that this is designed to expect that numbers will contain a mantissa. Exponents can either be aligned so that the "×" symbols match up vertically, or the exponent part can be allowed to move across as needed. Space for signs is added by using any sign in the `tabformat`, so for example `tabformat=+2.2` and `tabformat=-2.2` have exactly the same effect. Setting `tabformat` will automatically switch from `tabnumalign` from `centredecimal` to `centre`, if the former is currently set. In other cases, the current alignment option is retained.

```
\begin{table}
  \caption{Exponents in tables
    \label{tab:exptab}}
  \centering
  \begin{tabular}{%
    S[tabnumalign=right,tabformat=2.2e2]%
    S[tabnumalign=centre,tabformat=2.2e1.1]%
    S[tabnumalign=centre,tabformat=2.2e1.1,tabalignexp=false]%
    S[tabnumalign=centre,tabformat=+2.2]}
  \toprule
      {Longer values}
    & {Longer values}
    & {Longer values}
    & {Values} \\
  \midrule
  2.3e1    &  2.34e1  &  2.34e1  & +2.31 \\
  34.23e45 & 34.23e45 & 34.23e45 & 34.23 \\
  56.78    & 56.78    & 56.78    & -56.78 \\
  1.0e34   & 1.0e34   & 1.0e34   & +-1.0 \\
  \bottomrule
  \end{tabular}
\end{table}
```

Data not to be processed as a number should be protected by wrapping it in braces: this is most likely to be true for column headers (again as illustrated). By default, the contents of non-numeric cells are centred. This can be altered by setting `tabtextalign`, which can be set to `left`, `right` or `centre`. The use of digit separators in table columns is accounted for: extra space is reserved if digit separators will be added.

Table 2: Exponents in tables

Longer values	Longer values	Longer values	Values
2.3×10^{1}	2.34×10^{1}	2.34×10^{1}	2.31
34.23×10^{45}	34.23×10^{45}	34.23×10^{45}	34.23
56.78	56.78	56.78	−56.78
1.0×10^{34}	1.0×10^{34}	1.0×10^{34}	±1.0

Table 3: Number and units in tables

Value	Unit
2.16×10^{-5}	$\mathrm{m^2\,s^{-1}}$
2.83×10^{-6}	$\mathrm{m^2\,s^{-1}}$
7.39×10^{3}	$\mathrm{Pa\,m^3\,mol^{-1}}$
1.0×10^{5}	Pa

5.2 Columns of units

As a complement to the `S` column, siunitx also provides a second column type, `s`. This is intended for producing columns of units. The letters chosen are intended to be similar to `\SI` and `\si`, respectively. The alignment of material in `s` columns is governed by the `tabunitalign` option.

```
\begin{table}
\centering
\caption{Number and units in tables}
\label{tab:num-unit}
  \begin{tabular}{%
    S[tabformat=1.2e-1,tabnumalign=centre]%
    s[tabunitalign=left]}
  \toprule
    {Value} & \multicolumn{1}{c}{Unit} \\
  \midrule
    2.16e-5 & \metre\squared\per\second \\
    2.83e-6 & \metre\squared\per\second \\
    7.39e3  & \pascal\cubic\metre\per\mole \\
    1.0e5   & \pascal \\
  \bottomrule
  \end{tabular}
\end{table}
```

As the `\si` macro can take literal or macro input, the `s` column cannot validate the input. *Everything* in an `s` column is therefore passed to the `\si` macro for processing. To prevent this, you have to use `\multicolumn`, as is shown in Table 4. Notice that the braces do not prevent processing and colouring of the cell contents.

```
\begin{table}
\centering
\caption{The \texttt{s} column processes everything}
```

Table 4: The `s` column processes everything

Unit	Unit
m^3	m^3
kg	kg

```
\label{tab:s-limits}
  \begin{tabular}{%
    s[colourall,colour=orange]%
    s[colourall,colour=orange]}
  \toprule
    {Unit}     & \multicolumn{1}{c}{Unit}\\
  \midrule
    {m^3}      & \multicolumn{1}{c}{\si{m^3}} \\
    \kilogram & \kilogram \\
  \bottomrule
  \end{tabular}
\end{table}
```

6 Angles

`\ang` Angles can be typeset using the `\ang` command. This takes two arguments, `\ang[`⟨*options*⟩`]{`⟨*angle*⟩`}`, where ⟨*options*⟩ can be any of the package options to apply only to this value. ⟨*angle*⟩ can be given either as a decimal number or as a semi-colon separated list of degrees, minutes and seconds, *i.e.* `\ang{`⟨*decimal angle*⟩`}` or `\ang{`⟨*degrees*⟩`;`⟨*minutes*⟩`;`⟨*seconds*⟩`}`. By default, no space is introduced between angles and the degrees, minutes and seconds markers.

10° 12.3° 4.5°
1°2′3″ 0°0′1″
10° −0°1′

```
\ang{10} \ang{12.3} \ang{4,5}\\
\ang{1;2;3} \ang{;;1}\\
\ang{+10;;} \ang{-0;1;}
```

By default, angles with no degrees (or minutes) are zero-filled; angles with degrees but no minutes or seconds are not filled. This behaviour can be altered using the package options.

0°0′1″ 1″
2° 2°0′0″
0°3′0″ 4°0′0″ 0°0′5″

```
\ang{;;1} \ang[padangle=none]{;;1}\\
\ang{2;;} \ang[padangle=all]{2;;}\\
\sisetup{padangle=all} \ang{;3;} \ang{4;;} \ang{;;5}
```

The `\num` macro is used to typeset each number of the angle, so the options for `\num` also apply here. The `anglesep` value can be used to separate degrees, minutes and seconds.

1.05° 1,05°
3.678 90° 3.678,90°
9°8′7″ 9° 8′ 7″

```
\ang{1.05}    \ang[decimalsymbol=comma]{1.05}\\
\ang{3.67890} \ang[digitsep=comma]{3.67890}\\
\ang{9;8;7}   \ang[anglesep=thin]{9;8;7}
```

The degrees, minutes and seconds signs can be placed over the decimal sign using the `astroang` option. This is designed on the assumption that only the last number given has a decimal part.

1.2° 1°2	`\ang{1.2} \ang[astroang]{1.2}\\`
1°2.3′ 1°2′.3	`\ang{1;2.3;} \ang[astroang]{1;2.3;}\\`
1°2′3.4″ 1°2′3″.4	`\ang{1;2;3.4} \ang[astroang]{1;2;3.4}`

7 Units and values

`\SI` The core aim of siunitx is correctly typesetting values which have units. The main output macro here is `\SI`, which has the same syntax as the macros with the same name in SIstyle and unitsdef packages. The `\SI` macro takes two mandatory arguments, in addition to the optional set up argument, and a second optional argument: `\SI[`⟨*options*⟩`]{`⟨*number*⟩`}[`⟨*preunit*⟩`]{`⟨*unit*⟩`}`. The ⟨*number*⟩ argument operates in exactly the same manner as the equivalent argument of the `\num` macro. ⟨*unit*⟩ will be typeset with a non-breakable space between it and the preceding number, with font control as outlined earlier. Finally, ⟨*preunit*⟩ is a unit to be typeset *before* the numerical value (most likely to be a currency). Some examples illustrate the general power of the macro.

$1.23\,\mathrm{J\,mol^{-1}\,K^{-1}}$	`\SI[mode=text]{1.23}{J.mol^{-1}.K^{-1}}\\`
$0.23 \times 10^7\,\mathrm{cd}$	`\SI{.23e7}{\candela}\\`
£1.99/kg	`\SI[per=slash]{1.99}[\pounds]{\per\kilogram}\\`
$70\,\mathrm{m\,s^{-1}}$	`\SI{70}{\metre\per\second}\\`
$1.345\,^{\mathrm{A}}/_{\mathrm{mol}}$	`\SI[per=frac,fraction=nice]{1,345}{\ampere\per\mole}`

The use of unit macros outside of the `\SI` macro is described later.

7.1 Literal units

Units can be input in two ways, inspired by SIstyle and SIunits. The SIstyle-like method uses literal input. Four characters have a special meaning:

- "^" The superscript character is used without the usual need for surrounding maths characters (`$`);
- "." and ",": the full stop (point) symbol and comma are made active, and produce the current contents of the `unitsep` option;
- "~" The contents of the `unitspace` option are typeset by a tilde.

This allows ready input of units.

$10\,\mathrm{kg\,m\,s^{-2}}$	`\SI{10}{kg.m.s^{-2}}\\`
$1.453\,\mathrm{g/cm^3}$	`\SI{1.453}{g/cm^3}\\`
$33.562\,\mathrm{cd\,s}$	`\SI{33.562}{cd~s}\\`
$100\,\mathrm{m\,s^{-2}}$	`\SI[unitsep=medium]{100}{m.s^{-2}}`

The literal unit system will correctly typeset input containing the symbols µ (micro), ° (degree) and Å (ring-A).[9]

10 µm	`\SI{10}{µm}\\`
20 °C	`\SI{20}{°C}\\`
30 Å	`\SI{30}{Å}`

[9] Currently this works with XeTeX and inputenc using the `latin1`, `latin5` and `latin9` encodings.

7.2 The unit interpreter

The second operation mode for the `\SI` macro is based on the behaviour of Slunits. Here, each unit, SI multiple prefix and power is given a macro name. These are entered in a method very similar to the reading of the unit name in English.

$10\,\mathrm{kg\,m\,s^{-2}}$ `\SI{10}{\kilo\gram\metre\per\second\squared}\\`
$1.453\,\mathrm{g\,cm^{-3}}$ `\SI{1.453}{\gram\per\cubic\centi\metre}\\`
$33.562\,\mathrm{cd\,s}$ `\SI{33.562}{\candela\second}\\`
$100\,\mathrm{m\,s^{-2}}$ `\SI[unitsep=thin]{100}{\metre\per\Square\second}\\`
$4.56 \times 10^{3}\,\mathrm{m\,s^{-1}}$ `\SI[prefixsymbolic=false]{4.56}{\kilo\metre\per\second}`

On its own, this is very similar to Slunits, and is less convenient than the direct input method.[10] However, the package allows you to define new unit macros; a large number of pre-defined abbreviations are also supplied. More importantly, by defining macros for units, instead of literal values, new functionality is made available. Units may be re-defined to give different output, and handling of reciprocal values can be altered.

$10\,\frac{\mathrm{g\,m}}{\mathrm{s^2}}$ `\SI[per=frac,fraction=frac]{10}{\gram\metre\per\second\squared}\\`
$1.453\,\mathrm{g/cm^3}$ `\SI[per=slash]{1.453}{\gram\per\cubic\centi\metre}\\`
$33.562\,\mathrm{cd\,s}$ `\SI{33.562}{\candela\second}\\`
$100\,\mathrm{m/s^2}$ `\SI[per=frac,fraction=nice]{100}{\metre\per\Square\second}`

The unit processor will trap *some* errors in the input and give the "best guess" result. However, it is down to the user to check the output.

7.3 Powers of units

`\Square` `\ssquare` `\squared` `\cubic` `\cubed`

Including powers in units is handled using a "natural language" method. Thus preceding a unit by `\Square` or `\cubic` which raise the unit to the appropriate power, while `\squared` or `\cubed` follow the unit they apply to. The `\Square` macro is capitalised to avoid a name clash with pstricks; the alternative `\ssquare` is also provided.

$10\,\mathrm{m^2}$ `\SI{10}{\metre\squared}\\`
$20\,\mathrm{m^2}$ `\SI{20}{\Square\metre}\\`
$30\,\mathrm{m^3}$ `\SI{30}{\metre\cubed}\\`
$40\,\mathrm{m^3}$ `\SI{40}{\cubic\metre}`

`\per` The `\per` macro intelligently creates reciprocal powers, and also adds the power -1 when appropriate.

$10\,\mathrm{s^{-2}}$ `\SI{10}{\per\second\squared}\\`
$20\,\mathrm{s^{-2}}$ `\SI{20}{\per\Square\second}\\`
$30\,\mathrm{1/s^3}$ `\SI[per=frac,fraction=nice]{30}{\per\second\cubed}\\`
$40\,\mathrm{/s^3}$ `\SI[per=slash]{40}{\per\cubic\second}\\`
$50\,\mathrm{s^{-1}}$ `\SI{50}{\per\second}\\`
$60\,\mathrm{m^{-1}\,cd^2}$ `\SI{60}{\per\metre\Square\candela}`

`\tothe` `\raiseto` For powers not defined above or with `\newpower`, the `\tothe` macro can be

[10] Users of Slunits should note the lack of need for a `\usk`-type macro.

used “in line” to produce a power. As follows from standard English usage, this comes after the unit. `\raiseto` achieves the same, but is used *before* a unit to add a power *after*.[11]

$16.86\,\mathrm{m}^{4}$
$7.895\,\mathrm{N}^{-6}$
$1.34\,\mathrm{K}^{-7}$

```
\SI{16.86}{\metre\tothe{4}}\\
\SI{7.895}{\raiseto{-6}\newton}\\
\SI{1.34}{\per\kelvin\tothe{7}}
```

7.4 Units with no values

`\si` For typesetting the symbol for a unit on its own, with the full font control and without extra spaces, the `\si` macro is provided.[12] The macro name avoids a clash with the functionality of the earlier packages, but is similar to `\ilu` from the unitsdef package.

$\mathrm{kg\,m/s^{2}}$
$\mathrm{kg\,m/s^{2}}$
$\mathrm{mol\,dm^{-3}}$

```
\SI{}{kg.m/s^2}\\
\si{kg.m/s^2}\\
\si[mode=text,unitsep=thin]{\mole\per\cubic\deci\metre}
```

7.5 Free-standing units

Users of the unitsdef package will be a accustomed to using unit macros on their own (following a value) or with an optional argument containing a number. In both cases, only a single unit macro could be used. siunitx supports both operation modes, with the limitation that units trailing values loose font control of the value. When used in this way, the units *do not* take an optional keyval argument.

$123\,\mathrm{m}$
$123\,\mathrm{K}$
$234\,\mathrm{A}$
$6\,\mathrm{s}$

```
\sisetup{prespace,allowoptarg}
123\metre\\
\kelvin[123]\\
\sisetup{mode=text} \ampere[234]\\
6\second
```

7.6 Pre-defined units, prefixes and powers

`\metre` `\meter` The package always defines the seven base SI units, irrespective of any package options given (Table 5). The kilogram is notable as by default it is a *base* unit with a prefix. Thus, when the package is loaded with the option `load={}`, `\kilo` and `\gram` *are not defined*. As metre is often spelled as “meter” in the US, the macro `\meter` is provided in addition to the `\metre` macro.[13]

By default, a number of additional definitions are created by the package. These are controlled by the `load` and `noload` options. Unless specifically requested with the option `noload=prefix`, siunitx defines the standard prefixes for powers of ten (Table 6). This leads to the redefinition of `\kilogram` as `\kilo\gram`. The macro `\deka` is provided, as this is used as an alias for `\deca` in some places. The package also defines a number of derived SI units

[11] `\raiseto` acts in the same way as `\tothe` when used in a literal context: the power will be produced where the macro is, rather than moving after the next item.

[12] The same effect can be achieved using the `\SI` macro with an empty numerical argument.

[13] The official SI spelling for the unit is “metre”.

Table 5: The seven base SI units

Unit	Macro	Symbol
kilogram	\kilogram	kg
metre	\metre	m
second	\second	s
mole	\mole	mol
kelvin	\kelvin	K
ampere	\ampere	A
candela	\candela	cd

Table 6: The SI prefixes (load=prefix)

Prefix	Macro	Power	Symbol	Prefix	Macro	Power	Symbol
yocto	\yocto	10^{-24}	y	deca	\deca	10^{1}	da
zepto	\zepto	10^{-21}	z	hecto	\hecto	10^{2}	h
atto	\atto	10^{-18}	a	kilo	\kilo	10^{3}	k
femto	\femto	10^{-15}	f	mega	\mega	10^{6}	M
pico	\pico	10^{-12}	p	giga	\giga	10^{9}	G
nano	\nano	10^{-9}	n	tera	\tera	10^{12}	T
micro	\micro	10^{-6}	µ	peta	\peta	10^{15}	P
milli	\milli	10^{-3}	m	exa	\exa	10^{18}	E
centi	\centi	10^{-2}	c	zetta	\zetta	10^{21}	Z
deci	\deci	10^{-1}	d	yotta	\yotta	10^{24}	Y

which have assigned names and symbols (Table 7). Note that \Gray is capitalised to avoid a name clash with the pstricks package.[14]

In addition to these units, there are three other groups of units for use with the SI system which do not fit into the above. These are those derived from physical measurements (Table 8), those considered "accepted" (Table 9), and those accepted temporarily (Table 10).[15] The unit "litre" is often spelled "liter" in the US; both spellings are provided by siunitx, with \liter giving L and \litre producing l.

\litre
\liter

7.7 Prefixed and abbreviated units

Many basic units have prefixes which are commonly used with the unit, such as centimetre or megahertz. The package therefore defines a number of common prefixed units (load=prefixed). Several of these also have obvious abbreviations (such as \MHz for \megahertz), which are made available by load=abbr. In common with the units discussed above, the prefixed and abbreviated unit definitions are loaded by default.

[14] The macros \ohm and \celsius are not defined by siunitx if the gensymb package is loaded.
[15] These are supposed to be replaced over time by SI units.

Table 7: The derived SI units with defined names (`load=derived`)

Unit	Macro	Symbol	Unit	Macro	Symbol
becquerel	`\becquerel`	Bq	newton	`\newton`	N
celsius	`\celsius`	℃	ohm	`\ohm`	Ω
coulomb	`\coulomb`	C	pascal	`\pascal`	Pa
farad	`\farad`	F	radian	`\radian`	rad
Gray	`\Gray`	Gy	siemens	`\siemens`	S
	`\ggray`	Gy	sievert	`\sievert`	Sv
hertz	`\hertz`	Hz	steradian	`\steradian`	sr
henry	`\henry`	H	tesla	`\tesla`	T
joule	`\joule`	J	volt	`\volt`	V
katal	`\katal`	kat	watt	`\watt`	W
lumen	`\lumen`	lm	weber	`\weber`	Wb
lux	`\lux`	lx			

Table 8: Units derived from experiments (`load=physical`)

Unit	Macro	Symbol
electron volt	`\electronvolt`	eV
unified atomic mass unit	`\atomicmassunit`	u
	`\atomicmass`	u

Table 9: Units accepted for use with SI (`load=accepted`)

Unit	Macro	Symbol
bel	`\bel`	B
day	`\Day`	d
	`\dday`	d
degree	`\degree`	°
hour	`\hour`	h
litre	`\litre`	l
	`\liter`	L
minute	`\minute`	min
minute (arc)	`\arcmin`	′
neper	`\neper`	Np
percent	`\percent`	%
second (arc)	`\arcsec`	″
tonne	`\tonne`	t

Table 10: Additional (temporary) SI units (`load=addn`)

Unit	Macro	Symbol
ångström	`\angstrom`	Å
are	`\are`	a
curie	`\curie`	Ci
bar	`\BAR`	bar
	`\bbar`	bar
barn	`\barn`	b
gal	`\gal`	Gal
hectare	`\hectare`	ha
millibar	`\millibar`	mbar
rad	`\rad`	rad
rem	`\rem`	rem
roentgen	`\roentgen`	R

Table 11: Prefixed (`load=prefixed`) and abbreviated (`load=abbr`) units

Unit	Macro	Symbol	Abbreviation
Masses			
kilogram	`\kilogram`	kg	`\kg`
femtogram	`\femtogram`	fg	`\fg`
picogram	`\picogram`	pg	`\pg`
nanogram	`\nanogram`	ng	`\nanog`
microgram	`\microgram`	µg	`\micg`
milligram	`\milligram`	mg	`\mg`
atomic mass	`\atomicmass`	u	`\amu`
Lengths			
picometre	`\picometre`	pm	`\picom`
nanometre	`\nanometre`	nm	`\nm`
micrometre	`\micrometre`	µm	`\micm`
millimetre	`\millimetre`	mm	`\mm`
centimetre	`\centimetre`	cm	`\cm`
decimetre	`\decimetre`	dm	`\dm`
kilometre	`\kilometre`	km	`\km`
Times			
second	`\second`	s	`\Sec`
attosecond	`\attosecond`	as	`\as`
femtosecond	`\femtosecond`	fs	`\fs`
picosecond	`\picosecond`	ps	`\ps`
nanosecond	`\nanosecond`	ns	`\ns`
microsecond	`\microsecond`	µs	`\mics`
millisecond	`\millisecond`	ms	`\ms`

Continued on next page

Unit	Macro	Symbol	Abbreviation
Moles			
femtomole	`\femtomole`	fmol	`\fmol`
picomole	`\picomole`	pmol	`\pmol`
nanomole	`\nanomole`	nmol	`\nmol`
micromole	`\micromole`	µmol	`\micmol`
millimole	`\millimole`	mmol	`\mmol`
Currents			
picoampere	`\picoampere`	pA	`\pA`
nanoampere	`\nanoampere`	nA	`\nA`
microampere	`\microampere`	µA	`\micA`
milliampere	`\milliampere`	mA	`\mA`
kiloampere	`\kiloampere`	kA	`\kA`
Areas			
square centimetre	`\squarecentimetre`	cm^2	`\cms`
	`\centimetresquared`	cm^2	
square metre	`\squaremetre`	m^2	
square kilometre	`\squarekilometre`	km^2	
Volumes			
microlitre	`\microlitre`	µl	`\micl`
millilitre	`\millilitre`	ml	`\ml`
cubic centimetre	`\cubiccentimetre`	cm^3	`\cmc`
	`\centimetrecubed`	cm^3	
cubic decimetre	`\cubicdecimetre`	dm^3	`\dmc`
Frequencies			
hertz	`\hertz`	Hz	`\Hz`
millihertz	`\millihertz`	mHz	`\mHz`
kilohertz	`\kilohertz`	kHz	`\kHz`
megahertz	`\megahertz`	MHz	`\MHz`
gigahertz	`\gigahertz`	GHz	`\GHz`
terahertz	`\terahertz`	THz	`\THz`
Potentials			
millivolt	`\millivolt`	mV	`\mV`
kilovolt	`\kilovolt`	kV	`\kV`
Energies			
kilojoule	`\kilojoule`	kJ	`\kJ`
electronvolt	`\electronvolt`	eV	`\eV`
millielectronvolt	`\millielectronvolt`	meV	`\meV`
kiloelectronvolt	`\kiloelectronvolt`	keV	`\keV`
megaelectronvolt	`\megaelectronvolt`	MeV	`\MeV`
gigaelectronvolt	`\gigaelectronvolt`	GeV	`\GeV`
teraelectronvolt	`\teraelectronvolt`	TeV	`\TeV`
kilowatthour	`\kilowatthour`	kWh	`\kWh`

Continued on next page

Unit	Macro	Symbol	Abbreviation
Powers			
milliwatt	`\milliwatt`	mW	
kilowatt	`\kilowatt`	kW	
megawatt	`\megawatt`	MW	
Capacitances			
femtofarad	`\femtofarad`	fF	
picofarad	`\picofarad`	pF	
nanofarad	`\nanofarad`	nF	
microfarad	`\microfarad`	µF	
millifarad	`\millifarad`	mF	
Resistances			
kiloohm	`\kiloohm`	kΩ	
megaohm	`\megaohm`	MΩ	
gigaohm	`\gigaohm`	GΩ	
millisiemens	`\millisiemens`	mS	
Forces			
millinewton	`\millinewton`	mN	
kilonewton	`\kilonewton`	kN	
Other units			
hectopascal	`\hectopascal`	hPa	
megabecquerel	`\megabecquerel`	MBq	
millisievert	`\millisievert`	mSv	

7.8 Defining new units

`\newunit`
`\renewunit`
`\provideunit`

New units are produced using the `\newunit` macro. This works as might be expected: `\newunit[`⟨*options*⟩`]{`⟨*unit*⟩`}{`⟨*symbol*⟩`}`, where ⟨*symbol*⟩ can contain literal values, other units, multiple prefixes, powers and `\per`. The ⟨*options*⟩ argument can be any suitable options, and applies the specific unit macro only. The most obvious example for using this macro is the `\degree` unit.[16] The (first) optional argument to `\SI` and `\si` can be used to override the settings for the unit. The `\renewunit` and `\provideunit` macros take the same arguments.

```
\SI{3.1415}{\degree}\\
\newunit[valuesep=none]{\oddunit}{XXX}
\SI{12345}{\oddunit}
\SI[valuesep=thick]{67890}{\oddunit}
```

3.1415°
12 345XXX 67 890 XXX

As with the LaTeX commands `\newcommand`, *etc.*, the choice of `\newunit`, `\renewunit` or `\provideunit` depends on the presence of an existing definition. While `\newunit` should be used when a unit has not been previously defined, `\renewunit` will issue a warning if the named unit does not already exist. `\provideunit` defines the unit if it does not exist, and otherwise does nothing

[16] Although the `\ang` macro is preferred for this job.

at all. The same behaviour is seen with `\providepower` and `\provideprefix` (*vide infra*).

Output that is only valid in maths mode requires `\ensuremath`, text-only input requires `\text`. In the example below, `\mathnormal` is used to force the font choice only for the single character.[17]

$10\,\mathrm{m}\,\pi^{-2}$

```
\newunit{\SIpi}{\ensuremath{\mathnormal{\pi}}}
\SI{10}{\metre\per\SIpi\squared}
```

`\newpower` `\renewpower` `\providepower`

Powers are defined: `\newpower[post]{⟨power⟩}{⟨num⟩}`. Here, ⟨*power*⟩ is the name of the power macro and ⟨*num*⟩ is the (positive) number it represents. The later argument is always processed internally by `\num`, but *must* be a number. Giving the optional argument `post` indicates to the package that the power will come after the unit it applies to; by default it is assumed that it will come before.

kg^{4}
m^{4}

```
\newpower{\quartic}{4}
\newpower[post]{\totheforth}{4}\\
\si{\kilogram\totheforth}\\
\si{\quartic\metre}
```

`\newprefix` `\renewprefix` `\provideprefix`

The standard SI powers of ten are defined by the package, and are described above. However, the user can define new prefixes with `\newprefix`. This has syntax `\newunit[binary]{⟨prefix⟩}{⟨symbol⟩}{⟨powers-ten⟩}`, where ⟨*powers-ten*⟩ is the number of powers of ten the prefix represents. When the `binary` option is given, the prefix is a power of two. For example, `\kilo` and `\kibi` are defined:

```
\newprefix{\kilo}{k}{3}
\newprefix[binary]{\kibi}{Ki}{10}
```

8 Specialist units

In some subject area, there are units which are in common use even though they are outside of the SI system. Unlike the units discussed earlier, these specialist units are not loaded by default. In each case, they should be requested with the option `alsoload=`⟨*name*⟩.

8.1 Binary units (`binary`)

`\bit` `\byte`

The binary prefixes, `\bit`, `\byte` (Table 12) are not formally part of the SI system. They are available by giving the `alsoload=binary` option.

100 MiB

```
\SI{100}{\mebi\byte}
```

8.2 Synthetic chemistry (`synchem`)

`\mmHg` `\molar` `\Molar` `\torr` `\dalton`

The `synchem` file adds the common chemistry units `\mmHg`, `\molar`, `\Molar`, `\torr` and `\dalton` to siunitx. The `\Molar` macro is somewhat awkward, as it

[17] The `\mathrm` font used for this document has an "ß" at the π position.

Table 12: Binary prefixes (`alsoload=binary`)

Prefix	Macro	Power	Symbol
kibi	`\kibi`	2^{10}	Ki
mebi	`\mebi`	2^{20}	Mi
gibi	`\gibi`	2^{30}	Gi
tebi	`\tebi`	2^{40}	Ti
pebi	`\pebi`	2^{50}	Pi
exbi	`\exbi`	2^{60}	Ei

Table 13: High-energy physics units (`alsoload=hep`)

Unit	Macro	Symbol	Abbreviation
Areas			
yoctobarn	`\yoctobarn`	yb	`\yb`
zeptobarn	`\zeptobarn`	zb	`\zb`
attobarn	`\attobarn`	ab	`\ab`
femtobarn	`\femtobarn`	fb	`\fb`
picobarn	`\picobarn`	pb	`\pb`
nanobarn	`\nanobarn`	nb	`\nb`
Other units			
micron	`\micron`	µm	
millirad	`\mrad`	mrad	
gauss	`\gauss`	G	

can be given as either "M" or "M". The later is obviously easily confused with the sign for the prefix mega. By default, siunitx uses the UK default of a small-caps symbol. The `\dalton` unit is defined here as this name is not recognised by the various international bodies: the symbol u is preferred.

1 M HCl
760 Torr
0.01 mmHg
3.0 mol dm^{-3}
106.42 Da

```
\SI{1}{\Molar} HCl\\
\SI{760}{\torr}\\
\SI{0.01}{\mmHg}\\
\SI{3.0}{\molar}\\
\SI{106.42}{\dalton}
```

8.3 High-energy physics (hep)

In contrast to hepunits, siunitx does not define a long list of compound units for high-energy physics.[18] Instead, a small selection of new units are defined (Table 13). The mechanisms provided by siunitx should avoid the need for large numbers of abbreviations. For example, the hepunits `\MinveV` can be given as `\per\MeV` in siunitx, which requires only one more character.

`\clight` `\eVperc`

The `hep` option defines two units which are slightly unusual. `\clight` gives c, which is recognised as a unit when used in the appropriate circumstances. The

[18] Using the `emulate=hepunits` option will load a file defining those.

second unit provided is `\eVperc`, which is commonly-used and clear enough for a compound definition. Notice that the value of `eVcorrb` will need to be adjusted when using this unit.

4.657 MeV/c^2

```
\SI[per=slash,eVcorrb=0.4ex]{4.657}{\mega\eVperc\squared}
```

8.4 Astronomy (`astro`)

`\parsec` `\lightyear`

For astronomers, the `\parsec` and `\lightyear` units are available, and give the obvious results.

12 pc
1 ly

```
\SI{12}{\parsec}\\
\SI{1}{\lightyear}
```

9 Font control

Following the lead of SIstyle, siunitx provides control over the font used to typeset output. By default, all text is typeset using the current upright serif maths font, whether the macros are given in text or maths mode. Some examples will show the effect.

10 10
20° 20°
30 kg
40 kg
50

```
\num{10} $\num{10}$\\
\sffamily \ang{20} $\ang{20}$\\
\textbf{\SI{30}{\kilo\gram}}\\
\boldmath $\SI{40}{\kilo\gram}$
\[ \num{50} \]
```

In contrast, by setting `obeyall`, the current font is used: this may be maths or text, depending on the context.

1°1′1″ 1°1′1″
2°2′2″ 2°2′2″
3°3′3″ 3°3′3″
4°4′4″ 4°4′4″
5°5′5″

```
\sisetup{obeyall}\\
\ang{1;1;1} $\ang{1;1;1}$\\
\sffamily \ang{2;2;2} $\ang{2;2;2}$\\
\textbf{\ang{3;3;3}} \boldmath $\ang{3;3;3}$\\
\emph{\ang{4;4;4}} \emph{$\ang{4;4;4}$}
\[ \ang{5;5;5} \]
```

Fine control of which elements of the local font are used is available with the `obeyfamily`, `obeybold`, `obeyitalic` and `obeymode` options.

1°1′1″ 1°1′1″
2°2′2″ 2°2′2″
3°3′3″ 3°3′3″
4°4′4″ 4°4′4″
5°5′5″

```
\sisetup{obeyfamily}\\
\ang{1;1;1} $\ang{1;1;1}$\\
\sffamily \ang{2;2;2} $\ang{2;2;2}$\\
\sisetup{obeybold}
\textbf{\ang{3;3;3}} \boldmath $\ang{3;3;3}$\\
\emph{\ang{4;4;4}} \emph{$\ang{4;4;4}$}
\[ \ang{5;5;5} \]
```

10 Package options

`\sisetup`

The "native" options for the package are all given using the key–value method.

Most of the package options can be given both when loading the package and at any point in the document. This is achieved using the `\sisetup` macro.

The package options take a number of different forms.

- `option=`⟨*bool*⟩ Simple true/false values. These macros all default to `option=true`, meaning that giving the option name along will set the appropriate flag.
- `option=`⟨*choice*⟩ Take a single item from a pre-determined list. Depending on the value, one or more internal states will be altered. Values not on the list are ignored (with a warning).
- `option=`⟨*choice*`,`*literal*⟩ If the given value is a ⟨*choice*⟩, then the internal settings for that choice are used. Any other value is used directly.
- `option=`⟨*literal*⟩ The given value is used as a literal by the package.
- `option=`⟨*csname*⟩ These options expect a command sequence as a value.
- `option=`⟨*length*⟩ Requires a TEX length, for example `0.5ex`.
- `option=`⟨*list*⟩ Takes a list of one or more items, which are not determined in advance.
- `option=`⟨*number*⟩ Takes a number (possibly including an exponent part).

The package has a large range of options, to allow full control of the various features of the package. These control differing aspects of the package, and are given below in groups based on function. Where the key has a default value, it is given in bold.

10.1 Font family and style

The font used when typesetting material can be tightly controlled using siunitx. A number of options affect how the package matches the surrounding font, and the font families used to achieve this. The default is to use the current upright maths serif font with no variation.

`mode` `textmode` `valuemode` `unitmode` `obeymode`

The output of siunitx can occur using either text or maths mode. The package option `mode` determines which is used: valid options are **`maths`** and `text`.[19] The shortcut `textmode` is provided for setting `mode=text` quickly. Further refinement is possible using the `valuemode` and `unitmode` options. These apply to numbers (the output of `\num` and the first mandatory argument of `\SI`) and units (all other output), respectively. By setting the `obeymode` flag, the package will use the local typesetting mode (maths or text).

`obeyall` `obeyfamily` `obeybold` `obeyitalic`

The detection and matching of surrounding text can be controlled using a number of Boolean package options. `obeyall` turns on all of the detection. Thus output with `obeyall` in force will always match the local text appearance. `obeyfamily` instructs the package on detecting the surrounding font family (Roman, sans serif, fixed width), but does not detect bold or italic. `obeybold` detects the local bold setting, whilst `obeyitalic` picks up italic fonts.

[19] Here and in all other cases, either UK or US spelling may be used. Thus `mode=maths` or `mode=math` have exactly the same effect.

Bold detection is influenced by the value of `inlinebold`, which takes values **`text`** and `maths`. The package can detect the local value of bold for either the surrounding text, or the surrounding inline (`$...$`) maths. The `obeyitalic` option does *not* have the same facility (maths is italic anyway).

`mathsrm` `mathssf` `mathstt` `textrm` `textsf` `texttt`

The font commands used by the package to achieve the above are all available for user modification. The options `mathsrm`, `mathssf` and `mathstt` hold the command sequences used in maths mode,[20] while `textrm`, `textsf` and `texttt` do the same for text mode. By default, these contain the obvious command names, for example `mathsrm=mathrm` and `texttt=ttfamily`. However, they can be set at will: the macro names indicate the nature of the surrounding text detected. For example, the value of `mathssf` is used in maths mode when the surrounding text is sans serif.

Each of the font options can be given separately for the contents of numbers and units. The option names include `value` or `unit` before the mode name. For example, the `mathsrm` option may be split into `valuemathsrm` and `unitmathsrm`.

`detectdisplay`

The font detection system can treat displayed mathematical content in two ways. This is controlled by the `detectdisplay` option. When set to **`true`**, display mathematics is treated independently from the body of the document. Thus the local *maths* font is checked for matching. In contrast, when set to `false`, display material is treated with the current running text font.

Some text

$$x = 1.2 \times 10^{3}\,\mathrm{kg\,K\,cd}$$

More text

$$y = 3\,\mathrm{m\,s\,mol}$$

```
\sffamily
Some text
\sisetup{obeyall}
\[ x = \SI{1.2e3}{\kg\kelvin\candela} \]
More text
\sisetup{detectdisplay=false}
\[ y = \SI{3}{\metre\second\mole} \]
```

10.2 Spacing and separators

`unitsep` `valuesep` `digitsep` `anglesep`

The separators between items can all be set using options taking a list of predefined items or a literal value. The "`sep`" options (`unitsep`, `valuesep`, `digitsep` and `anglesep`) all recognise `thin`, `medium`, `med`, `thick`, `space`, `cdot`, `times`, `tightcdot`, `tighttimes`, `fullstop`, `stop`, `period` and `none`. The named spaces are the normal maths separations, with `space` representing a full (non-breakable text) space, and with the obvious meanings for `cdot` and `times`. The `tight` variants reduce the spacing available. Three possible values are provided for "`.`", and `none` yields no space at all. In all cases, other values are treated literally and are typeset in maths mode. The default value is **`thin`** for all separations except `anglesep`, which is set to **`none`**.

`unitspace` `errspace`

The `unitspace` and `errspace` options again take a list or literal value, but only the "real" spaces `thin`, `medium`, `med`, `thick`, `space` and `none` are recognised in the list. The `unitspace` option controls the output generated by an explicit space (`~`) inside a unit macro, while `errspace` is used to separate a bracketed error from the main number.

[20] These can also be set using `mathrm`, `mathsf` and `mathtt`

10.3 Number formatting

numdigits
numdecimal
numexp
numsign
numaddn
numgobble

There are two groups of options for formatting numbers. The first group all begin with "num", and take literal values used by the package to parse numbers. numdigits contains the valid number symbols (**0123456789**), with numdecimal containing the decimal markers (**.,**). As in the numprint package, numexp (the list of exponent markers) recognises **deDE** as valid by default. numsign contains the sign markers for numbers (**+-\pm\mp**). numaddn and numgobble both control which other characters do not give an error when present in a number. numaddn contains valid characters which should be included in the final output "as is", whereas numgobble lists the characters that are completely ignored. In all cases, the content of the options is a simple string, for example numdigits=1234567890.

decimalsymbol
digitsep
sepfour

The second group of number options control the output of numbers after parsing. The symbol used by siunitx as a decimal marker is set by the decimalsymbol option, which can take a list of choices or a literal. The valid choices here are **fullstop**, comma, cdot and tightcdot.[21] Notice that this does not have to agree with the input marker. The other separator for numerical output is the division of digits into groups of three. The result is dependant on two options. The previously-described digitsep option controls the spacing added between groups of three numbers. For numbers consisting of exactly four digits, the sepfour Boolean option controls whether separation occurs in these cases. The default is **false**.

1234
1 234

```
\num{1234}\\
\num[sepfour]{1234}
```

seperr
numopenerr
numcloseerr

For numbers given with an error [*e.g.* 1.23(4)], the package can separate out the error part, to give for example 1.23 ± 0.04. This behaviour is activated by the seperr option, and requires that numopenerr and numcloseerr contain the left- and right-hand delimiters for the error [defaults numopenerr=(and numcloseerr=)].

1.234 ± 0.005
$(1.234 \pm 0.005) \times 10^{6}$

```
\sisetup{seperr}\\
\num{1.234(5)}\\
\num{1.234(5)e6}
```

trapambigerr
openerr
closeerr
tightpm

If the number has an exponent, or if units are not repeated, then the result can be considered ambiguous. By default, the package adds the markers stored in openerr and closeerr to remove the ambiguity; the options have the same default values as the input error markers. Detection of a potentially-ambiguous error is controlled by the trapambigerr option, although for numbers with units the repeatunits option is also important. The spacing around the $\pm$ sign is normally set by TEX. However, using the tightpm option will cause this to be reduced to a minimum.

$1.234 \times 10^{6} \pm 0.005 \times 10^{6}$
1.234 m ± 0.005 m
(1.234 ± 0.005) m
1.234 ± 0.005 m
1.234±0.005

```
\sisetup{seperr}\\
\num[trapambigerr=false]{1.234(5)e6}\\
\SI{1.234(5)}{\metre}\\
\sisetup{repeatunits=false}
\SI{1.234(5)}{\metre}\\
\SI[trapambigerr=false]{1.234(5)}{\metre}\\
\num[tightpm]{1.234(5)}
```

[21] fullstop also has aliases stop and period.

numprod
repeatunits

The number processor can recognise products in the numerical input; the symbols used for products are stored in `numprod`, with the default value of "`x`". By default, the `\SI` macro will repeat the units for a number given in this way. This behaviour is altered by the `repeatunits` option, which takes the values **`true`**, `false` and `power`. The later applies only when providing multiplied numbers with units, and converts the unit to the appropriate power rather than repeating the units.[22] Notice that when applied to errors, `repeatunits` takes priority over `trapambigerr`.

$1 \times 2 \times 3$
$4\,\mathrm{m} \times 5\,\mathrm{m} \times 6\,\mathrm{m}$
$1.2 \times 3.4 \times 5.6\,\mathrm{mm}^3$
$(1.234 \pm 0.005)\,\mathrm{m}$
$9.109\,389\,7 \pm 0.000\,005\,4 \times 10^{-31}\,\mathrm{kg}$
$9.109\,389\,7 \times 10^{-31}\,\mathrm{kg} \pm 0.000\,005\,4 \times 10^{-31}\,\mathrm{kg}$
$1 \times 2 \times 3\,\mathrm{m}^3$

```
\num{1 x 2 x 3}\\
\SI{4 x 5 x 6}{\metre}\\
\sisetup{repeatunits=false}
\SI{1.2 x 3.4 x 5.6}{\milli\metre\cubed}\\
\SI[seperr]{1.234(5)}{\metre}\\
\SI[seperr,
  trapambigerr=false]
  {9.1093897(54)e-31}{\kilo\gram}\\
\SI[seperr,repeatunits,
  trapambigerr=false]
  {9.1093897(54)e-31}{\kilo\gram}\\
\SI[repeatunits=power]{1 x 2 x 3}{\metre}
```

expproduct
expbase
allowzeroexp

The formatting of exponents is controlled by `expproduct` and `expbase`. `expproduct` sets the symbol used to indicate a product for exponents (*e.g.* the $\times$ in 2×10^2), while the value of `expbase` sets the power used (the 10 in the example). Both options accept a very short list of options: **`times`**, `tighttimes`, `cdot` and `tightcdot` for the product, and **`ten`** and `two` for the power.[23] Other choices are used literally. Also relevant to exponent processing is the `allowzeroexp` option. By default, the package will suppress a zero exponent, but setting the flag will allow the output of 10^0.

addsign
sign
retainplus

Additions to the input can take the form of implicit signs and padded zeros. The `addsign` option takes a list of potential sites to add a sign: **`none`**, `mantissa`, `exponent` and `both`.[24] If no sign is given in the input, the setting here determines if one is added. The sign to add is stored in `sign`, which takes the list of choices **`plus`**, `minus`, `pm` and `mp`, or uses the input literally (in maths mode). For positive numbers, the `retainplus` option causes a $+$ sign explicitly in the input to be retained. By default, the package will remove such signs.

padnumber

The `padnumber` option controls the addition of zeros to the input, to "pad" the result. The option takes a list of choices: **`leading`**, `trailing`, `both` and `none`.[25] No additional precision is added by this option; integer input will not add a decimal point.

[22] This is a very simply option: do not expect it to work with anything except areas and volumes.
[23] The `tighttimes` and `tightcdot` options give the rather questionable results: 1×10^2 and $1\cdot10^2$, as opposed to 1×10^2 and $1 \cdot 10^2$.
[24] Aliases are provided: `mant` = `mantissa`, `exp` = `exponent`, `all` = `true` = `both`, `false` = `none`.
[25] Aliases: `all` = `true` = `both`, `false` = `none`.

0.1
2
3.0
4.0
0.5
6
7
.8

```
\num[padnumber=leading]{.1}\\
\num[padnumber=leading]{2.}\\
\num[padnumber=trailing]{3.}\\
\num[padnumber=both]{4.}\\
\num[padnumber=both]{.5}\\
\num[padnumber=both]{6}\\
\num[padnumber=none]{7.}\\
\num[padnumber=none]{.8}
```

`fixdp` `dp`

In contrast to the `padnumber` option, the package can alter the precision of the input number if the `fixdp` option is set. The will fix the decimal places of the output to the number stored in the `dp` option. The later should be a positive integer.

1.000
53.900
4.568
−1.294
−1.295
10.000

```
\sisetup{fixdp,dp=3}
\num{1}\\
\num{53.9}\\
\num{4.56783}\\
\num{-1.2942}\\
\num{-1.2949}\\
\num{9.9999}
```

10.4 Angle formatting

`padangle` `strictarc`

The angle formatter uses `\num` to format numbers: any options for numbers are therefore applicable here. The `padangle` option takes choices **`small`**, `large`, `all` and `none`, and controls how angles are padded when given in degrees, minutes and seconds.[26] When giving angles as arcs (in degrees, minutes and seconds), the package can detect if the correct number of semi-colons have been given. This is controlled by the `strictarc` option, which is a Boolean switch with a **`true`** default. With `strictarc` set to `false`, an incomplete arc is interpreted as degrees and minute, while an over-complete one will drop excess input.

1°
2°0′0″
3°
0°4′0″
5′5″
1°2′
1°2′3″

```
\ang[padangle=none]{1;;}\\
\ang[padangle=large]{2;;}\\
\ang[padangle=small]{3;;}\\
\ang[padangle=both]{;4;}\\
\ang[padangle=none]{;5;5}\\
\ang[strictarc=false]{1;2}\\
\ang[strictarc=false]{1;2;3;4;}
```

`angformat`

The angle formatting system can convert between decimal angles and those given as degrees, minutes and seconds. This is controlled by the `angformat` option, which takes choices **`unchanged`**, `decimal` and `arc`.[27] When set to `unchanged`, nothing is done to the input. The conversion is based on TEX dimensions, and is therefore limited in accuracy. For this reason, the output is automatically rounded: output as a decimal angle is limited to three places, and that as an arc is given to a single decimal place for the seconds component.

[26] Aliases: `all` = `true` = `both`, `false` = `none`.
[27] Aliases: `decimal` = `dec`, `arc` = `dms`, `unchanged` = `none`.

$1°2′3″ = 1.034°$
$4.56° = 4°33′36.0″$

```
$\ang{1;2;3} = \ang[angformat=dec]{1;2;3}$\\
$\ang{4.56} = \ang[angformat=arc]{4.56}$
```

`astroang` For astronomers, the `astroang` option is provided. This moves the degrees, minutes or seconds symbol (as appropriate) over the decimal marker rather than after the number.

$1°2′3.4″$
$5°6′7.″8$

```
\ang{1;2;3.4}\\
\ang[astroang]{5;6;7.8}
```

10.5 Tabular material

`tabnumalign` Material typeset in `S` columns is processed internally by the `\num` macro. Thus, as with angles, the number options also apply here. The positioning of tabular material is controlled by the two options `tabnumalign` and `tabformat`. `tabnumalign` takes values **`centredecimal`**, `centre`, `left` and `right`.[28] When using `centredecimal`, the package places the decimal marker of the mantissa at the centre of the column, which then grows to accommodate the widest number given. For equal numbers of digits before and after the decimal sign, this is the easiest option. The other choices use a fixed-width box to store the number; the box is then aligned with the edges of the column.

`tabformat` The `tabformat` option sets the amount of space reserved by siunitx for the alignment box when not using the `centredecimal` setting of `tabnumalign`. The numerical parts of `tabformat` are interpreted as ⟨*pre*⟩⟨*dec*⟩⟨*post*⟩; ⟨*pre*⟩ and ⟨*post*⟩ are the number of digits before and after the decimal sign, respectively. Both signs and exponents can be included in `tabformat`, resulting in appropriate space being reserved. The entire `tabformat` input is processed using the `\num` macro internally. Thus the decimal and exponent signs used in `tabformat` are checked against `numdecimal` and `numexp`, respectively.

`tabalignexp` When `tabformat` contains exponents, two possibilities are available for alignment. The first method is to place the exponent parts so that the "×10" parts form a column, with whitespace after shorter mantissa components. In the second method, no additional space is added after the mantissa, and the exponents do no line up (Table 14). This is controlled by the `tabexpalign` option, which can be set to **`true`** or `false`.

```
\begin{table}
  \centering
  \caption{The \opt{tabalignexp} option}
  \label{tab:alignexp}
  \sisetup{tabformat=1.3e2,tabnumalign=centre}
  \begin{tabular}{SS[tabalignexp=false]}
    \toprule
    {Header} & {Header} \\
    \midrule
    1.2e3    & 1.2e3    \\
    1.234e56 & 1.234e56 \\
    \bottomrule
  \end{tabular}
\end{table}
```

[28] Aliases `centerdecimal` = `centredecimal`; `center` = `centre`.

Table 14: The tabalignexp option

Header	Header
1.2 $\times 10^3$	1.2×10^3
1.234 $\times 10^{56}$	1.234×10^{56}

Table 15: The tabautofit option

Header	Header	Header
1.2	1.200	1.2
1.2345	1.235	1.2345

tabtextalign
tabunitalign
tabalign

Cells containing no numbers are handled by siunitx in a manner similar to \multicolumn. The setting of tabtextalign is taken from the list **centre**, right and left.[29] As would be expected, these settings centre, right- or left-align the cell contents. In s columns, all content is treated as input to the \si macro. The alignment of the contents relative to the cell is controlled by the tabunitalign option, which takes options **left**, right and centre. The settings for tabnumalign, tabtextalign and tabunitalign can be set to the same value in one go with the tabalign option.

tabautofit

The contents of table cells can automatically be rounded or zero-filled to the number of decimal places given in tabformat. The is activated by the tabautofit Boolean option. As tabformat does not apply to columns with alignment centredecimal, tabautofit is also inactive for these columns (Table 15).

```
\begin{table}
  \centering
  \caption{The \opt{tabautofit} option}
  \label{tab:autofit}
  \sisetup{tabformat=1.3,tabnumalign=centre}
  % Notice the overfull hbox which results with
  % the first column
  \begin{tabular}{%
    S%
    S[tabautofit]%
    S[tabautofit,tabnumalign=centredecimal]}
    \toprule
    {Header} & {Header} & {Header} \\
    \midrule
    1.2    & 1.2    & 1.2    \\
    1.2345 & 1.2345 & 1.2345 \\
    \bottomrule
  \end{tabular}
\end{table}
```

[29] Alias centre = center.

10.6 Units

per

Most of the unit options are concerned with the processing of named units. The processor for units given as macro names can be influenced to give a variety of output formats. The `per` option defines how the keyword macro `\per` is handled. This option takes a choice from the list **`reciprocal`**, `slash` and `fraction`.[30] The default option uses `\per` to indicate reciprocal powers, whereas `slash` causes the package to use "/" to show division.

fraction
slash

The `fraction` option defines how `per=fraction` is interpreted. The list of applicable values here is **`frac`**, `nice`, `ugly` and `sfrac`. In each case, the unit is typeset as a fraction, but the macro use to achieve this varies. `frac` uses the TEX `\frac`, macro, while `nice` makes use of a `\nicefrac`-like method. The `ugly` option uses a slash in text mode and `\frac` in maths mode.[31] Finally, the setting `fraction=sfrac` uses the `\sfrac` macro from the xfrac package, when available.[32] The `slash` option sets the symbol used when `per=slash` is in force. This recognises the single keyword `slash`; anything else is used literally.

$\frac{\mathrm{m}}{\mathrm{s}}$
m/s
m/s
m/s

```
\sisetup{per=fraction}
\si[fraction=frac]{\metre\per\second}\\
\si[fraction=nice]{\metre\per\second}\\
\si[fraction=sfrac]{\metre\per\second}\\
\si[per=slash]{\metre\per\second}
```

stickyper

By default, `\per` applies only to the next unit given.[33] By setting the `stickyper` flag, this behaviour is changed so that `\per` applies to all subsequent units.[34]

$\mathrm{kg\,m^{-1}\,A}$
$\mathrm{kg\,m^{-1}\,A^{-1}}$

```
\si{\kilogram\per\metre\ampere}\\
\si[stickyper]{\kilogram\per\metre\ampere}
```

trapambigfrac
openfrac
closefrac

When using `per=slash`, multiple units in the denominator will yield a potentially ambiguous result. The `trapambigfrac` determines whether the package checks for this: this takes **`true`** and `false`. When set, the contents of `openfrac` are inserted before the denominator, and `closefrac` is inserted after.

$\mathrm{m}/(\mathrm{s\,kg})$
$\mathrm{m}/\mathrm{s\,kg}$

```
\sisetup{trapambigfrac,openfrac=(,closefrac=),per=slash}\\
\si{\metre\per\second\per\kilogram}\\
\si[trapambigfrac=false]{\metre\per\second\per\kilogram}
```

prefixsymbolic
prefixbase
prefixproduct

The unit prefixes (`\kilo`, *etc.*) are normally given as letters. However, the package can convert these into numerical powers.[35] This is controlled by the `prefixsymbolic` Boolean option, which by default is **`true`**. If `prefixsymboli` is set to `false`, the format of the prefix is controlled by

[30] Aliases `reciprocal` = `rp` = `power`, `fraction` = `frac`.

[31] Similar to the ugly option of the nicefrac package.

[32] xfrac is part of the experimental system for LaTeX3. As it requires a number of additional packages to work, siunitx does not load xfrac. If it is unavailable, the `sfrac` setting will fall back to using `\nicefrac`. See the xfrac documentation for reasons to prefer `\sfrac` to `\nicefrac`.

[33] This is the standard method of reading units in English: for example, $\mathrm{J\,mol^{-1}\,K^{-1}}$ is pronounced "joules per mole per kelvin".

[34] This is the behaviour in SIunits.

[35] Provided things are not too complex!

`prefixbase` and `prefixproduct`, which work in the same way as `expbase` and `expproduct`.

`prespace` `xspace` `allowoptarg`

By default, the single unit macros (*e.g.* `\metre`) add no space either before or after the unit. Setting the `xspace` flag to true means that the single macros are followed by the `\xspace` command (when used outside of `\SI`/`\si`). For users of unitsdef, the `prespace` macro changes the behaviour of the unit macros, so that they can immediately follow a number. As a result, the unit macros will *always* be preceded by a fixed space when the `prespace` flag is true: this will be in addition to any other space. Also relevant to users moving from unitsdef is the `allowoptarg` option. This allows single unit macros to take an optional numerical argument, in the same way that occurs in that package.

```
\metre is the symbol for metres\\
\sisetup{xspace}
No, \metre is the correct symbol\\
\sisetup{prespace}
30\metre\\
Do not use \metre in running text\\
\sisetup{allowoptarg}
\metre[40]
```

mis the symbol for metres
No, m is the correct symbol
30 m
Do not use m in running text
40 m

10.7 Symbols

User access to control the symbols used for Ω, µ, °, ′, ″, Å and ℃ is provided here. These are all literal options, which are available in text and maths mode variants. For example, `textmicro` is the code used for the µ symbol in text mode. The text mode macros should be safe when forced into text, and the maths ones when forced into maths. The symbols defined in this way are:

- `textOmega`;
- `mathsOmega`;
- `textmu`;
- `mathsmu`;
- `textdegree`;
- `mathsdegree`;
- `textminute`;
- `mathsminute`;
- `textsecond`;
- `mathssecond`;
- `textringA`;
- `mathsringA`;
- `textcelsius`;
- `mathscelsius`.

redefsymbols

When siunitx is loaded, it can check for the presence of the textcomp and upgreek packages, to provide better symbols for certain items. To prevent this, set the `redefsymbols` option `false` (the default is **`true`**).

eVcorra
eVcorrb

The eV symbol requires some fine-tuning, and so has two options of its own, both TEX lengths. `eVcorra` is the correction applied to the gap between "e" and "V" of the unit: the default is **`0.3ex`**. `eVcorrb` is the correction applied to the gap between "V" of the unit and whatever follows; the default is **`0ex`**. The optimal value for these options will depend on the current font settings.[36]

eV/m
eV/m

```
\si[per=slash]{\electronvolt\per\metre}\\
\si[per=slash,eVcorrb=0.7ex]{\electronvolt\per\metre}
```

10.8 Colour

colourall
colourunits
colourvalues

The package provides internal hooks for applying colour to part or all of the output. This requires the user to load the color or xcolor package to support colour in the output; siunitx will ignore a colour request if support is unavailable. The Boolean options `colourall`, `colourunits` and `colourvalues` are used to turn application of a given colour on or off for all output, only units and only values, respectively. All three switches are available with US spelling, *e.g.* `colorall` and `colourall` behave in the same way. With colour turned off, no `\color` command is issued internally, and output follows the surrounding text.

colour
unitcolour
valuecolour

The colour names to use for colouring output are set by the `colour`, `unitcolour` and `valuecolour` options (all also available with US spelling). The `colour` option internally sets both `unitcolour` and `valuecolour`.

Text 50
10 m
Text 70
40 s

```
{\color{brown} Text \num{50}}\\
\sisetup{colourall,colour=green}
\SI{10}{\metre}\\
{\color{purple} Text \num{70}}\\
\sisetup{colourunits=true,colourvalues=false}
\SI{40}{\second}
```

colourneg
negcolour

siunitx can automatically add a colour to negative numbers. This is turned on using the `colourneg` switch. The colour used is set by the `negcolour` option; both options are available using US spellings.

10.9 International support

locale

siunitx allows the user to switch between the typographic conventions of different (geographical) areas by using *locales*. Currently, the package is supplied with configurations for locales `UK`, `USA`, `DE` (Germany) and `ZA` (South Africa). The `locale` option is used to switch to a particular locale.

1.234 m
6,789 m

```
\SI{1.234}{\metre}\\
\SI[locale=DE]{6.789}{\metre}
```

loctolang

Locales are distinct from babel languages, as typographic conventions are not tightly integrated with language. However, it is useful to be able to associate a particular locale with a babel language. The option `loctolang` handles this, and expects pairs of values: `loctolang=`⟨*locale*⟩`:`⟨*language*⟩.

[36] This document uses `eVcorra=0.1ex`.

$6.022 \times 10^{23}\,\mathrm{mol}^{-1}$

$6{,}022 \cdot 10^{23}\,\mathrm{mol}^{-1}$

```
\sisetup{loctolang={UK:UKenglish,DE:german}}\\
\SI{6.022e23}{\per\mole}\\
\selectlanguage{german}\\
\SI{6.022e23}{\per\mole}
```

10.10 Package control

`load` `noload` `alsoload`

The package keeps most of the unit and abbreviations definitions in files separate from `siunitx.sty`. To control what is loaded, three complementary options are provided, all of which take a list of one or more choices. `load` and `alsoload` define which support configuration files are loaded. The list in `load` recognises the value `default`, which is expanded to the normal list before loading. The difference between `load` and `alsoload` is that `load` specifies the *complete* list of files to load, whereas `alsoload` adds to the existing list. To use the `load` option successfully requires knowing everything that is needed. The `noload` option can be used to delete one or more items from the `load` list, without needing to know what is on it.[37]

`log` `debug`

To control data written to the `.log` file, the `log` option is provided. This takes a value from the list **`normal`**, `none`, `minimal`, `errors` and `debug`. As would be expected, these indicate the amount of detail written to the log file. As a shortcut to `log=debug`, the package also recognises the `debug` option directly.

`strict`

Some users will want to stick closely to the official rules for typesetting units. This could be made complicated if the options for non-standards behaviour could not be turned off. The load-time option `strict` resets package behaviour to follow the rules closely, and disables options which deviate from this. If the package is loaded with the `strict` option, all output is made in maths mode using the upright serif font.

10.11 Back-compatibility options

`emulate`

The package can emulate SIunits, SIstyle, unitsdef, units, hepunits, fancyunits and fancynum. Giving the `emulate=`⟨*package*⟩ option will give the desired emulation, and combinations which would be possible with the real packages will also work here. The package will recognise the options of the emulated packages. This will automatically cause emulation to be switched on.

10.12 Summary of all options

Table 16 lists a summary of the package options (excluding those for backward-compatibility). A reminder or the input format is also provided.

Table 16: All package options

Option	Type	Description
`addsign`	List	Add sign to number
`allowzeroexp`	Boolean	Allow 10^0

Continued on next page

[37] `noload` does not prevent the loading of a file needed by one which is loaded. Thus the package may internally override a `noload` value if needed.

Option	Type	Description
angformat	List	Conversion of angle format
anglesep	List or literal	Space between angle components
astroang	Boolean	Astronomy-style angles
closeerr	Literal	Closes potential-ambiguous error
closefrac	Literal	Closes potential-ambiguous fraction
color	Literal	Colour used for units and values
colour	Literal	Colour used for units and values
colorall	Boolean	Switch for colouring all output
colourall	Boolean	Switch for colouring all output
colorneg	Boolean	Colour negative numbers
colourneg	Boolean	Colour negative numbers
colorunits	Boolean	Switch for colouring units
colourunits	Boolean	Switch for colouring units
colorvalues	Boolean	Switch for colouring values
colourvalues	Boolean	Switch for colouring values
decimalsymbol	List or literal	Decimal symbol
debug	Boolean	Write debugging data to log
detectdisplay	Boolean	Treat display maths separately
digitsep	List	Separation of digits in large numbers
dp	Integer	Number of decimal places to output numbers to
emulate	Modules	Emulation modules to load
errspace	List	Spacing of bracketed error
eVcorra	Length	Spacing correction in eV
eVcorrb	Length	Spacing correction after eV
expbase	List or Literal	Base used for exponents
expproduct	List or literal	Product sign for exponents
fixdp	Boolean	Switch for fixing decimal places of numbers
fraction	List	Method used when setting per=frac
inlinebold	List	Select how inline bold is tested
load	Modules	Modules to load
locale	Modules	Locale to follow
loctolang	Special	Associate locale with babel language
log	List	Amount of data added to log
mathOmega	Literal	"Ω" symbol in maths mode
mathcelsius	Literal	"℃" symbol in maths mode
mathdegree	Literal	"°" symbol in maths mode
mathminute	Literal	"′" symbol in maths mode
mathmu	Literal	"μ" symbol in maths mode
mathringA	Literal	"Å" symbol in maths mode
mathrm	Csname	Roman maths font
mathsOmega	Literal	"Ω" symbol in maths mode
mathscelsius	Literal	"℃" symbol in maths mode
mathsdegree	Literal	"°" symbol in maths mode
mathsecond	Literal	"″" symbol in maths mode
mathsf	Csname	Sans serif maths font

Continued on next page

Option	Type	Description
mathsminute	Literal	"′" symbol in maths mode
mathsmu	Literal	"μ" symbol in maths mode
mathsringA	Literal	"Å" symbol in maths mode
mathsrm	Csname	Roman maths font
mathssecond	Literal	"″" symbol in maths mode
mathssf	Csname	Sans serif maths font
mathstt	Csname	Fixed-width maths font
mathtt	Csname	Fixed-width maths font
mode	List	Use text or maths mode for typesetting
negcolor	Literal	Colour used for negative numbers
negcolour	Literal	Colour used for negative numbers
noload	Modules	Modules not to load
numaddn	Literal	Additional input allowed in numbers
numcloseerr	Literal	Character indicating end of numerical error
numdecimal	Literal	Decimal symbols in numbers
numdigits	Literal	Digit characters in numbers
numexp	Literal	Exponent characters in numbers
numgobble	Literal	Characters to ignore in numbers
numopenerr	Literal	Character indicating start of numerical error
numprod	Literal	Characters used for a product
numsign	Literal	Sign characters in numbers
obeyall	Boolean	Combination of obeybold, obeyitalic and obeymode
obeybold	Boolean	Check local bold setting
obeyitalic	Boolean	Check local italic setting
obeymode	Boolean	Check local mode (text/maths)
openerr	Literal	Opens potentially-ambiguous error
openfrac	Literal	Opens potentially-ambiguous fraction
padangle	List	Add zeros to blank parts of angles
padnumber	List	Add zeros to blank parts of numbers
per	List	Behaviour of \per
prefixbase	List or literal	Base used when making prefixes numerical
prefixproduct	List or literal	Product sign for prefixes
prefixsymbolic	Boolean	Behaviour of unit prefixes
prespace	Boolean	Add space before units
redefsymbols	Boolean	Use better symbols if available
repeatunits	List	Repeat units with separated errors and products
retainplus	Boolean	Retain explicit plus sign
seperr	Boolean	Separate number and error
sepfour	Boolean	Separate four-digit numbers
sign	List or literal	Sign to add to numbers
slash	List or literal	Symbol used for "/"

Continued on next page

Option	Type	Description
stickyper	Boolean	Require \per only once
strict	Boolean	Obey the rules strictly
strictarc	Boolean	Require exactly zero or two semi-colons
tabalign	List	Positioning of all column data
tabalignexp	Boolean	Alignment of exponents in tables
tabautofit	Boolean	Switch for rounding numbers to length given by tabformat
tabformat	Number	Space reserved in table for numbers
tabnumalign	List	Alignment of S column numbers
tabtextalign	List	Positioning of text in S columns
tabunitalign	List	Positioning of units in s columns
textcelsius	Literal	"°C" symbol in text mode
textdegree	Literal	"°" symbol in text mode
textminute	Literal	"′" symbol in text mode
textmu	Literal	"μ" symbol in text mode
textomega	Literal	"Ω" symbol in text mode
textringa	Literal	"Å" symbol in maths mode
textrm	Csname	Roman text font
textsecond	Literal	"″" symbol in text mode
textsf	Csname	Sans serif text font
texttt	Csname	Fixed-width text font
tightpm	Boolean	Reduce space around ±
trapambigerr	Boolean	Check for ambiguous errors
trapambigfrac	Boolean	Check for ambiguous fractions
unitcolor	Literal	Switch for colouring units
unitcolour	Literal	Switch for colouring units
unitmathrm	Csname	Roman maths font for units
unitmathsf	Csname	Sans serif maths font for units
unitmathsrm	Csname	Roman maths font for units
unitmathssf	Csname	Sans serif maths font for units
unitmathstt	Csname	Fixed-width maths font for units
unitmathtt	Csname	Fixed-width maths font for units
unitmode	List	As mode, for units only
unitsep	List or literal	Separator for units
unitspace	List or literal	Space used for "~" in units
valuecolor	Literal	Switch for colouring value
valuecolour	Literal	Switch for colouring value
valuemathrm	Csname	Roman maths font for values
valuemathsf	Csname	Sans serif maths font for values
valuemathsrm	Csname	Roman maths font for values
valuemathssf	Csname	Sans serif maths font for values
valuemathstt	Csname	Fixed-width maths font for values
valuemathtt	Csname	Fixed-width maths font for values
valuemode	List	As mode, for values only
valuesep	List or literal	Separator between value and unit
xspace	Boolean	Use xspace after units

11 Emulation of other packages

siunitx has been designed as a replacement for SIunits, SIstyle, unitsdef, units, hepunits, fancyunits and fancynum. It therefore provides options reproduce the functions of all of these packages. In this way, siunitx should be usable as a straight replacement for the older packages.[38] This means for example that the `\num` macro takes an optional star when emulating SIstyle. However, there are some points that should be remembered. In particular, siunitx validates numerical input, meaning that places where a number is expected in the older packages *require* a number when emulated by siunitx.

The numprint package has provided many useful ideas for the code used here for number formatting. The basic use of the `\numprint` (or `\np`) macro can be reproduced using siunitx. However, numprint is large and complex, with its own backward-compatibility options. As a result, emulation of numprint is not provided here. To use an numprint document with siunitx, the `\numprint` macro could be provided using the following code.

```
\newcommand*{\numprint}[2][]{\SI[obeymode]{#2}{#1}}\\
\numprint{-123456} \\
\numprint[N/mm^2]{-123456}
```

−123 456
−123 456 N/mm^2

siunitx can be used more-or-less directly to replace both dcolumn and rccol. As is explained in the code section, much of the column-alignment system here is taken from dcolumn, while rccol provided a model for a customisable system. However, neither package has been directly emulated here. The `S` column type can be used to replace both `D` and `R` columns by setting the appropriate package options.

12 Configuration files

siunitx is a modular package. The unit definitions, abbreviations and locales are all stored in configuration files. These all take names of the form `si-`⟨*name*⟩`.cfg`, where ⟨*name*⟩ is the part of the filename used as an option in `\sisetup` or when loading siunitx. Producing new configuration files therefore consists of making a suitably-named file and adding it to the path searched by TEX. The files should normally consist of settings (in `\sisetup`) and unit definitions, *etc*.

`\addtolocale` To allow arbitrary macros to be stored in locales, the `\addtolocale` macro is provided. This ensures that arbitrary text is only executed when using a locale, not when loading it. The macro takes two arguments, `{`⟨*locale*⟩`}` and `{`⟨*code*⟩`}`.

```
\addtolocale{DE}{TEST}
Some text as filler \\
\sisetup{locale=DE}
This example is rather trivial!
```

Some text as filler
TEST This example is rather trivial!

`\requiresiconfigs` To load one or more configuration files from inside another configuration file, the `\requiresiconfigs` macro is provided. This accepts a comma-separated list of configuration names, in the same way as `load` or `noload`.

[38] User macros means that they are described in the package documentation; simply not containing an `@` does not mean they will have been emulated.

In addition to the various configuration files provided with the package, a local file `siunitx.cfg` may be provided. This is read at the end of loading siunitx, and allows the user to include any local definitions or settings easily.

13 Common questions

13.1 Why do I need `\per` more than once?

The unit engine of siunitx is based around the English method for reading units out loud. Thus $\mathrm{J\,mol^{-1}\,K^{-1}}$ is pronounced "joules per mole per kelvin". Hence by default you need to put `\per` before each item in the denominator of a unit. The behaviour can be altered by setting the `stickyper` option.

13.2 Why is the order of my units changed?

Then using `per=reciprocal`, the units are typeset as given. However, when using `per=slash` or `per=fraction`, the package needs to find which ones are in the denominator. It then prints the numerator and denominator separately. So if you give a unit in the denominator *before* one in the numerator, they have to be re-ordered.[39]

$\mathrm{88\,kg^{-1}\,m}$
$\mathrm{66\,m/kg}$

```
\SI{88}{\per\kilogram\metre} \\
\SI[per=slash]{66}{\per\kilogram\metre}
```

13.3 Why are compound units not recommended outside of `\SI`/`\si`?

To fully process units made up of several parts, the processor has to know where the end of the unit is. When the unit macros are used outside of `\SI`/`\si`, this is not the case. The package therefore does its best, but results may be sub-optimal. To get consistent results, either define a new *single* unit, or keep compound units inside `\SI` and `\si`.

$\mathrm{m/\mu s}$
$\mathrm{m\,\mu s^{-1}}$
$\mathrm{m\,\mu s^{-1}}$
$\mathrm{kg\,m\,\mu s^{-1}}$
$\mathrm{kgm\,\mu s^{-1}}$

```
\metre\per\micro\second \\
\si{\metre\per\micro\second} \\
\newunit{\myunit}{\metre\per\micro\second}
\myunit
\newunit{\myunittwo}{\kilogram\myunit} \\
\myunittwo \\
\kilogram\myunit
```

Notice the difference in behaviour of `\per` in the first two lines, and the spacing error on the last line in the example.

13.4 How do I set superscripts to use lining numbers?

Lowercase ("old style") numbers are favoured by many people for use in running text. However, this does not necessarily look good in superscripts. The mode used to typeset data can be varied, so that maths mode numbers are used for the unit part of the output.

[39] "Double division" (1 s/m/kg) is mathematically incorrect.

$1234\,\mathrm{m\,s^{-1}}$
$1234\,\mathrm{m\,s^{-1}}$

```
\SI[mode=text]{1234}{\metre\per\second}\\
\SI[valuemode=text,unitmode=maths]{1234}{\metre\per\second}
```

13.5 Why do most of the examples use $\mathrm{J\,mol^{-1}\,K^{-1}}$?

The package author is a chemist, and this is the unit of entropy (disorder). It nicely demonstrates the use of the `\per` macro, and so it crops up a lot. It is also in the subsection heading here to act as a test with hyperref and moving arguments!

13.6 What can numprint do that siunitx cannot?

siunitx uses a lot of ideas from numprint: a reasonable amount of the number-processing code here is based on that in numprint. However, the two packages have somewhat different aims, and as a result there are things that numprint can do that siunitx does not implement. The main features of numprint not available here are:

- General support for numbers with base other than 10 (see nbaseprt);
- Alignment of the decimal marker of powers in tables;
- Alignment of numbers in running text;
- Specific formatting commands for TEX counters and lengths.

14 Tricks and known issues

14.1 Ensuring maths mode

Due to the possibility of output in either maths or text mode, any input which requires a particular mode needs to be protected. You cannot use `$...$`, as this can get "caught out", but also as it may give hard-to-follow errors. Always use `\ensuremath` to force maths processing, and `\text` (from the $\mathcal{AMS}$TEX bundle) to ensure text mode.

14.2 Using . and fixed spaces in units

To use a literal . in a unit, it has to be within an extra set of braces. This does not need any extra protection, unlike the situation with SIstyle (for example, no `\text` macro is needed). The fixed space (~) is more problematic: set `unitspace=space` to get a full space here.

10 V vs. NHE

```
\newunit[unitspace=space]{\myunit}{V~vs{.}~NHE}%
\SI{10}{\myunit}
```

Table 17: Passing single digit characters

Heading
1-2
1
1
-

14.3 Passing unprocessed digits through an S column

The method used to detect numbers in an S column will pick up material wrapped inside braces if there is more than a single character inside the braces. If you want to pass a *single* numerical character without processing it, you need two sets of braces (Table 17).

```
\begin{table}
  \centering
  \caption{Passing single digit characters}
  \label{tab:S-limits}
  \begin{tabular}{S[colourall,colour=orange]}
    \toprule
      {Heading} \\
    \midrule
      {1-2} \\
      {1}   \\
      {{1}} \\
      {{-}} \\ % Using {-} gives an error!
    \bottomrule
  \end{tabular}
\end{table}
```

14.4 Limitations of \mathrm

The package uses the \mathrm font family by default to typeset output in maths mode. This however has a few side-effects. For example, the Greek alphabet can give odd results.[40] The use of the \mathnormal font *may* get around this issue.

$4ß \times 10^{-7}$
$4\pi \times 10^{-7}$

```
\num[numaddn=\pi]{4\pi e-7}\\
\num[numaddn=\pi,mathsrm=mathnormal]{4\pi e-7}
```

On the other hand, you may want to use text mode, in which case \ensuremath is needed. Depending on the exact circumstances, the LaTeX protection mechanism (\DeclareRobustCommand) may be sufficient; in some cases, this will fail and the ε-TeX \protected system may be required. There are several potential pitfalls in this area; experimentation may well be needed.

$4\pi \times 10^{-7}$

```
\DeclareRobustCommand*{\numpi}{\ensuremath{\pi}}
\num[numaddn=\numpi,mode=text]{4\numpi e-7}
```

[40] This depends on your font setup; this document uses T1 encoding, which shows the issue, whereas using OT1 does not.

14.5 Entire document in sans serif font

If your entire document is not in a Roman font, using the font detection system is not the most efficient method for setting the siunitx output. Instead, the `mathrm` and `textrm` package options can be redefined.

Some text
1×10^{2}
3 N

$4 \times 10^{5}\,\mathrm{Pa}$

```
\sffamily
Some text \\
\sisetup{obeyfamily=false,mathrm=mathsf,textrm=sffamily}
\num{1e2} \\
\SI{3}{\newton}
\[ \num{4e5} \si{\pascal} \]
```

14.6 Effects of emulation

The package has been designed so that almost everything can be set using the options. In the emulation code, some internal macros are redefined. This is because the legacy packages do odd things, which are deliberately not implemented by siunitx. Using an emulation file will prevent subsequent loading of the real package. This is to prevent errors or, worse, difficult to diagnose changes to output.

14.7 Centring columns on non-decimal input

The dcolumn manual suggests using that package to align a column on a $\pm$ sign. The same type of output is possible using siunitx, but some care is needed (Table 18). Odd things may happen: use with care!

```
\begin{table}
  \caption{Non-standard \texttt{S} column}
  \label{tab:dcolumn}
  \centering
  \begin{tabular}{%
    S[digitsep=none,decimalsymbol={\,\pm\,},
      numdigits={0123456789.},numdecimal=+]}
  \toprule
  {Some Values} \\
  \midrule
  2.3456  + 0.02  \\
  34.2345 + 0.001 \\
  56.7835 + 0.067 \\
  90.473  + 0.021 \\
  \bottomrule
  \end{tabular}
\end{table}
```

14.8 Expanding content in tables

When processing `S` columns, siunitx works hard to expand any items which may give numbers. So for example you can define values as macros (Table 19). To avoid the expansion of single macros, you must either wrap them in two

Table 18: Non-standard `S` column

Some Values
2.3456 ± 0.02
34.2345 ± 0.001
56.7835 ± 0.067
90.473 ± 0.021

Table 19: Values as macros

Some Values
1.234
20.345
0.987 654
1.234
88.88

sets of braces or make them robust (using either `\DeclareRobustCommand` or `\protected`).

```
\begin{table}
  \caption{Values as macros}
  \label{tab:vmacros}
  \centering
  \newcommand*{\myvala}{1.234}%
  \newcommand*{\myvalb}{20.345}%
  \newcommand*{\myvalc}{0.987654}%
  \DeclareRobustCommand*{\myvald}{88.88}%
  \begin{tabular}{S[tabtextalign=left]}
  \toprule
  {Some Values} \\
  \midrule
  \myvala  \\
  \myvalb \\
  \myvalc \\
  {{\myvala}} \\
  \myvald \\
  \bottomrule
  \end{tabular}
\end{table}
```

It is possible to use calculated values in tables. For this to work, the calculation must take place before attempting to parse the number. This means that any calculation code should be wrapped in braces and appear before the use of the number. This is illustrated by using the datatool package to set up a miniature database, and then output the values doubled (Table 20). This example also shows that macros such as `\DLTforeach`, which construct a table while TEX is running, cannot be placed inside an `S` column (or an `s` one). Instead, an additional dummy column has been added with no inter-column space. This is

Table 20: Calculated values

Value	Doubled
66.7012	133.4024
66.0212	132.0424
64.9026	129.8052

used to contain the tale-building macro.

```
\DTLnewdb{data}
\DTLnewrow{data}\DTLnewdbentry{data}{value}{66.7012}
\DTLnewrow{data}\DTLnewdbentry{data}{value}{66.0212}
\DTLnewrow{data}\DTLnewdbentry{data}{value}{64.9026}
\begin{table}
  \caption{Calculated values}
  \label{tab:calc}
  \centering
  \sisetup{tabformat=2.4}
  \begin{tabular}{SS@{}l}
    \toprule
    {Value} & {Doubled}  &
    \DTLforeach{data}{\myvalue=value}{%
      \DTLiffirstrow {\\ \midrule}{\\}%
      \myvalue & % First column
      {\DTLmul{\myvalue}{\myvalue}{2}} \myvalue % second column
      & }\\
    \bottomrule
  \end{tabular}
\end{table}
```

14.9 Adding items after the last column of a tabular

If you use an `S` or `s` column as the last one in a tabular, and you use the array "<" construction to add items after it, the spacing may be wrong. This will occur if the column contents are of differing widths. Changing the LaTeX `\\` line ending for the plain TeX `\cr` will give the correct spacing, but does not allow adjustment of inter-row distance (Table 21).[41] In most cases, this should not be a serious issue. Notice that an extra set of braces is needed here, compared to usual LaTeX use; this is to prevent any expansion of the material by siunitx.

```
\begin{table}
  \caption{Correcting spacing in last \texttt{S} column}
  \label{tab:cr}
  \hfil
  \begin{tabular}{S<{{}\,\si{\kg}}}S<{{}\,\si{\kg}}}}
```

[41] For the TeX experts, the issue here is that the system to gather up cell contents is added in using the < construction. Normally, this comes after the cell contents and any other < arguments, so collects the user additions. However, in the last cell the contents include `\\`, which is converted to `\cr` before gathering can occur. By using `\cr` directly, the gathering process receives all of the cell contents as normal.

Table 21: Correcting spacing in last S column

Long header	Long header
1.23 kg	1.23 kg
4.56 kg	4.56 kg
7.8 kg	7.8 kg

Long header	Long header
1.23 kg	1.23 kg
4.56 kg	4.56 kg
7.8 kg	7.8 kg

```
    \toprule
    \multicolumn{1}{c}{Long header} &
      \multicolumn{1}{c}{Long header} \\
    \midrule
    1.23 & 1.23 \\
    4.56 & 4.56 \\
    7.8  & 7.8  \\
    \bottomrule
  \end{tabular}
  \hfil
  \begin{tabular}{S<{{\,\si{\kg}}}S<{{\,\si{\kg}}}}
    \toprule
    \multicolumn{1}{c}{Long header} &
      \multicolumn{1}{c}{Long header} \\
    \midrule
    1.23 & 1.23 \cr
    4.56 & 4.56 \cr
    7.8  & 7.8  \cr
    \bottomrule
  \end{tabular}
  \hfil
\end{table}
```

15 Reporting a problem

siunitx is quite long and complicated, and works hard to cover all possible eventualities. However, there will be bugs in the code and unexpected interactions with other packages. If you think you have found a bug, please report it. A short test-case demonstrating the problem would be very welcome. The following is a suitable template, and is available as `si-bug.ltx` in the `doc/latex/siunitx` directory or by running the `.dtx` or `.ins` file through TEX.

```
\listfiles
\documentclass{article}
```

Load other packages needed here.

```
\usepackage{siunitx}
```

Normally, debugging the load procedure will not be needed; the debug option here means that all run-time information is logged.

```
\sisetup{debug}
\begin{document}
This is the bug test-case document for the \textsf{siunitx}
```

```
package.\\
Please put your demonstration here, and e-mail to the package
author.
\begin{center}
  \texttt{joseph.wright@morningtar2.co.uk}
\end{center}
\end{document}
```

16 Feature requests

Feature requests for siunitx are welcome. The package maintainer will consider any ideas within the remit of the package (units and values). If suggesting a new feature, an example of how it should work would be appreciated. If new controls are needed, some suggestions for option names would be welcome.

17 Acknowledgements

Many thanks indeed to Stefan Pinnow, who has made a very large number of suggestions and found numerous bugs in the package. His contribution to the package has been vital. The package author has learned LaTeX tricks from far too many people to thank all of them. However, for this package specific thanks must go to the authors of the existing "unit" packages: Danie Els (SIstyle), Marcel Heldoorn (SIunits), Patrick Happel (unitsdef), Axel Reichert (units) and Harald Harders (numprint). Will Robertson and Heiko Oberdiek deserve much credit for demonstrating LaTeX coding best practice. Victor Eijkhout's excellent (and free) *TEX by Topic* has provided some useful coding hints [2]. The idea for combining and extending unit provision in LaTeX was heavily inspired by Philip Lehmann's biblatex. Thanks to the various contributors of ideas for the package: Donald Arseneau, Michele Dondi, Paul Gans, Ben Morrow, Lan Thuy Pham, Alan Ristow, Patrick Heinze, Andrea Blomenhofer, Morten Høgholm, Burkhard Moddemann and Patrick Steegstra.

Part III
Correct application of (SI) units

18 Background

Consistent and logical units are a necessity for scientific work, and have applicability everywhere. Historically, a number of systems have been used for physical units. SI units were introduced by the *Conférence Générale des Poids et Mesures* (CGPM) in 1960. SI units are a coherent system based on seven base units, from which all other units may be derived.

At the same time, physical quantities with units are mathematical entities, and as such way that they are typeset is important. In mathematics, changes of type (such as using bold, italic, sans serif typeface and so on) convey information. This means that rules exist not only for the type of units to be used under the SI system, but also the way they should appear in print. Advice on best practice has been made available by the *National Institute of Standards and Technology* (NIST) [3].

As befits an agreed international standard, the full rules are detailed. It is not appropriate to reproduce these in totality here; instead, a useful summary of the key points is provided. The full details are available from the *Bureau International des Poids et Mesures* (BIPM) in French [4] and English [5]. They also publish a very useful and detailed guide to using units, values and so on, available online in a number of different formats [6].

siunitx takes account of the information given here, so far as is possible. Thus the package defaults follow the recommendations made for typesetting units and values. Spacing and so forth is handled in such a way as to make implementing the rules (relatively) easy.

19 Units

19.1 SI base units

There are seven base SI units, listed in Table 5. Apart from the kilogram, these are defined in terms of a measurable physical quantity needing the definition alone.[42] The base units have been chosen such that all physical quantities can be expressed using an appropriative combination of these units, needing no others and with no redundancy. The kilogram is slightly different from the other base units as it is still defined in terms of a "prototype" held in Paris.[43]

19.2 SI derived units

All other units within the SI system are regarded as "derived" from the seven base units. At the most basic, all other SI units can be expressed as combinations of the base units. However, many units (listed in Table 7, Table 8 and Table 9)

[42] Some base units need others defined first; there is therefore a required order of definition.
[43] At the time of writing, this is under review and will be altered.

have a special name and symbol.[44] Most of these units are simple combinations of one or more base units (raised to powers as appropriate). A small number of units derived from experimental data are allowed as SI units (Table 8).

Some of these units (in Table 9) are regarded as only "temporarily" accepted, as the use of only the base and fully-consistent derived units in Table 7 is encouraged. They are accepted as they are in common use in one or more disciplines; some are still very widespread in the appropriate areas. These units are mainly multiples of base units (for example, a tonne is 1000 kg).

One point to note is that "unitless ratios" are regarded as having base units which cancel out. For example, the radian is regarded as having base unit $\mathrm{m\,m^{-1}}$. The result of this division ("1") is therefore regarded as a derived SI unit in this context.

19.3 SI prefixes

A series of SI prefixes for decimal multiples and submultiples are provided, and can be used as modifiers for any SI unit (either base or derived units) with the exception of the kilogram. The prefixes are listed in Table 6. No space should be used between a prefix and the unit, and only a single prefix should be used. Even the degree Celsius can be given a prefix, for example 1 m°C. The only exception to this rule is for degrees, minutes and seconds of an arc: 1°2′3″.

It is important to note that the kilogram is the only SI unit with a prefix as part of its name and symbol. Only single prefix may be used, and so in the case of the kilogram prefix names are used with the unit name "gram" and the prefix symbols are used with the unit symbol g. For example $1 \times 10^{-6}\,\mathrm{kg} = 1\,\mathrm{mg}$.

19.4 Other units

The application of SI units is meant to provide a single set of units which ensure consistency and clarity across all areas. However, other units are common is many areas, and are not without merit. The units provided by siunitx by default do not included any of these; only units which are part of the SI set or are accepted for use with SI units are defined. However, several other sets of units can be loaded as optional modules. The binary prefixes and units (Section 8.1 and Table 12) are the most obvious example. These are *not* part of the SI specifications, but the prefix names are derived from those in Table 6.

Other units (such as those provided by the modules `synchem`, `hep` and `astro`) are normally to be avoided where possible. SI units should, in the main, be preferred due to the advantages of clear definition and self-consistency this brings. However, there will probably always be a place for specialist or non-standard units. This is particulary true of units derived from basic physical constants; for example reason, the `hep` module defines the speed of light, c, as a unit. For work in basic science, a small number of physical constants are recognised as units provided the results for comparison with experiment are given in SI units.

There are also many areas where non-standard units are used so commonly that to do otherwise is difficult or impossible. For example, most synthetic che-

[44] The nautical mile has a given name but no agreed symbol, and although accepted by the SI is not provided by siunitx as a unit macro.

mists measure the pressure inside vacuum apparatus in mmHg, partly because the most common gauge for the task still uses a column of mercury metal. For these reasons, siunitx does define non-SI units.

20 Units and values in print

20.1 Mathematical meaning

As explained earlier, a unit–value combination is a single mathematical entity. This has implications for how both the number and the unit should be printed. Firstly, the two parts should not be separated. With the exception of the symbols for plane angles (°, ′ and ″), it is usual to have a space between the unit and the value. This should therefore be a non-breaking space between the two. Different geographical areas have different conventions on the size of this space; a "small" space (`\,`) is the siunitx default.

A space for 10 %
and also for 100 ℃
but not for 1.23°.

```
A space for \SI{10}{\percent}\\
and also for \SI{100}{\celsius}\\
but not for \ang{1.23}.
```

The mathematical meaning of units also means that the shape, weight and family are important. Units are supposed to be typeset in an upright, medium weight serif font. Italic, bold and sans serif are all used mathematically to convey other meanings. siunitx package defaults again follow this convention: any local settings are ignored, and uses the current upright serif maths font. However, there are occasions where this may not be the most desirable behaviour. A classic example would be in an all-bold section heading. As the surrounding text is bold, some people feel that any units should follow this.

Units should **not be bold:** 54 F
But perhaps in a running block,
it might look better: 54 F

```
Units should \textbf{not be bold: \SI{54}{\farad}}\\
\textbf{But perhaps in a running block,\\
it might look better: \SI[obeybold]{54}{\farad}}
```

20.2 Unit multiplication and division

Symbols for units formed from other units by multiplication are indicated by means of either a half-height (that is, centred) dot or a (thin) space. This document uses a half-height dot as (*i*) this is the recommendation of NIST, amongst others and (*ii*) it avoids potential confusion between unit prefixes and multiplied units.

m s = metre second
ms = millisecond
m s = metre second
ms = millisecond

```
$\si{\metre\second} = \mbox{metre second}$ \\
$\si{\milli\second} = \mbox{millisecond}$ \\
\sisetup{unitsep=thin}
$\si{\metre\second} = \mbox{metre second}$ \\
$\si{\milli\second} = \mbox{millisecond}$
```

There are some circumstances under which it is permissable to omit any spaces. The classic example is kWh, where "kW h" does not add any useful information. If using such a unit repeatedly, users of siunitx are advised to create a custom unit to ensure consistency.[45]

[45] `\kWh` and `\kilowatthour` are defined by siunitx in this way.

Symbols for units formed from other units by division are indicated by means of a virgule (oblique stroke, slash, /), a horizontal line, or negative exponents.[46] However, to avoid ambiguity, the virgule must not be repeated on the same line unless parentheses are used. This is ensured when using named unit macros in siunitx, which will "trap" repeated division and format it correctly. In complicated cases, negative exponents are to be preferred over other formats.

$\mathrm{J\,mol^{-1}\,K^{-1}}$
$\frac{\mathrm{J}}{\mathrm{mol\,K}}$
$\mathrm{J/(mol\,K)}$

```
\si{\joule\per\mole\per\kelvin}\\
\si[per=fraction]{\joule\per\mole\per\kelvin}\\
\si[per=slash]{\joule\per\mole\per\kelvin}
```

20.3 Repeating units

Products and errors should show what unit applies to each value given. Thus $2\,\mathrm{m} \times 3\,\mathrm{m}$ is an ordered set of lengths of a geometric area, whereas $2 \times 3\,\mathrm{m}$ is a length (and equal to 6 m). Thus, $\times$ is not a product but is a mathematical operator; in the same way, a 2×3 matrix is not a 6 matrix! In some areas, areas and volumes are given with separated units but a unit raised to the appropriate power: $2 \times 3\,\mathrm{m}^2$. Although this does display the correct overall units, it is potentially-confusing and is not encouraged.

20.4 Clarity in writing values of quantities

Care must be taken when writing ranges of numbers. For purely numerical values, it is common to use a en-dash between values, for example "see pages 1–5". On the other hand, values with units could be misinterpret as negative values if written in this way. As the unit–value combination is a single mathematic entity, writing the values with an en-dash followed by a single unit is also incorrect. As a result, using the word "to" is strongly recommended.

1 m to 5 m long.

```
\SI{1}{\metre} to \SI{5}{\metre} long.
```

20.5 Graphs and tables

In tables and graphs, repetition of the units following each entry or axis mark is confusing and repetitive. It is therefore best to place the unit in the label part of the information. Placing the unit in square brackets is common but mathematically poor.[47] Much better is to show division of all values by the unit, which leaves the entries as unitless ratios. This is illustrated in Table 22 and Figure 1.

```
\begin{table}
  \centering
  \caption{An example of table labelling}
  \label{tab:label}
  \begin{tabular}{cS[tabformat=1.4,tabnumalign=centre]}
    \toprule
    {Entry} & {Length/\si{\metre}} \\
```

[46] Notice that a virgule and a solidus are not the same symbol.

[47] For example, for an acceleration a, the expression $[a]$ is the dimensions of a, *i.e.* length per time squared in this case.

Table 22: An example of table labelling

Entry	Length/m
1	1.1234
2	1.1425
3	1.7578
4	1.9560

```
    \midrule
    1 & 1.1234 \\
    2 & 1.1425 \\
    3 & 1.7578 \\
    4 & 1.9560 \\
    \bottomrule
  \end{tabular}
\end{table}


\begin{figure}
  \centering
  \begin{tikzpicture}
    \begin{axis}[xlabel=$t/\si{\second}$,ylabel=$d/\si{\metre}$]
      \addplot[smooth,mark=x]
        plot coordinates {
          (0,0)
          (1,5)
          (2,8)
          (3,9)
          (4,8)
          (5,5)
          (6,0)
         };
    \end{axis}
  \end{tikzpicture}
  \caption{An example of graph labelling}
  \label{fig:label}
\end{figure}
```

In most cases, adding exponent values in the body of a table is less desirable than adding a fixed exponent to column headers. An example is shown in Table 23. The use of `\multicolumn` is needed here due to the "<"; without `\multicolumn`, the titles are followed by "kg"!

```
\begin{table}
  \centering
  \caption{Good and bad columns}
  \label{tab:exp}
  \sisetup{tabnumalign=centre}
  \begin{tabular}{%
    c%
    S[tabformat=1.3e1]<{\,\si{\kilogram}}%
```

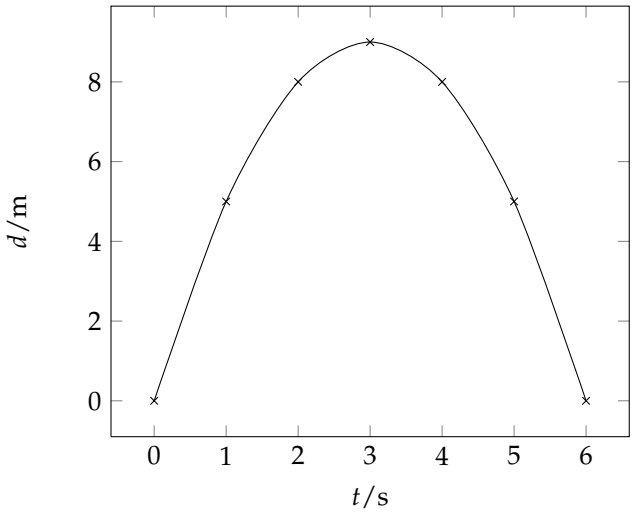

Figure 1: An example of graph labelling

Table 23: Good and bad columns

Entry	Mass	Mass/10^3 kg
1	4.56×10^3 kg	4.56
2	2.40×10^3 kg	2.40
3	1.345×10^4 kg	13.45
4	4.5×10^2 kg	0.45

```
    S[tabformat=2.2]}
    \toprule
    Entry & \multicolumn{1}{c}{Mass} &
      {Mass/\SI{e3}{\kilogram}} \\
    \midrule
    1 & 4.56e3  & 4.56 \\
    2 & 2.40e3  & 2.40 \\
    3 & 1.345e4 & 13.45 \\
    4 & 4.5e2   & 0.45 \\
    \bottomrule
  \end{tabular}
\end{table}
```

Part IV
Implementation

21 Main package

Much of the code here is taken, with little or no modification, from the existing packages. These are all released under the LPPL, and so this use is entirely allowed. Rather than confuse the source here with repeated references, note that code here could be copied from SIstyle, SIunits, numprint, unitsdef or units. Some ideas have also been borrowed from biblatex; again these will not be specifically noted. Code from other packages will be marked when used.

User-space commands (those not containing `@`) defined here should give the same result as macros with the same name in the older packages.[48] However, internal package macros will behave differently; if the user has redefined internal macros, then compatibility will be impaired.

The code used here uses LaTeX rather than TeX commands where possible.[49] For example, `\newcommand*` is used in place of `\def`, unless custom parameters are needed. Hopefully, this will aid future maintenance. Grouping is used where possible to limit the scope of temporary assignments.

21.1 Setup code

As always, the package starts with identification.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{siunitx}
  [2008/09/03 v1.0l A comprehensive (SI) units package]
```

The package requires ε-TeX, so the usual test is made.

```
\begingroup
  \@ifundefined{eTeXversion}
    {\PackageError{siunitx}
       {Not running under e-TeX}
       {This package requires e-TeX. Try compiling the document
        with\MessageBreak `elatex' instead of `latex'. When
        using pdfTeX, try `pdfelatex'\MessageBreak instead of
        `pdflatex'}%
     \endgroup\endinput}
    {\endgroup}
```

`\si@catcodes` There are circumstances under which some odd category codes might be in place. The category codes for `{`, `}`, `[`, `]` and `#` are assumed to be okay; if issues arise, this can be altered. LaTeX will have set `@` as a letter on loading siunitx.

```
\edef\si@catcodes{%
  \catcode\string`\string ` \the\catcode\string`\`\relax
  \catcode\string`\string = \the\catcode\string`\=\relax
  \catcode\string`\string ^ \the\catcode\string`\^\relax
  \catcode\string`\string ~ \the\catcode\string`\~\relax
  \catcode\string`\string : \the\catcode\string`\:\relax
```

[48] Although extra optional arguments may be added.

[49] This applies to LaTeX kernel commands only; for example, ifthenelse is not used.

```
  \catcode\string`\string - \the\catcode\string`\-\relax
  \catcode\string`\string + \the\catcode\string`\+\relax
  \catcode\string`\string ; \the\catcode\string`\;\relax
  \catcode\string`\string , \the\catcode\string`\,\relax
  \catcode\string`\string . \the\catcode\string`\.\relax}
\catcode\string`\` 12\relax
\catcode`\= 12\relax
\catcode`\^ 7\relax
\catcode`\~ \active\relax
\@makeother{\:}
\@makeother{\-}
\@makeother{\+}
\@makeother{\;}
\@makeother{\,}
\@makeother{\.}
```

Packages needed for functionality are loaded. xkeyval handles the package options, while amstext from the $\mathcal{AMS}$ bundle is needed for `\text`. array is needed for the new column type for tabular material. xspace provides "magic" spacing after macros, if requested.

```
\RequirePackage{xkeyval}[2005/05/07]
\RequirePackage{amstext,array,xspace}
```

`\si@tempa` `\si@tempb` `\si@tempc` Some scratch commands are defined; apart from where a known value is carried through, these could contain anything.

```
\newcommand*{\si@tempa}{}
\newcommand*{\si@tempb}{}
\newcommand*{\si@tempc}{}
```

`\ifsi@switch` Various items will need a switch. To avoid name pollution, a single switch is defined here; grouping will keep the definition local.

```
\newif\ifsi@switch
```

`\si@tempboxa` `\si@tempboxb` `\si@tempboxc` `\si@tempboxd` Some boxes are also needed.

```
\newbox\si@tempboxa
\newbox\si@tempboxb
\newbox\si@tempboxc
\newbox\si@tempboxd
```

`\si@temptoks` A token register is also handy.

```
\newtoks{\si@temptoks}
```

`\si@packagecheck` `\si@blockpkgs` `\si@checkpkgs` As siunitx is intended to replace the other unit-management packages, these are tested for before any further processing. If any are loaded, the package halts compilation; name clashes or unexpected results could occur if this is not tested. Notice that SIunits and SIstyle could be loaded with variable capitalisation (at least on Windows); both possibilities are tested.

```
\newcommand*{\si@blockpkgs}{SIunits,sistyle,siunits,SIstyle,%
  unitsdef,fancyunits}
\newcommand*{\si@checkpkgs}{units,hepunits,fancynum}
\newcommand*{\si@packagecheck}{%
  \begingroup
```

```
\@for\si@tempa:=\si@blockpkgs\do{
  \@ifpackageloaded{\si@tempa}
    {\PackageError{siunitx}
       {Package `\si@tempa' incompatible}
       {The \si@tempa\space package and siunitx are
        incompatible.\MessageBreak Use the
        `emulate=\si@tempa' package option when loading
        siunitx}}
    {}}
```

Some packages should not cause a clash, but are emulated and would be better handled that way.

```
\@for\si@tempa:=\si@checkpkgs\do{%
  \@ifpackageloaded{\si@tempa}
    {\PackageWarning{siunitx}
       {Consider loading the siunitx package
        with\MessageBreak option `emulate=\si@tempa', rather
        than\MessageBreak loading both \si@tempa\space and
        siunitx}}
    {}}
\endgroup}
```

The check is carried out on loading and at the beginning of the document, so that packages loaded both before and after siunitx are caught.

```
\si@packagecheck
\AtBeginDocument{\si@packagecheck}
```

`\si@ifdefinable` `#1` : macro

Using `\@ifdefinable` to check macro definitions gives a generic error. To give something more helpful, `\@ifundefined` is used, but this needs some `\expandafter` work. This way it can also be used as a form of `\@ifundefined` for macro names.

```
\newcommand*{\si@ifdefinable}[1]{%
  \expandafter\expandafter\expandafter\@ifundefined%
    \expandafter\expandafter\expandafter%
      {\expandafter\@gobble\string#1}}
```

`\si@addtolist` `#1` : macro
`#2` : items

It is quite useful to be able to add to a comma-separated list of expandable items.

```
\newcommand*{\si@addtolist}[2]{%
  \ifx\@empty#1\@empty
    \edef#1{#2}%
  \else
    \edef#1{#1,#2}%
  \fi}
```

`\si@addtocsname` `#1` : csname
`#2` : tokens

A second item to add to a command sequence.

```
\newcommand*{\si@addtocsname}[2]{%
  \@ifundefined{#1}
    {\expandafter\gdef\csname #1\endcsname{#2}}
    {\si@temptoks\expandafter\expandafter\expandafter{%
```

```
      \csname #1\endcsname#2}%
    \expandafter\xdef\csname #1\endcsname{\the\si@temptoks}}}
```

\si@ifmtarg
\si@xifmtarg
\si@ifnotmtarg

To keep down dependance on other packages, the very short code block from ifmtarg is copied here with an internal name.

```
\begingroup
  \catcode`\Q=3
  \long\gdef\si@ifmtarg#1{%
    \si@xifmtarg#1QQ\@secondoftwo\@firstoftwo\@nil}
  \long\gdef\si@xifmtarg#1#2Q#3#4#5\@nil{#4}
  \long\gdef\si@ifnotmtarg#1{%
    \si@xifmtarg#1QQ\@firstofone\@gobble\@nil}
\endgroup
```

\si@newrobustcmd
\si@newcommand
\si@newcmd
\si@xargdef

#1 : macro
[#2]: num-args
[#3]: default
#4 : def

Some more copying, this time from etoolbox. Various macros need to be *really* robust. This is achieved using the ε-TEX `\protected` primitive in various places. However, it would be nice to have a `\protected` version of `\newcommand`. etoolbox has code for that, but to avoid needing to load it, the necessary stuff is copied here. The only changes form the original are names, and the use of `\newcommand*` for the `\si@newcommand` macro.

```
\@ifpackageloaded{etoolbox}
  {\let\si@newrobustcmd\newrobustcmd}
  {\protected\def\si@newrobustcmd{%
     \@ifstar
       {\let\l@ngrel@x\protected\si@newcommand}
       {\def\l@ngrel@x{\protected\long}\si@newcommand}}
   \newcommand*{\si@newcommand}[1]{\@testopt{\si@newcmd#1}0}
   \def\si@newcmd#1[#2]{%
     \@ifnextchar[%]
       {\si@xargdef#1[#2]}
       {\@argdef#1[#2]}}
   \long\def\si@xargdef#1[#2][#3]#4{%
     \@ifdefinable#1{%
       \expandafter\protected
       \expandafter\def
       \expandafter#1%
       \expandafter{%
         \expandafter\@testopt
         \csname\string#1\endcsname{#3}}%
       \expandafter\@yargdef
       \csname\string#1\endcsname\tw@{#2}{#4}}}}
```

21.2 Logging

\ifsi@debug
\ifsi@logmin
\ifsi@lognone

To control logging, some new switches are declared.

```
\newif\ifsi@debug
\newif\ifsi@logmin
\newif\ifsi@lognone
```

`\si@log@err` `#1` : error
`#2` : explanation
Some handy re-usable macros are defined here. These all take names beginning
These pop up in various places. First errors are handled.

```
\newcommand*{\si@log@err}[2]{%
  \ifsi@lognone\else
    \ifsi@logmin
      \PackageWarning{siunitx}{#1}%
    \else
      \PackageError{siunitx}{#1}{#2}%
    \fi
  \fi}
```

`\si@log@warn` `#1` : text
`\si@log@inf` Then warnings and information.

```
\newcommand*{\si@log@warn}[1]{%
  \ifsi@lognone\else
    \ifsi@logmin\else
      \PackageWarning{siunitx}{#1}%
    \fi
  \fi}
\newcommand*{\si@log@inf}[1]{%
  \ifsi@lognone\else
    \ifsi@logmin\else
      \PackageInfo{siunitx}{#1}%
    \fi
  \fi}
```

`\si@log@debug` `#1` : debug-info
The debug macro only gives output if the appropriate package option is set.

```
\newcommand*{\si@log@debug}[1]{%
  \ifsi@lognone\else
    \ifsi@debug
      \PackageInfo{siunitx}{#1}%
    \fi
  \fi}
```

21.3 String comparison

`\si@str@ifchrstr` `#1` : char
`\si@str@chrstr` `#2` : string
At various points, the package needs to compare two strings, to find if one occurs in the other. The first test is if a single character is part of a second string; this is used, for example, to check that a character is valid as input. The first argument is not expanded further, but the second is two allow division into individual units.

```
\newcommand*{\si@str@ifchrstr}[2]{%
  \begingroup
    \si@switchfalse
    \renewcommand*{\si@tempa}{#1}%
    \protected@edef\si@tempb{#2}%
    \expandafter\si@str@chrstr\si@tempb\@empty\@empty\@empty
```

```
    \ifsi@switch
      \aftergroup\@firstoftwo
    \else
      \aftergroup\@secondoftwo
    \fi
  \endgroup}
\def\si@str@chrstr#1#2\@empty{%
  \renewcommand*{\si@tempc}{#1}%
  \ifx\si@tempa\si@tempc
    \expandafter\si@switchtrue
  \else
    \ifx\@empty#2\@empty\else
      \si@str@chrstr#2\@empty\@empty
    \fi
  \fi}
```

`\si@str@ifonlychrs` `\si@str@onlychrs` `#1` : string
`#2` : chars

The second test builds on the first. Here, a check is made to see if the first string is made up only of characters from the second string. In this case, the first string is expanded before testing. The second string will be expanded by the internal character by character test.

```
\newcommand*{\si@str@ifonlychrs}[2]{%
  \begingroup
    \si@switchtrue
    \protected@edef\si@tempa{#1}%
    \renewcommand*{\si@tempb}{#2}%
    \expandafter\si@str@onlychrs\si@tempa\@empty\@empty\@empty
    \ifsi@switch
      \aftergroup\@firstoftwo
    \else
      \aftergroup\@secondoftwo
    \fi
  \endgroup}
\def\si@str@onlychrs#1#2\@empty{%
  \si@str@ifchrstr{#1}{\si@tempb}
    {}{\si@switchfalse}%
  \ifx\@empty#2\@empty\else
    \si@str@onlychrs#2\@empty\@empty
  \fi}
```

21.4 Option handling

`\sisetup` `#1` : options

To allow modification of options at run time, a setup macro is provided.

```
\newcommand*{\sisetup}{\setkeys[si]{key}}
```

`\si@opt@key` `#1` : keyname
`#2` : code

To aid maintenance, some shortcuts are defined for generating keys. These also allow the debugging messages to be added automatically to every key. First of all the basic key definition.

```
\newcommand*{\si@opt@key}[2]{%
  \define@key[si]{key}{#1}
  {#2\si@log@debug{Option #1 set to ##1}}}
```

`\si@opt@cmdkey [#1]` : default
`#2` : keyname
`#3` : function

For a single command key, a function must be specified.

```
\newcommand*{\si@opt@cmdkey}[3][]{%
  \define@cmdkey[si]{key}[si@]{#2}[#1]{#3}}
```

`\si@opt@cmdkeys [#1]` : default
`#2` : keyname

Whereas multiple definitions do not take a function.

```
\newcommand*{\si@opt@cmdkeys}[2][]{%
  \define@cmdkeys[si]{key}[si@]{#2}[#1]}
```

`\si@opt@boolkey [#1]` : function
`#2` : keyname

Keys which only take switch values; anything other than `true` or `false` will generate a warning from xkeyval.

```
\newcommand*{\si@opt@boolkey}[2][]{%
  \define@boolkey[si]{key}[si@]{#2}[true]
  {#1\si@log@debug{Option #2 set to ##1}}}
```

`\si@opt@choicekey [#1]` : default
`#2` : keyname
`#3` : choices
`#4` : function

A "fill in the blanks" choice key. In all cases, `\si@tempa` is used to hold the value given to the key, so that `\ifx` testing can occur.

```
\newcommand*{\si@opt@choicekey}[4][]{%
  \define@choicekey*+[si]{key}{#2}[\si@tempa]{#3}[#1]
    {#4\si@log@debug{Option #2 set to ##1}}
    {\si@log@warn{Unknown value `##1' for option #2}}}
```

`\si@opt@xchoicekey` `#1` : keyname
`#2` : choices
`#3` : initial-value

Several of the package options can take either a choice from a list of known options, or a value to be interpreted literally. To aid maintenance, the necessary code can be set up here. These keys all define a new macro, which must exist. The `\si@opt@xchoicekey` macro therefore ensures that this is defined, as well as setting up the xkeyval key.

```
\newcommand*{\si@opt@xchoicekey}[3]{%
  \define@choicekey*+[si]{key}{#1}[\si@tempa]{#2}[#1]
```

This code will execute if the option is on the list. There will be a "fixed" macro with a matching name, which is used for this.

```
    {\si@log@debug{Option #1 set to ##1}%
     \expandafter\renewcommand\expandafter*\expandafter{%
      \csname si@#1\endcsname}{\@nameuse{si@fix@##1}}}
```

The user has given something that is not on the list as an argument. It is used literally.

```
    {\si@log@debug{Option #1 set to ##1}%
     \expandafter\renewcommand\expandafter*\expandafter{%
      \csname si@#1\endcsname}{##1}}
```

Finally, the initial value of the macro is set up.

```
  \expandafter\newcommand\expandafter*\expandafter{%
    \csname si@#1\endcsname}%
    {\@nameuse{si@fix@#3}}}
```

`\si@opt@compatkey` `#1` : package
`#2` : keyname

An all-in-one definition for a back-compatibility key. These should only be used at load time, so are automatically disabled once the package is loaded. Emulation is also automatically turned on.

```
\newcommand*{\si@opt@compatkey}[2]{%
  \define@boolkey[si]{key}[si@old@]{#2}[true]
  {\si@log@debug{Emulating #1 package option\MessageBreak #2}%
   \sisetup{emulate=#1}%
   \si@log@debug{Option #2 set to ##1}}
   \AtEndOfPackage{\si@opt@disablekey{#2}
     {Compatibility option #2 only\MessageBreak
        available when loading siunitx package}}}
```

`\si@opt@disablekey` `#1` : keyname
`#2` : text

The ability to disable a key with a meaningful message is a must; the warning will come from siunitx, and not from xkeyval.

```
\newcommand*{\si@opt@disablekey}[2]{%
  \key@ifundefined[si]{key}{#1}
    {}
    {\si@log@debug{Disabling key #1}%
     \si@opt@key{#1}{\si@log@warn{#2}}}}
```

The xkeyval package option for logging is declared. This is then processed to set the switches correctly.

```
\si@opt@choicekey[normal]{log}{debug,verbose,normal,errors,none}
```

A series of comparisons are made to assign the logging mode. The `normal` option is not tested, as executing the option sets the switches appropriately.

```
  {\si@debugfalse
   \si@logminfalse
   \si@lognonefalse
   \renewcommand*{\si@tempb}{none}%
   \ifx\si@tempa\si@tempb
     \si@lognonetrue
   \fi
   \renewcommand*{\si@tempb}{minimal}%
   \ifx\si@tempa\si@tempb
     \si@logmintrue
   \fi
   \renewcommand*{\si@tempb}{debug}%
```

```
    \ifx\si@tempa\si@tempb
      \si@debugtrue
    \fi
    \renewcommand*{\si@tempb}{verbose}%
    \ifx\si@tempa\si@tempb
      \si@debugtrue
    \fi}
```

A quick method to set `log=debug`.

```
\si@opt@boolkey{debug}
```

`\ifsi@strict` It would be useful to be able to disable some keys, when strict interpretation of the rules is desired. This is a load-time option, and has to disable various options.

```
\si@opt@boolkey[%
  \ifsi@strict
    \sisetup{
      obeymode=false,
      obeybold=false,
      obeyitalic=false,
      mode=maths,
      repeatunits=true,
      trapambigerr=true,
      trapambigfrac=true}
    \@for\si@tempa:=obeyall,obeymode,obeyitalic,mode,unitmode,%
      valuemode,textmode,obeybold,repeatunits,trapambigerr,%
      trapambigfrac\do{%
      \begingroup
        \edef\si@tempb{\endgroup
          \noexpand\si@opt@disablekey{\si@tempa}
          {Option `\si@tempa' forbidden in strict mode}}%
      \si@tempb}
  \fi]{strict}
\AtEndOfPackage{
  \si@opt@disablekey{strict}
    {Option `strict' only available when\MessageBreak
     loading package}}
```

`\si@emulate` The `emulate` option is used for back-compatibility mode; the option is only valid when loading siunitx.

```
\newcommand*{\si@emulate}{}
\si@opt@key{emulate}{\si@addtolist{\si@emulate}{#1}}
\AtEndOfPackage{
  \si@opt@disablekey{emulate}
    {Option `emulate' only available when\MessageBreak
     loading package}}
```

`\si@unitsep` `\si@unitspace` `\si@errspace` `\si@valuesep` The two `...sep` options control the size of spaces between the number and the unit (`\si@valuesep`), and that used to represent a product (`\si@unitsep`). Known values here are `thin`, `med`, `medium`, `thick`, `cdot`, `tightcdot`[50] and `none`;[51] other entries will be treated as custom spaces.

[50] Both `\cdot`-based options only valid for `unitsep`.

[51] Only valid for `valuesep`.

```
\si@opt@xchoicekey{unitsep}
  {thin,med,medium,thick,none,comma,stop,fullstop,period,
    times,tighttimes,cdot,tightcdot}{thin}
\si@opt@xchoicekey{unitspace}{space,thin,med,medium,thick,
  none}{thin}
\si@opt@xchoicekey{errspace}{space,thin,med,medium,thick,
  none}{none}
\si@opt@xchoicekey{valuesep}
  {thin,med,medium,thick,none,comma,stop,fullstop,period,
    times,tighttimes,cdot,tightcdot}{thin}
```

`\si@digitsep` Separation of digits in large numbers is controlled by the `digitsep` option. As with the other `sep` values, this one has a choice of possible values. The list is quite long, so that a range of options are handled automatically. Notice that `digitsep=none` will be used for no separation at all.

```
\si@opt@xchoicekey{digitsep}
  {thin,med,medium,thick,none,comma,stop,fullstop,period,
    times,tighttimes,cdot,tightcdot}{thin}
```

`\si@decimalsymbol` The symbol used for the decimal position is varied here. There are only two real options, but options are given for the name of a full stop.

```
\si@opt@xchoicekey{decimalsymbol}{comma,stop,fullstop,period,
  cdot,tightcdot}{fullstop}
```

`\si@anglesep` The separator between degrees and minutes, and between minutes and seconds, when using `\ang`.

```
\si@opt@xchoicekey{anglesep}
    {thin,med,medium,thick,none,comma,stop,fullstop,period,
      times,tighttimes,cdot,tightcdot}{none}
```

`\ifsi@obeymode` The first test for the font control is whether to respect the surrounding maths or text mode.

```
\si@opt@boolkey{obeymode}
```

`\ifsi@numtextmode` `\ifsi@unittextmode` The output of the package can be typeset using either text or maths mode fonts. Two switches are needed, for numbers and units.

```
\newif\ifsi@numtextmode
\newif\ifsi@unittextmode
```

The `textmode` option has to set both flags.

```
\si@opt@choicekey[true]{textmode}{true,false}
  {\si@numtextmodefalse
   \si@unittextmodefalse
   \renewcommand*{\si@tempb}{true}%
   \ifx\si@tempa\si@tempb
     \si@numtextmodetrue
     \si@unittextmodetrue
   \fi}
```

The `mode` option applies to numbers and units.

```
\si@opt@choicekey{mode}{math,maths,text}
  {\si@numtextmodefalse
   \si@unittextmodefalse
```

```
    \renewcommand*{\si@tempb}{text}%
    \ifx\si@tempa\si@tempb
      \si@numtextmodetrue
      \si@unittextmodetrue
    \fi}
```

Now the two options for numbers or units alone.

```
\si@opt@choicekey{valuemode}{math,maths,text}
  {\si@numtextmodefalse
   \renewcommand*{\si@tempb}{text}%
   \ifx\si@tempa\si@tempb
     \si@numtextmodetrue
   \fi}
\si@opt@choicekey{unitmode}{math,maths,text}
  {\si@unittextmodefalse
   \renewcommand*{\si@tempb}{text}%
   \ifx\si@tempa\si@tempb
     \si@unittextmodetrue
   \fi}
```

`\ifsi@obeyfamily` The package can work to match the font family (serif, sans serif, typewriter) of the surrounding text. This is controlled by a Boolean option.

```
\si@opt@boolkey{obeyfamily}
```

`\ifsi@obeybold` The package can attempt to respect bold, or may ignore it.

```
\si@opt@boolkey{obeybold}
```

`\ifsi@inlinebtext` For inline maths, two options for checking what is bold are available, the maths environment (*i.e.* `\boldmath`) and the surrounding text (`\textbf` or `\bffamily`).

```
\newif\ifsi@inlinebtext
\si@opt@choicekey{inlinebold}{text,maths,math}
  {\si@inlinebtextfalse
   \renewcommand*{\si@tempb}{text}%
   \ifx\si@tempa\si@tempb
     \si@inlinebtexttrue
   \fi}
```

`\ifsi@obeyitalic` Italic is slightly different to bold, as there is no convenient switch for maths. Thus a choice key is used, with the appropriate check.

```
\si@opt@boolkey{obeyitalic}
```

`\ifsi@detectdisplay` For handling display mathematics, a setting is needed for whether to treat it differently from other maths.

```
\si@opt@boolkey{detectdisplay}
```

The option to obey all font switching commands is Boolean-like but needs alternative setup.

```
\si@opt@choicekey[true]{obeyall}{true,false}
  {\si@obeyboldfalse
   \si@obeyitalicfalse
   \si@obeymodefalse
   \si@obeyfamilyfalse
   \renewcommand*{\si@tempb}{true}%
```

```
339   \ifx\si@tempa\si@tempb
340     \si@obeyboldtrue
341     \si@obeyitalictrue
342     \si@obeymodetrue
343     \si@obeyfamilytrue
344   \fi}
```

\si@valuemathsrm
\si@valuemathssf
\si@valuemathstt
\si@unitmathsrm
\si@unitmathssf
\si@unitmathstt

The fonts used by the package default to the obvious LaTeX ones; however, this needs to be exposed to user modification. First the maths mode fonts are sorted out.

```
345 \si@opt@cmdkeys{valuemathsrm,valuemathssf,valuemathstt}
346 \si@opt@cmdkeys{unitmathsrm,unitmathssf,unitmathstt}
```

To make life easier for the user, US spellings are provided for the maths keys.

```
347 \si@opt@key{valuemathrm}{\sisetup{valuemathsrm=#1}}
348 \si@opt@key{valuemathsf}{\sisetup{valuemathssf=#1}}
349 \si@opt@key{valuemathtt}{\sisetup{valuemathstt=#1}}
350 \si@opt@key{unitmathrm}{\sisetup{unitmathsrm=#1}}
351 \si@opt@key{unitmathsf}{\sisetup{unitmathssf=#1}}
352 \si@opt@key{unitmathtt}{\sisetup{unitmathstt=#1}}
```

The combined options are given, for setting numbers and units at the same time.

```
353 \si@opt@key{mathsrm}{\sisetup{valuemathsrm=#1,unitmathsrm=#1}}
354 \si@opt@key{mathssf}{\sisetup{valuemathssf=#1,unitmathssf=#1}}
355 \si@opt@key{mathstt}{\sisetup{valuemathstt=#1,unitmathstt=#1}}
356 \si@opt@key{mathrm}{\sisetup{valuemathsrm=#1,unitmathsrm=#1}}
357 \si@opt@key{mathsf}{\sisetup{valuemathssf=#1,unitmathssf=#1}}
358 \si@opt@key{mathtt}{\sisetup{valuemathstt=#1,unitmathstt=#1}}
```

\si@valuetextrm
\si@valuetextsf
\si@valuetexttt
\si@unittextrm
\si@unittextsf
\si@unittexttt

The same thing for text mode fonts. Once again the default values are pretty obvious.

```
359 \si@opt@cmdkeys{valuetextrm,valuetextsf,valuetexttt}
360 \si@opt@cmdkeys{unittextrm,unittextsf,unittexttt}
361 \si@opt@key{textrm}{\sisetup{unittextrm=#1,valuetextrm=#1}}
362 \si@opt@key{textsf}{\sisetup{unittextsf=#1,valuetextsf=#1}}
363 \si@opt@key{texttt}{\sisetup{unittexttt=#1,valuetexttt=#1}}
```

\si@numdigits
\si@numdecimal
\si@numexp
\si@numprod
\si@numgobble
\si@numsign

The list of possible valid characters for parsing numbers is set up. This is similar to numprint, but with the `extra` class, and with characters ignored with no output renamed as `gobble`.

```
364 \si@opt@cmdkeys{numdigits,numdecimal,numexp,numgobble,numsign,%
365   numcloseerr,numopenerr,numaddn,numprod}
```

\si@numvalid
\si@numcloseerr
\si@numopenerr
\si@numextra
\si@numaddn

The various valid characters are collected together in a single macro for later. In common with the above macros, this one starts `\si@num...`. The order here is the order the values are tested later on.

```
366 \newcommand*{\si@numextra}{%
367   \si@numopenerr\si@numcloseerr\si@numaddn}
368 \newcommand*{\si@numvalid}{\si@numgobble\si@numexp\si@numsign
369   \si@numdecimal\si@numdigits\si@numextra\si@numprod}
```

\ifsi@seperr
\ifsi@trapambigerr
\si@openerr
\si@closeerr

An option to control whether numerical error values are printed with or separate from the number.

```
\si@opt@boolkey{seperr}
\si@opt@boolkey{trapambigerr}
\si@opt@cmdkeys{openerr,closeerr}
```

`\ifsi@sepfour` With four digits in a number, separating may or may not be desired. Note that this option is the same as one for numprint.

```
\si@opt@boolkey{sepfour}
```

`\ifsi@retainplus` An option to keep an explicit positive sign.

```
\si@opt@boolkey{retainplus}
```

`\si@expbase` The options for exponents are set up.

`\si@expproduct`

```
\si@opt@xchoicekey{expproduct}{times,tighttimes,
  cdot,tightcdot}{times}
\si@opt@xchoicekey{expbase}{ten}{ten}
```

`\ifsi@allowzeroexp` The package normally prevents 10^0.

```
\si@opt@boolkey{allowzeroexp}
```

`\si@prefixproduct` The marker for multiplication in prefixes.

```
\si@opt@xchoicekey{prefixproduct}{times,tighttimes,cdot,
  tightcdot,none}{times}
```

`\si@prefixbase` In the same area, the power for prefixes is variable. Here, two choices ar needed.

```
\si@opt@xchoicekey{prefixbase}{ten,two}{ten}
```

`\ifsi@prefixsymbolic` Unit prefixes can be given as either symbols or numerically.

```
\si@opt@boolkey{prefixsymbolic}
```

`\ifsi@num@padlead` `\ifsi@num@padtrail` A setting is needed to indicate when to add zeros to decimal numbers, either before the decimal marker (`.1` giving "0.1") or after (`1.` giving "1.0").

```
\newif\ifsi@num@padlead
\newif\ifsi@num@padtrail
\si@opt@choicekey[all]{padnumber}
  {leading,lead,trailing,trail,all,both,true,none,false}
  {\si@num@padleadfalse
   \si@num@padtrailfalse
   \renewcommand*{\si@tempb}{leading}%
   \ifx\si@tempa\si@tempb
     \si@num@padleadtrue
   \fi
   \renewcommand*{\si@tempb}{lead}%
   \ifx\si@tempa\si@tempb
     \si@num@padleadtrue
   \fi
   \renewcommand*{\si@tempb}{trailing}%
   \ifx\si@tempa\si@tempb
     \si@num@padtrailtrue
   \fi
   \renewcommand*{\si@tempb}{trail}%
   \ifx\si@tempa\si@tempb
     \si@num@padtrailtrue
   \fi
```

```
    \renewcommand*{\si@tempb}{all}%
    \ifx\si@tempa\si@tempb
      \si@num@padleadtrue
      \si@num@padtrailtrue
    \fi
    \renewcommand*{\si@tempb}{true}%
    \ifx\si@tempa\si@tempb
      \si@num@padleadtrue
      \si@num@padtrailtrue
    \fi
    \renewcommand*{\si@tempb}{both}%
    \ifx\si@tempa\si@tempb
      \si@num@padleadtrue
      \si@num@padtrailtrue
    \fi}
```

`\si@sign` Some new switches for adding signs to numbers

`\ifsi@num@signmant` `\ifsi@num@signexp`

```
\newif\ifsi@num@signmant
\newif\ifsi@num@signexp
```

Signs can be added to numbers by default. Two options are needed here; whether to add a sign by default, and what the sign is.

```
\si@opt@xchoicekey{sign}{plus,minus,pm,mp}{plus}
\si@opt@choicekey[all]{addsign}
  {mantissa,exponent,mant,exp,all,both,true,none,false}
```

The option is now processed.

```
  {\si@num@signmantfalse
   \si@num@signexpfalse
   \renewcommand*{\si@tempb}{mantissa}%
   \ifx\si@tempa\si@tempb
     \si@num@signmanttrue
   \fi
   \renewcommand*{\si@tempb}{mant}%
   \ifx\si@tempa\si@tempb
     \si@num@signmanttrue
   \fi
   \renewcommand*{\si@tempb}{exponent}%
   \ifx\si@tempa\si@tempb
     \si@num@signexptrue
   \fi
   \renewcommand*{\si@tempb}{exp}%
   \ifx\si@tempa\si@tempb
     \si@num@signexptrue
   \fi
   \renewcommand*{\si@tempb}{all}%
   \ifx\si@tempa\si@tempb
     \si@num@signmanttrue
     \si@num@signexptrue
   \fi
   \renewcommand*{\si@tempb}{true}%
   \ifx\si@tempa\si@tempb
     \si@num@signmanttrue
     \si@num@signexptrue
```

```
    \fi
    \renewcommand*{\si@tempb}{both}%
    \ifx\si@tempa\si@tempb
      \si@num@signmanttrue
      \si@num@signexptrue
    \fi}
```

`\ifsi@tightpm` To reduce spacing, it might be necessary to use a "tight" $\pm$ sign.

`\si@pm`

```
\si@opt@boolkey{tightpm}
\newcommand*{\si@pm}{%
  \ifsi@tightpm
    \si@fix@tightpm
  \else
    \si@fix@pm
  \fi}
```

`\ifsi@ang@padsmall` `\ifsi@ang@padlarge` A switch for determining whether to typeset `\ang{;;1}` as 0°0′1″ or 1″. First, two new Boolean switches are needed to indicate padding.

```
\newif\ifsi@ang@padsmall
\newif\ifsi@ang@padlarge
\si@opt@choicekey[all]{padangle}
  {small,large,all,both,true,none,false}
  {\si@ang@padsmallfalse
   \si@ang@padlargefalse
   \renewcommand*{\si@tempb}{small}%
   \ifx\si@tempa\si@tempb
     \si@ang@padsmalltrue
   \fi
   \renewcommand*{\si@tempb}{large}%
   \ifx\si@tempa\si@tempb
     \si@ang@padlargetrue
   \fi
   \renewcommand*{\si@tempb}{all}%
   \ifx\si@tempa\si@tempb
     \si@ang@padsmalltrue
     \si@ang@padlargetrue
   \fi
   \renewcommand*{\si@tempb}{true}%
   \ifx\si@tempa\si@tempb
     \si@ang@padsmalltrue
     \si@ang@padlargetrue
   \fi
   \renewcommand*{\si@tempb}{both}%
   \ifx\si@tempa\si@tempb
     \si@ang@padsmalltrue
     \si@ang@padlargetrue
   \fi}
```

`\ifsi@ang@toarc` `\ifsi@ang@todec` To control whether angles are formatted as decimals or degrees–minutes–seconds, a package option plus two switches are needed. The later format is referred to as "`arc`" most readily. An option to leave the input unchanged is also provided.

```
\newif\ifsi@ang@toarc
\newif\ifsi@ang@todec
\si@opt@choicekey[all]{angformat}
```

```
  {dec,decimal,arc,dms,unchanged,none}
  {\si@ang@toarcfalse
   \si@ang@todecfalse
   \renewcommand*{\si@tempb}{dec}%
   \ifx\si@tempa\si@tempb
     \si@ang@todectrue
   \fi
   \renewcommand*{\si@tempb}{decimal}%
   \ifx\si@tempa\si@tempb
     \si@ang@todectrue
   \fi
   \renewcommand*{\si@tempb}{arc}%
   \ifx\si@tempa\si@tempb
     \si@ang@toarctrue
   \fi
   \renewcommand*{\si@tempb}{dms}%
   \ifx\si@tempa\si@tempb
     \si@ang@toarctrue
   \fi}
```

`\ifsi@astroang` A slightly odd option to allow the method used by astronomers for angles.

```
\si@opt@boolkey{astroang}
```

`\ifsi@strictarc` For the `\ang` macro, the default is to require two semi-colons in the input for arc angles. This is controlled here.

```
\si@opt@boolkey{strictarc}
```

`\ifsi@tab@fixed` `\si@tabnumalign` `\si@tab@rfill@S` `\si@tab@lfill@S` To determine the control of table alignment, two options are provided. The `tabnumalign` option controls which centring method is used, and the fills used for achieving this.

```
\newif\ifsi@tab@fixed
\si@opt@choicekey{tabnumalign}
  {centredecimal,centerdecimal,right,left,centre,center}
  {\si@tab@fixedtrue
   \let\si@tab@rfill@S\hfil
   \let\si@tab@lfill@S\hfil
   \renewcommand*{\si@tempb}{right}%
   \ifx\si@tempa\si@tempb
     \let\si@tab@lfill@S\hfill
   \fi
   \renewcommand*{\si@tempb}{left}%
   \ifx\si@tempa\si@tempb
     \let\si@tab@rfill@S\hfill
   \fi
   \renewcommand*{\si@tempb}{centredecimal}%
   \ifx\si@tempa\si@tempb
     \expandafter\si@tab@fixedfalse
   \fi
   \renewcommand*{\si@tempb}{centerdecimal}%
   \ifx\si@tempa\si@tempb
     \expandafter\si@tab@fixedfalse
   \fi}
\si@opt@key{tabalign}{\sisetup{tabnumalign=#1,tabtextalign=#1,
  tabunitalign=#1}}
```

\ifsi@tabalignexp A switch for aligning exponents.

```
\si@opt@boolkey{tabalignexp}
```

\si@tab@mantprecnt To process the format information, various internal number-processing macros
\si@tab@mantpostcnt are used. First, some storage areas are created.
\si@tab@expprecnt
\si@tab@exppostcnt
\ifsi@tab@mantsign
\ifsi@tab@expsign

```
\newcount\si@tab@mantprecnt
\newcount\si@tab@mantpostcnt
\newcount\si@tab@expprecnt
\newcount\si@tab@exppostcnt
\newif\ifsi@tab@mantsign
\newif\ifsi@tab@expsign
```

The input is split into a mantissa and exponent, then passed to a re-useable macro for further processing.

```
\si@opt@cmdkey{tabformat}
  {\si@num@fixpm
   \renewcommand*{\si@num@arg}{tabformat data}%
   \renewcommand*{\si@num@exp}{}%
   \renewcommand*{\si@num@mant}{}%
   \si@tab@mantsignfalse
   \si@tab@expsignfalse
   \si@switchfalse
   \si@num@sepmantexp{#1}%
```

When checking for a sign, the internal flag for finding but deleting a plus sign is used.

```
   \si@num@sepsign{mant}%
   \ifx\@empty\si@num@mantsign\@empty
     \ifsi@num@delplus
       \expandafter\expandafter\expandafter\si@tab@mantsigntrue
     \fi
   \else
     \expandafter\si@tab@mantsigntrue
   \fi
   \si@num@sepsign{exp}%
   \ifx\@empty\si@num@expsign\@empty
     \ifsi@num@delplus
       \expandafter\expandafter\expandafter\si@tab@expsigntrue
     \fi
   \else
     \expandafter\si@tab@expsigntrue
   \fi
   \si@opt@proctform{mant}%
   \si@opt@proctform{exp}%
```

If both the integer and decimal parts are empty, then something has probably gone wrong.

```
   \ifnum\si@tab@mantpostcnt=\z@\relax
     \ifnum\si@tab@mantprecnt=\z@\relax
       \si@log@err{Empty mantissa argument for tabformat}
         {The argument '#1' contains no valid entry for
          a mantissa\MessageBreak It should be of the
          form 'm.n', where  m and n are integers}%
     \fi
   \fi
```

If `tabformat` has been given with `tabnumalign=centredecimal` active, then the alignment is changed to centred.

```
\ifsi@tab@fixed\else
  \sisetup{tabnumalign=centre}%
\fi
\let\pm\si@num@pm
\let\mp\si@num@mp}
```

`\si@opt@proctform` `#1` : either `mant` or `exp`

Processing the number further uses the `\si@num@digits` macro. The results are stored in the appropriate counter.

```
\newcommand*{\si@opt@proctform}[1]{%
  \renewcommand*{\si@num@predec}{}%
  \renewcommand*{\si@num@postdec}{}%
  \si@switchfalse
  \expandafter\si@ifnotmtarg\expandafter{%
    \csname si@num@#1\endcsname}
     {\expandafter\expandafter\expandafter\si@num@digits
        \csname si@num@#1\endcsname\@empty\@empty}%
   \csname si@tab@#1precnt\endcsname\z@\relax
   \csname si@tab@#1postcnt\endcsname\z@\relax
   \ifx\@empty\si@num@predec\@empty\else
     \csname si@tab@#1precnt\endcsname\si@num@predec\relax
   \fi
   \ifx\@empty\si@num@postdec\@empty\else
     \csname si@tab@#1postcnt\endcsname\si@num@postdec\relax
   \fi}
```

The alignment of tabular material when not processed by `\num` needs to be available.

```
\si@opt@choicekey{tabtextalign}{left,right,centre,center}
```

`\si@tab@rfill@t` `\si@tab@lfill@t` By default, centring happens on both sides of the content of tabular material.

```
{\let\si@tab@rfill@t\hfill
 \let\si@tab@lfill@t\hfill
 \renewcommand*{\si@tempb}{right}%
 \ifx\si@tempa\si@tempb
   \let\si@tab@rfill@t\relax
 \fi
 \renewcommand*{\si@tempb}{left}%
 \ifx\si@tempa\si@tempb
   \let\si@tab@lfill@t\relax
 \fi}
```

The alignment of unit columns for tabular material has a similar control option.

```
\si@opt@choicekey{tabunitalign}{left,right,centre,center}
```

`\si@tab@rfill@s` `\si@tab@lfill@s` By default, centring happens on both sides of the content of tabular material.

```
{\let\si@tab@rfill@s\hfill
 \let\si@tab@lfill@s\hfill
 \renewcommand*{\si@tempb}{right}%
 \ifx\si@tempa\si@tempb
   \let\si@tab@rfill@s\relax
```

```
    \fi
    \renewcommand*{\si@tempb}{left}%
    \ifx\si@tempa\si@tempb
      \let\si@tab@lfill@s\relax
    \fi}
```

`\ifsi@fixdp` `\si@num@dp` To allow control of rounding, two options are needed. One sets how many fixed digits to use, the second turns this function on and off.

```
\si@opt@boolkey{fixdp}
\newcount\si@num@dp
\si@opt@key{dp}{%
  \si@str@ifonlychrs{#1}{0123456789}
    {}
    {\si@log@err{Invalid input for dp option}
      {The dp option must be given a positive integer}}%
  \si@num@dp#1\relax
  \si@fixdptrue}
```

`\ifsi@tabautofit` To apply rounding automatically in a table, a separate option is used.

```
\si@opt@boolkey{tabautofit}
```

`\ifsi@xspace` Unit macros on their own may need xpsace.

```
\si@opt@boolkey{xspace}
```

`\ifsi@prespace`

```
\si@opt@boolkey
  [\si@unt@numfalse
   \ifsi@prespace
     \si@unt@numtrue
  \fi]
  {prespace}
```

`\ifsi@allowoptarg` For unitsdef users, a method to absorb optional arguments is needed.

```
\si@opt@boolkey{allowoptarg}
```

`\ifsi@frac` `\ifsi@slash` `\ifsi@stickyper` The option processing for formatting units with `\per` in them needs two switches.

```
\newif\ifsi@slash
\newif\ifsi@frac
\si@opt@boolkey{stickyper}
\si@opt@choicekey[reciprocal]{per}
  {reciprocal,rp,power,slash,frac,fraction}
  {\si@slashfalse
   \si@fracfalse
   \renewcommand*{\si@tempb}{slash}%
   \ifx\si@tempa\si@tempb
     \si@fractrue
     \si@slashtrue
     \let\si@frac\si@frc@slash
   \fi
   \renewcommand*{\si@tempb}{frac}%
   \ifx\si@tempa\si@tempb
     \si@fractrue
   \fi
```

```
    \renewcommand*{\si@tempb}{fraction}%
    \ifx\si@tempa\si@tempb
      \si@fractrue
    \fi}
```

`\si@slash` For the `slash` option, the separator can be customised.

```
\si@opt@xchoicekey{slash}{slash}{slash}
```

`\ifsi@repeatunits` An option is needed for cases where units should be repeated.

`\ifsi@addunitpower`

```
\newif\ifsi@repeatunits
\newif\ifsi@addunitpower
\si@opt@choicekey[true]{repeatunits}{true,false,power}
  {\si@repeatunitsfalse
   \si@addunitpowerfalse
   \renewcommand*{\si@tempb}{true}%
   \ifx\si@tempa\si@tempb
     \si@repeatunitstrue
   \fi
   \renewcommand*{\si@tempb}{power}%
   \ifx\si@tempa\si@tempb
     \si@addunitpowertrue
   \fi}
```

`\ifsi@trapambigfrac` `\si@closefrac` `\si@openfrac` Macros for the right and left brackets added to potentially ambiguous denominators.

```
\si@opt@boolkey{trapambigfrac}
\si@opt@cmdkeys{closefrac,openfrac}
```

In the case of fractional handling of the `\per` operator, further refinement is available.

```
\si@opt@choicekey[frac]{fraction}
  {frac,nicefrac,nice,sfrac,xfrac,uglyfrac,ugly}
  {\let\si@frac\si@frc@frac
   \renewcommand*{\si@tempb}{nicefrac}%
   \ifx\si@tempa\si@tempb
     \let\si@frac\si@frc@nice
   \fi
   \renewcommand*{\si@tempb}{uglyfrac}%
   \ifx\si@tempa\si@tempb
     \let\si@frac\si@frc@ugly
   \fi
   \renewcommand*{\si@tempb}{nice}%
   \ifx\si@tempa\si@tempb
     \let\si@frac\si@frc@nice
   \fi
   \renewcommand*{\si@tempb}{sfrac}%
   \ifx\si@tempa\si@tempb
     \let\si@frac\si@frc@sfrac
   \fi
   \renewcommand*{\si@tempb}{xfrac}%
   \ifx\si@tempa\si@tempb
     \let\si@frac\si@frc@sfrac
   \fi
```

```
    \renewcommand*{\si@tempb}{ugly}%
    \ifx\si@tempa\si@tempb
      \let\si@frac\si@frc@ugly
    \fi}
```

\si@load
\si@noload

Loading of support files is controlled by two keys. The first defines a list of files that may be loaded, the second a list that will not. This makes it easy to exclude a single file from a long list.

```
\si@opt@cmdkeys{load,noload}
\si@opt@key{alsoload}{\si@addtolist{\si@load}{#1}}
\AtEndOfPackage{
  \si@opt@disablekey{load}
    {Configuration files can only be used\MessageBreak
     when loading package}
  \si@opt@disablekey{noload}
    {Configuration files can only be used\MessageBreak
     when loading package}}
\AtEndOfPackage{
  \si@opt@key{alsoload}{%
    \@for\si@tempa:=#1\do{\si@loadfile{\si@tempa}}}}
```

\ifsi@colourunits
\ifsi@colourvalues
\si@unitcolour
\si@valuecolour

Colour is turned on and off using two switches and the appropriate options. US spellings are also provided.

```
\si@opt@boolkey{colourunits}
\si@opt@boolkey{colourvalues}
\si@opt@choicekey[true]{colorunits}
  {true,false}
  {\si@colourunitsfalse
   \renewcommand*{\si@tempb}{true}%
   \ifx\si@tempa\si@tempb
     \si@colourunitstrue
   \fi}
\si@opt@choicekey[true]{colorvalues}
  {true,false}
  {\si@colourvaluesfalse
   \renewcommand*{\si@tempb}{true}%
   \ifx\si@tempa\si@tempb
     \si@colourvaluestrue
   \fi}
\si@opt@choicekey[true]{colorall}
  {true,false}
  {\si@colourvaluesfalse
   \si@colourunitsfalse
   \renewcommand*{\si@tempb}{true}%
   \ifx\si@tempa\si@tempb
     \si@colourunitstrue
     \si@colourvaluestrue
   \fi}
\si@opt@choicekey[true]{colourall}
  {true,false}
  {\si@colourvaluesfalse
   \si@colourunitsfalse
   \renewcommand*{\si@tempb}{true}%
   \ifx\si@tempa\si@tempb
```

```
      \si@colourunitstrue
      \si@colourvaluestrue
    \fi}
\si@opt@cmdkeys{unitcolour,valuecolour}
\si@opt@key{unitcolor}{\sisetup{unitcolour=#1}}
\si@opt@key{valuecolor}{\sisetup{valuecolour=#1}}
\si@opt@key{colour}{\sisetup{unitcolour=#1,valuecolour=#1}}
\si@opt@key{color}{\sisetup{unitcolour=#1,valuecolour=#1}}
```

\ifsi@colourneg \si@negcolour

The set up for colouring negative numbers is similar.

```
\si@opt@boolkey{colourneg}
\si@opt@choicekey[true]{colorneg}
  {true,false}
  {\si@colournegfalse
   \renewcommand*{\si@tempb}{true}%
   \ifx\si@tempa\si@tempb
     \si@colournegtrue
   \fi}
\si@opt@cmdkeys{negcolour}
\si@opt@key{negcolor}{\sisetup{negcolour=#1}}
```

\si@textOmega \si@mathsOmega

The various non-Latin symbols need to be handled, and given user interfaces. Some definitions are more complex than others; for Ω things are easy.

```
\si@opt@cmdkeys{textOmega,mathsOmega}
\si@opt@key{mathOmega}{\sisetup{mathsOmega=#1}}
\newcommand*{\si@mathsOmega}{\text{\ensuremath{\Omega}}}
\newcommand*{\si@textOmega}{\ensuremath{\Omega}}
```

\si@textmu \si@mathsmu

For the μ symbol, some direct loading of symbols is needed as the maths mu sign (μ) is wrong.

```
\si@opt@cmdkeys{textmu,mathsmu}
\si@opt@key{mathmu}{\sisetup{mathsmu=#1}}
\DeclareFontEncoding{TS1}{}{}
\DeclareFontSubstitution{TS1}{cmr}{m}{n}
\DeclareTextSymbol{\si@textmu}{TS1}{181}
\DeclareTextSymbolDefault{\si@textmu}{TS1}
\@ifpackageloaded{upgreek}
  {}
  {\DeclareFontFamily{OML}{eur}{\skewchar\font'177}
   \DeclareFontShape{OML}{eur}{m}{n}{%
    <-6> eurm5 <6-8> eurm7 <8-> eurm10}{}}
\AtBeginDocument{
  \@ifpackageloaded{upgreek}
    {\let\si@mathsmu\upmu}
    {\DeclareSymbolFont{si@greek}{OML}{eur}{m}{n}
     \DeclareMathSymbol{\si@mathsmu}{\mathord}{si@greek}{"16}}}
```

\si@textdegree \si@mathsdegree \si@textminute \si@mathsminute \si@textsecond \si@mathssecond

The angle signs.

```
\si@opt@cmdkeys{textdegree,mathsdegree,textminute,mathsminute,
  textsecond,mathssecond}
\si@opt@key{mathdegree}{\sisetup{mathsdegree=#1}}
\si@opt@key{mathminute}{\sisetup{mathsminute=#1}}
\si@opt@key{mathsecond}{\sisetup{mathssecond=#1}}
```

```
\newcommand*{\si@textdegree}{\ensuremath{{}^{\circ}}}
\newcommand*{\si@mathsdegree}{{}^{\circ}}
\newcommand*{\si@textminute}{\ensuremath{{}^{\prime}}}
\newcommand*{\si@mathsminute}{{}^{\prime}}
\newcommand*{\si@textsecond}{\ensuremath{{}^{\prime\prime}}}
\newcommand*{\si@mathssecond}{{}^{\prime\prime}}
```

`\si@textcelsius` `\si@mathscelsius` Finally, degrees Celsius, which may need the degree symbol.

```
\si@opt@cmdkeys{textcelsius,mathscelsius}
\si@opt@key{mathcelsius}{\sisetup{mathscelsius=#1}}
\newcommand*{\si@textcelsius}{%
  \si@textdegree\kern-\scriptspace C}
\newcommand*{\si@mathscelsius}{%
  \si@mathsdegree\kern-\scriptspace\mathrm{C}}
```

`\si@textringA` `\si@mathsringA` The Å sign.

```
\si@opt@cmdkeys{textringA,mathsringA}
\si@opt@key{mathringA}{\sisetup{mathsringA=#1}}
\newcommand*{\si@textringA}{\AA}
\newcommand*{\si@mathsringA}{\text{\AA}}
```

`\ifsi@redefsymbols` A flag for using textcomp and upgreek to provide better symbols.

```
\si@opt@boolkey{redefsymbols}
\AtBeginDocument{
  \si@opt@disablekey{redefsymbols}
    {Symbols can only be redefined\MessageBreak
     when loading siunitx}}
```

`\si@eVcorra` `\si@eVcorrb`

```
\newlength\si@eVcorra
\newlength\si@eVcorrb
\si@opt@key{eVcorra}{\setlength\si@eVcorra{#1}}
\si@opt@key{eVcorrb}{\setlength\si@eVcorrb{#1}}
```

`\si@locale` `\si@loctolang` Handling typographic conventions needs three keys. `locale` is used to set the locale, `loctolang` to bind to babel.

```
\si@opt@key{locale}{%
  \si@loc@load{#1}%
  \si@loc@set{#1}}%
\si@opt@key{loctolang}{\si@loc@ltol{#1}}
```

21.5 Compatibility options

`\ifsi@old@ugly` `\ifsi@old@nice` `\ifsi@old@loose` `\ifsi@old@tight` With the options for the package set up, the next stage is to provide support for users of the older packages. These all set up switches, but do not do anything. That is left to the emulation files, loaded at the end of the package. First of all, the units options are dealt with; there are not many.

```
\si@opt@compatkey{units}{ugly}
\si@opt@compatkey{units}{nice}
\si@opt@compatkey{units}{loose}
\si@opt@compatkey{units}{tight}
```

\ifsi@old@OHM
\ifsi@old@ohm
\ifsi@old@redef-gensymb
\ifsi@gensymb

The unitsdef package is unfortunately much more profligate with options. The first set are to do with support for gensymb.

```
828 \si@opt@compatkey{unitsdef}{OHM}
829 \si@opt@compatkey{unitsdef}{ohm}
830 \si@opt@compatkey{unitsdef}{redef-gensymb}
831 \newif\ifsi@gensymb
```

\ifsi@old@LITER
\ifsi@old@liter
\ifsi@old@noxspace
\ifsi@old@noconfig

The second set are more general functionality.

```
832 \si@opt@compatkey{unitsdef}{LITER}
833 \si@opt@compatkey{unitsdef}{liter}
834 \si@opt@compatkey{unitsdef}{noxspace}
835 \si@opt@compatkey{unitsdef}{noconfig}
```

\ifsi@old@noabbr
\ifsi@old@noamperageabbr
\ifsi@old@nofrequncyabbr
\ifsi@old@nomolabbr
\ifsi@old@novoltageabbr
\ifsi@old@novolumeabbr
\ifsi@old@noweightabbr
\ifsi@old@noenergyabbr
\ifsi@old@nolengthabbr
\ifsi@old@notimeabbr

The final set are for control of abbreviations, and are a good demonstration of why to use xkeyval!

```
836 \si@opt@compatkey{unitsdef}{noabbr}
837 \si@opt@compatkey{unitsdef}{noampereageabbr}
838 \si@opt@compatkey{unitsdef}{nofrequncyabbr}
839 \si@opt@compatkey{unitsdef}{nomolabbr}
840 \si@opt@compatkey{unitsdef}{novoltageabbr}
841 \si@opt@compatkey{unitsdef}{novolumeabbr}
842 \si@opt@compatkey{unitsdef}{noweightabbr}
843 \si@opt@compatkey{unitsdef}{noenergyabbr}
844 \si@opt@compatkey{unitsdef}{nolengthabbr}
845 \si@opt@compatkey{unitsdef}{notimeabbr}
```

\ifsi@old@cdot
\ifsi@old@thickspace
\ifsi@old@mediumspace
\ifsi@old@thinspace
\ifsi@old@thickqspace
\ifsi@old@mediumqspace
\ifsi@old@thinqspace

The SIunits package has lots of options. These ones are all related to spacing.

```
846 \si@opt@compatkey{SIunits}{cdot}
847 \si@opt@compatkey{SIunits}{thickspace}
848 \si@opt@compatkey{SIunits}{mediumspace}
849 \si@opt@compatkey{SIunits}{thinspace}
850 \si@opt@compatkey{SIunits}{thickqspace}
851 \si@opt@compatkey{SIunits}{mediumqspace}
852 \si@opt@compatkey{SIunits}{thinqspace}
```

\ifsi@old@amssymb
\ifsi@old@squaren
\ifsi@old@pstricks
\ifsi@old@Gray
\ifsi@old@italian

These options are used by SIunits to control clashes with other packages.

```
853 \si@opt@compatkey{SIunits}{amssymb}
854 \si@opt@compatkey{SIunits}{squaren}
855 \si@opt@compatkey{SIunits}{pstricks}
856 \si@opt@compatkey{SIunits}{Gray}
857 \si@opt@compatkey{SIunits}{italian}
```

\ifsi@old@textstyle
\ifsi@old@binary
\ifsi@old@noams
\ifsi@old@derivedinbase
\ifsi@old@derived

The miscellaneous options.

```
858 \si@opt@compatkey{SIunits}{textstyle}
859 \si@opt@compatkey{SIunits}{binary}
860 \si@opt@compatkey{SIunits}{noams}
861 \si@opt@compatkey{SIunits}{derivedinbase}
862 \si@opt@compatkey{SIunits}{derived}
```

\ifsi@old@noprefixcmds

The hepunits package only has one option.

```
863 \si@opt@compatkey{hepunits}{noprefixcmds}
```

\ifsi@old@english
\ifsi@old@french

fancynum provides a few options. First the rather oddly named `english` and `french` ones.

```
\si@opt@compatkey{fancynum}{english}
\si@opt@compatkey{fancynum}{french}
```

\ifsi@old@tight
\ifsi@old@loose

A couple for spacing multiplication.

```
\si@opt@compatkey{fancynum}{tight}
\si@opt@compatkey{fancynum}{loose}
```

\ifsi@old@thinspaces
\ifsi@old@commas
\ifsi@old@plain

Three for digit separation.

```
\si@opt@compatkey{fancynum}{thinspaces}
\si@opt@compatkey{fancynum}{commas}
\si@opt@compatkey{fancynum}{plain}
```

\ifsi@old@spaceqspace

The fancyunits package provides one option not available with SIunits.

```
\si@opt@compatkey{fancyunits}{spaceqspace}
```

21.6 Constants

A number of macros are needed by the package that provide a non-changing output. These are defined here; the intention is that these should not be macros that the user is likely to need to alter. All of these macros have preface `\si@fix@`, to flag that that are intended as constants. The package may rely on the contents of these macros for functionality.

\si@fix@thin
\si@fix@med
\si@fix@medium
\si@fix@thick
\si@fix@space

First, there are the various space macros. To allow both `med` and `medium` to be used as a space description, two macros are needed for the same output.

```
\newcommand*{\si@fix@thin}{\,}
\newcommand*{\si@fix@med}{\:}
\newcommand*{\si@fix@medium}{\:}
\newcommand*{\si@fix@thick}{\;}
\newcommand*{\si@fix@space}{\text{ }}
```

\si@fix@cdot
\si@fix@comma
\si@fix@stop
\si@fix@fullstop
\si@fix@period
\si@fix@times
\si@fix@tighttimes
\si@fix@tightcdot

Next there are macros for material that is not simply whitespace. To allow several options, the full-stop gets lots of names.

```
\newcommand*{\si@fix@cdot}{{}\cdot{}}
\newcommand*{\si@fix@comma}{{,}}
\newcommand*{\si@fix@stop}{{.}}
\newcommand*{\si@fix@fullstop}{{.}}
\newcommand*{\si@fix@period}{{.}}
\newcommand*{\si@fix@times}{\times}
\newcommand*{\si@fix@tighttimes}{\bgroup\times\egroup}
\newcommand*{\si@fix@tightcdot}{\bgroup\cdot\egroup}
```

\si@fix@plus
\si@fix@minus
\si@fix@pm
\si@fix@tightpm
\si@fix@mp

Signs for numbers are needed.

```
\newcommand*{\si@fix@plus}{+}
\newcommand*{\si@fix@minus}{-}
\newcommand*{\si@fix@pm}{\pm}
\newcommand*{\si@fix@tightpm}{\bgroup\pm\egroup}
\newcommand*{\si@fix@mp}{\mp}
```

\si@fix@two The literals "2" and "10" are needed for exponents.

\si@fix@ten
```
\newcommand*{\si@fix@two}{2}
\newcommand*{\si@fix@ten}{10}
```

\si@fix@slash Another optional component that will probably not be used by many people.

```
\newcommand*{\si@fix@slash}{/}
```

\si@fix@none Finally for spacing, there is the possibility of nothing at all

```
\newcommand*{\si@fix@none}{}
```

21.7 Symbols

\si@symbol Each of the symbol macros needs to be set up; the options give a maths and text mode sign, but internally a single macro is needed for each.

```
\newcommand*{\si@symbol}[1]{%
  \expandafter\protected\expandafter\def
    \csname si@sym@#1\endcsname{%
    \ifmmode
      \expandafter\csname si@maths#1\expandafter\endcsname
    \else
      \expandafter\csname si@text#1\expandafter\endcsname
    \fi}}
```

\si@sym@Omega The various symbols are now declared.

\si@sym@ringA
\si@sym@mu
\si@sym@degree
\si@sym@minute
\si@sym@second
\si@sym@celsius

```
\si@symbol{Omega}
\si@symbol{ringA}
\si@symbol{mu}
\si@symbol{degree}
\si@symbol{minute}
\si@symbol{second}
\si@symbol{celsius}
```

The issue of redefinition of symbols now arises. siunitx can check for the loading of a number of support package, and can then redefine the appropriate symbols.

```
\AtBeginDocument{%
  \ifsi@redefsymbols
    \@ifpackageloaded{textcomp}
      {\si@log@debug{Redefining symbols using textcomp}%
       \renewcommand*{\si@textdegree}{\textdegree}%
       \renewcommand*{\si@mathsdegree}{\text{\textdegree}}%
```

mathptmx will give issues with textcomp and the Ω sign.

```
       \@ifpackageloaded{mathptmx}{}
         {\renewcommand*{\si@textmu}{\textmu}%
          \renewcommand*{\si@textOmega}{\textohm}}%
```

The Å symbol is only redefined if the encoding is OT1; other encodings should have a proper glyph used for \AA. The \encodingdefault macro is \long for some reason.

```
       \renewcommand*{\si@tempa}{OT1}%
       \ifx\si@tempa\encodingdefault
         \renewcommand*{\si@mathsringA}{%
           \text{\capitalring{A}}}%
```

```
          \renewcommand*{\si@textringA}{\capitalring{A}}
        \fi}{}
    \@ifpackageloaded{upgreek}
      {\si@log@debug{Redefining symbols using upgreek}%
       \renewcommand*{\si@mathsOmega}{\Upomega}}{}
  \fi}
```

21.8 Handling fractions

\si@frac
\si@frc@hook
\si@frc@frac
\si@frc@slash
\si@frc@nice
\si@frc@sfrac

Various methods of handling fractions are provided.

```
\newcommand*{\si@frc@frac}[2]{%
  \ensuremath{\si@frc@hook\frac{%
    \expandafter\si@unt@out\expandafter{#1}}%
    {\expandafter\si@unt@out\expandafter{#2}}}}
\let\si@frac\si@frc@frac
\newcommand*{\si@frc@hook}{}
\newcommand*{\si@frc@slash}[2]{%
    \expandafter\si@unt@out\expandafter{#1}%
    \si@out{\ensuremath{\si@slash}}%
    \expandafter\si@unt@out\expandafter{#2}}
\newcommand*{\si@frc@nice}[2]{%
  \ensuremath{\si@frc@nicefrac{\expandafter\si@unt@out%
    \expandafter{#1}}{\expandafter\si@unt@out\expandafter
      {#2}}}}
\newcommand*{\si@frc@sfrac}[2]{%
    \sfrac{\expandafter\si@unt@out\expandafter{#1}}%
      {\expandafter\si@unt@out\expandafter{#2}}}
\AtBeginDocument{
  \@ifpackageloaded{xfrac}
    {}
    {\si@log@inf{xfrac package unavailable\MessageBreak
       using 'fraction=sfrac' will fall back on\MessageBreak
       nicefrac-like method}%
     \renewcommand*{\si@frc@sfrac}[2]{%
      \si@log@warn{xfrac package unavailable}%
      \si@frc@nice{#1}{#2}}}}
```

\si@frc@nicefrac
\si@frc@displen
\si@frc@textlen
\si@frc@suplen
\si@frc@ssuplen

To avoid needing units installed, the `\nicefrac` macro needs to be emulated here. The code is taken (with permission) from kgnicefrac.[52]

```
\newlength\si@frc@displen
\newlength\si@frc@textlen
\newlength\si@frc@suplen
\newlength\si@frc@ssuplen
\newcommand*{\si@frc@nicefrac}{%
  \ifmmode
    \expandafter\si@frc@mathsnf
  \else
    \expandafter\si@frc@textnf
  \fi}
```

\si@frc@mathsnf

`#1` : numerator

[52] The original is licensed under the GPL; thanks to the author Axel Reichert for permission to copy the code here.

`#2` : denominator

The maths mode system.

```
\newcommand*{\si@frc@mathsnf}[2]{%
  \begingroup
    \settoheight{\si@frc@displen}{\ensuremath{%
      \displaystyle{M}}}%
    \settoheight{\si@frc@textlen}{\ensuremath{%
      \textstyle{M}}}%
    \settoheight{\si@frc@suplen}{\ensuremath{%
      \scriptstyle{M}}}%
    \settoheight{\si@frc@ssuplen}{%
      \ensuremath{\scriptscriptstyle{M}}}%
    \addtolength{\si@frc@displen}{-\si@frc@ssuplen}%
    \addtolength{\si@frc@textlen}{-\si@frc@ssuplen}%
    \addtolength{\si@frc@suplen}{-\si@frc@ssuplen}%
    \mathchoice
      {\raisebox{\si@frc@displen}{\ensuremath{%
        \scriptstyle{#1}}}}%
      {\raisebox{\si@frc@textlen}{\ensuremath{%
         \scriptstyle{#1}}}}%
      {\raisebox{\si@frc@suplen}%
        {\ensuremath{\scriptscriptstyle{#1}}}}%
      {\raisebox{\si@frc@ssuplen}%
        {\ensuremath{\scriptscriptstyle{#1}}}}%
    \mkern-2mu\relax/\mkern-1mu\relax
    \bgroup
      \mathchoice
        {\scriptstyle}%
        {\scriptstyle}%
        {\scriptscriptstyle}%
        {\scriptscriptstyle}%
      {#2}%
    \egroup
  \endgroup}
```

`\si@frc@textnf` `#1` : numerator

`#2` : denominator

A stripped down version of the nicefrac system for text mode.

```
\newcommand*{\si@frc@textnf}[2]{%
  \begingroup
    \settoheight{\si@frc@textlen}{M}%
    \settoheight{\si@frc@ssuplen}{\fontsize\sf@size\z@\relax
      \selectfont{M}}%
    \addtolength{\si@frc@textlen}{-\si@frc@ssuplen}%
    \raisebox{\si@frc@textlen}{\fontsize\sf@size\z@\relax
      \selectfont{#1}}%
    \hspace{-0.25ex}/\hspace{-0.25ex}%
    \hbox{\fontsize\sf@size\z@\selectfont{#2}}%
  \endgroup}
```

`\si@frc@ugly` `#1` : numerator

The `\si@frc@ugly` macro is needed to emulate the `ugly` option in units, where output depends on the current mode.

```
\newcommand*{\si@frc@ugly}[1]{%
  \renewcommand*{\si@tempa}{#1}%
  \ifmmode
    \expandafter\si@frc@frac
  \else
    \renewcommand*{\si@tempb}{1}%
    \ifx\si@tempa\si@tempb
```

The slash switch cannot be used, so the possibility of the numerator being one is handled here.

```
      \setbox\si@tempboxa=\hbox{\ensuremath{\si@valuesep}}%
      \hskip-\wd\si@tempboxa\relax
      \renewcommand*{\si@tempa}{}%
    \fi
    \expandafter\si@frc@slash
  \fi
  {\si@tempa}}
```

21.9 Font control

A number of controls and tests are needed to control the font used for output. Underlying all of this is the $\mathcal{AMS}$ package amstext package, providing the `\text` command. Much of the font control system here is taken more or less verbatim from SIstyle; modifications have been made to fit the siunitx interface.

`\si@fam@sf` `\si@fam@tt` The package needs to know the maths font families in use. This is set right at the start of the document, after any changes can have been made by, for example, fontspec.

```
\g@addto@macro{\document}{%
  \ifdefined\mathsf
    \setbox\si@tempboxa=\hbox{%
      $\mathsf{\global\chardef\si@fam@sf=\fam}$}%
  \else
    \si@log@inf{\string\mathsf not found}%
    \global\chardef\si@fam@sf=99\relax
  \fi
  \ifdefined\mathtt
    \setbox\si@tempboxa=\hbox{%
      $\mathtt{\global\chardef\si@fam@tt=\fam}$}%
  \else
    \si@log@inf{\string\mathtt not found}%
    \global\chardef\si@fam@tt=99\relax
  \fi}
```

`\si@fam@ifbtext` `\si@fam@ifbmaths` `#1 : code` These tests check for bold in text and maths mode, respectively.

```
\newcommand*{\si@fam@ifbtext}[1]{%
  \if b\expandafter\@car\f@series\@nil
    #1\fi}
\newcommand*{\si@fam@ifbmaths}[1]{%
  \renewcommand*{\si@tempa}{bold}%
  \ifx\math@version\si@tempa
    #1\fi}
```

\si@fam@ifbinline For compatibility with units, a method to change the behaviour when in inline maths is needed for the bold detector.

```
\newcommand*{\si@fam@ifbinline}{%
  \ifsi@inlinebtext
    \expandafter\si@fam@ifbtext
  \else
    \expandafter\si@fam@ifbmaths
  \fi}
```

\si@fam@ifitext `#1 : code`

This test check for italic or slanted text in text mode, by negation (upright text is `n`).

```
\newcommand*{\si@fam@ifitext}[1]{%
  \if n\expandafter\@car\f@series\@nil\else
    #1\fi}
```

\si@fam@mode Detection of the current mode needs to happen"early" (before any change of `\ensuremath`). So a short macro is provided to do the job.

```
\newcommand*{\si@fam@mode}{%
  \ifsi@obeymode
    \ifmmode
      \sisetup{mode=maths}%
    \else
      \sisetup{mode=text}%
    \fi
  \fi}
```

\si@fam@colourcmd The colour command is set up.

```
\AtBeginDocument{
  \@ifpackageloaded{color}
    {\let\si@fam@colourcmd\color}
    {\let\si@fam@colourcmd\@gobble}}
```

\ifsi@fam@set A marker is set up to check if font-matching has been taken place. A second flag \ifsi@textmode is used to track the use of text mode.

```
\newif\ifsi@fam@set
\newif\ifsi@textmode
```

\si@fam@set Using the code from SIstyle as a base, a set of tests are used to set the current font families and weights. To begin with, the mode to use is set up.

```
\newcommand*{\si@fam@set}{%
  \ifsi@out@num
    \ifsi@numtextmode
      \expandafter\expandafter\expandafter\si@textmodetrue
    \else
      \expandafter\expandafter\expandafter\si@textmodefalse
    \fi
  \else
    \ifsi@unittextmode
      \expandafter\expandafter\expandafter\si@textmodetrue
    \else
      \expandafter\expandafter\expandafter\si@textmodefalse
    \fi
  \fi
```

\si@mathsrm
\si@mathssf
\si@mathstt
\si@textrm
\si@textsf
\si@texttt
\si@colourcmd
\si@colour

The appropriate font macros are now established, if necessary.

```
\ifsi@fam@set\else
  \let\si@colourcmd\@gobble
  \ifsi@out@num
    \let\si@mathsrm\si@valuemathsrm
    \let\si@mathssf\si@valuemathssf
    \let\si@mathstt\si@valuemathstt
    \let\si@textrm\si@valuetextrm
    \let\si@textsf\si@valuetextsf
    \let\si@texttt\si@valuetexttt
    \ifsi@colourvalues
      \let\si@colourcmd\si@fam@colourcmd
    \fi
    \let\si@colour\si@valuecolour
  \else
    \let\si@mathsrm\si@unitmathsrm
    \let\si@mathssf\si@unitmathssf
    \let\si@mathstt\si@unitmathstt
    \let\si@textrm\si@unittextrm
    \let\si@textsf\si@unittextsf
    \let\si@texttt\si@unittexttt
    \ifsi@colourunits
      \let\si@colourcmd\si@fam@colourcmd
    \fi
    \let\si@colour\si@unitcolour
  \fi
\fi
\si@fam@settrue
```

The temporary macros are needed for the `\ifx` tests, which need to be expanded once.

```
\edef\si@tempa{\sfdefault}%
\edef\si@tempb{\ttdefault}%
```

\si@fam@maths
\si@fam@text

The surrounding font family is only tested if matching is requested. First, the defaults are set up assuming no detection takes place.

```
\expandafter\let\expandafter\si@fam@maths
  \csname\si@mathsrm\endcsname
\expandafter\let\expandafter\si@fam@text
  \csname\si@textrm\endcsname
\ifsi@obeyfamily
  \si@log@debug{Font detection: checking font}%
```

The detection code has to check the mode currently in operation. Display mathematics can be handled in two ways, so this means some code is repeated: it is spun out to separate routines.

```
  \ifmmode
    \ifinner
      \si@log@debug{Font detection: inline maths}%
      \si@fam@dettext
    \else
      \si@log@debug{Font detection: display maths}%
      \ifsi@detectdisplay
```

```
          \si@fam@detmaths
        \else
          \si@fam@dettext
        \fi
      \fi
    \else
      \si@log@debug{Font detection: text}%
      \si@fam@dettext
    \fi
  \else
    \si@log@debug{Font detection: inactive}%
  \fi
```

`\si@fam@bold` With the font family set, the next check is for bold text. This again needs to examine the current mode. Things are a bit more complex than in SIstyle as it is possible to be typesetting in either text or maths mode. The bold command is set up with `\def`, as nested calls can occur.

```
  \def\si@fam@bold{\unboldmath\mdseries}%
  \ifsi@obeybold
    \si@log@debug{Weight detection: checking weight}%
    \ifmmode
      \ifdim\displaywidth>0pt\relax
        \ifsi@detectdisplay
          \expandafter\si@fam@ifbmaths
        \else
          \expandafter\si@fam@ifbtext
        \fi
        \si@fam@setbold
      \else
        \si@fam@ifbinline\si@fam@setbold
      \fi
    \else
      \si@fam@ifbtext\si@fam@setbold
    \fi
  \fi
```

`\si@fam@italic` The value of `obeyitalic` is now tested; as this does nothing in maths mode, a reminder is added to the log.

```
  \let\si@fam@italic\upshape
  \ifsi@obeyitalic
    \si@log@debug{Italic detection: checking italic}%
    \si@fam@ifitext
      {\let\si@fam@italic\relax
       \si@log@debug{Italic detection: italic}}%
  \fi}
```

`\si@fam@detmaths` `\si@fam@dettext` Two detection macros are needed for maths and text mode. This allows handling of the various combinations without needing too many code lines.

```
\newcommand*{\si@fam@detmaths}{%
  \ifnum\the\fam=\si@fam@sf
    \si@log@debug{Font detection: sf}%
    \expandafter\let\expandafter\si@fam@maths
      \csname\si@mathssf\endcsname
```

```
    \expandafter\let\expandafter\si@fam@text
      \csname\si@textsf\endcsname
  \else
    \ifnum\the\fam=\si@fam@tt
      \si@log@debug{Font detection: tt}%
      \expandafter\let\expandafter\si@fam@maths
        \csname\si@mathstt\endcsname
      \expandafter\let\expandafter\si@fam@text
        \csname\si@texttt\endcsname
    \else
      \si@log@debug{Font detection: rm}%
      \expandafter\let\expandafter\si@fam@maths
        \csname\si@mathsrm\endcsname
      \expandafter\let\expandafter\si@fam@text
        \csname\si@textrm\endcsname
    \fi
  \fi}
\newcommand*{\si@fam@dettext}{%
  \ifx\f@family\si@tempa
    \si@log@debug{Font detection: sf}%
    \expandafter\let\expandafter\si@fam@maths
      \csname\si@mathssf\endcsname
    \expandafter\let\expandafter\si@fam@text
      \csname\si@textsf\endcsname
  \else
    \ifx\f@family\si@tempb
      \si@log@debug{Font detection: tt}%
      \expandafter\let\expandafter\si@fam@maths
        \csname\si@mathstt\endcsname
      \expandafter\let\expandafter\si@fam@text
        \csname\si@texttt\endcsname
    \else
      \si@log@debug{Font detection: rm}%
      \expandafter\let\expandafter\si@fam@maths
        \csname\si@mathsrm\endcsname
      \expandafter\let\expandafter\si@fam@text
        \csname\si@textrm\endcsname
    \fi
  \fi}
```

`\si@fam@setbold` For setting bold, a couple of control macros are needed.

`\si@fam@boldify`

```
\newcommand*{\si@fam@setbold}{%
  \si@log@debug{Weight detection: bold weight}%
  \let\si@fam@bold\si@fam@boldify}
\newcommand*{\si@fam@boldify}{\boldmath\bfseries}
```

21.10 Formatting numbers

`\num [#1]` : keyval options
`#2` : number

The system used here is modelled on that in numprint; the input is broken down into single tokens, each one is examined and the result is re-assembled into an output number. However, various changes have been made to the system used,

and so the macros here are not simply renamed copies of those in numprint. The user macro \num sets any local keys, then calls the number formatting macro on the processed number.

```
\si@newrobustcmd*{\num}[2][]{%
  \begingroup
    \sisetup{#1}%
    \si@fam@mode
    \si@num@intabfalse
    \si@log@debug{Processing \string\num\space input `#2'}%
    \expandafter\si@out@num\expandafter{\si@num{#2}}%
  \endgroup}
```

\ifsi@num@intab A flag for processing inside a table is needed.

```
\newif\ifsi@num@intab
```

\si@num #1 : number

This is the main processing macro. Unlike the related macro in numprint, the output of this macro is not subjected to any font changes. That is left to one of the \si@out@... macros. No grouping is applied here; any call to \si@num (or any of the sub-macros) must be within a group as the definitions used rely on this. Grouping is not applied here so that other macros can get the various separated parts of the input.

```
\newcommand*{\si@num}[1]{%
```

The argument of the macro is fully expanded before any processing. By using \scantokens, any odd problems from packages with active characters can be avoided.

```
  \si@num@fixpm
  \begingroup
    \makeatletter
    \@makeother{\,}%
    \@makeother{\.}%
    \@makeother{\+}%
    \@makeother{\-}%
    \def~{}%
    \def\,{}%
    \catcode`\~=\active\relax
    \catcode`\^=\active\relax
    \everyeof{\noexpand}%
    \endlinechar\m@ne
    \protected@xdef\si@tempa{\scantokens{#1}}%
  \endgroup
```

Processing only takes place if there is actually something in the argument. This is tested once "hard" spaces have been stripped out; if there is input other than spaces, the processor first checks the validity of the input, then moves on to format it.

```
  \si@ifnotmtarg{\si@tempa}
    {\si@num@ifvalid{\si@tempa}
      {\si@num@format{\si@tempa}}
```

The parser has to bailed-out, and so no further processing of the input is done. Instead, whatever was passed to the macro is returned as supplied.

```
      {\si@log@err{Invalid character `#1' in numerical input}%
         {Only characters from the list
          `\si@numvalid'\MessageBreak should be present in the
          argument of the \string\num\space  macro\MessageBreak
         (or derivative such as an `s' column)}%
       {#1}}}}
```

`\si@num@fixpm` `\si@num@pm` `\si@num@mp` With certain packages loaded, there can be issues with `\pm` and `\mp`. To avoid this, the ε-TEX `\protected` system is employed; this is only used within local groups.

`\pm` `\mp`

```
\newcommand*{\si@num@fixpm}{%
  \let\si@num@pm\pm
  \let\si@num@mp\mp
  \protected\def\pm{\si@num@pm}%
  \protected\def\mp{\si@num@mp}}
```

`\si@num@ifvalid` `#1` : chars

`\si@num@valid` Assuming that there is a non-space argument to `\si@num`, every character is checked to ensure it is valid in the context, so that further processing can occur without sanity checks. If the character is valid, recursion occurs.

```
\newcommand*{\si@num@ifvalid}[1]{%
  \begingroup
    \si@switchtrue
    \expandafter\si@num@valid#1\@empty\@empty
    \ifsi@switch
      \aftergroup\@firstoftwo
    \else
      \aftergroup\@secondoftwo
    \fi
  \endgroup}
\def\si@num@valid#1#2\@empty{%
  \si@str@ifchrstr{#1}{\si@numvalid}
    {\ifx\@empty#2\@empty\else
       \si@num@valid#2\@empty\@empty\@empty
     \fi}
    {\si@switchfalse}}
```

`\si@num@in` Various storage macros are needed.

`\si@num@out` `\si@num@exp` `\si@num@expsign` `\si@num@mant` `\si@num@mantsign` `\si@num@err` `\si@num@xpart` `\si@num@ambig` `\si@tab@out` `\si@tab@expout`

```
\newcommand*{\si@num@in}{}
\newcommand*{\si@num@out}{}
\newcommand*{\si@num@exp}{}
\newcommand*{\si@num@expsign}{}
\newcommand*{\si@num@mant}{}
\newcommand*{\si@num@mantsign}{}
\newcommand*{\si@num@err}{}
\newcommand*{\si@num@xpart}{}
\newcommand*{\si@num@ambig}{}
\newcommand*{\si@tab@out}{}
\newcommand*{\si@tab@expout}{}
```

`\ifsi@num@erropen` A flag is set up for tracking unclosed errors.

```
\newif\ifsi@num@erropen
```

`\si@num@format` `#1` : number
`\si@num@arg` The number processor starts by saving `#1` (odd things happen otherwise). A hook is also provided to allow modifications by other macros.

```
\newcommand*{\si@num@arg}{}
\newcommand*{\si@num@format}[1]{%
  \protected@edef\si@num@arg{#1}%
  \si@log@debug{Formatting number `\si@num@arg'}%
```

The storage areas are emptied.

```
  \renewcommand*{\si@num@in}{}%
  \renewcommand*{\si@num@exp}{}%
  \renewcommand*{\si@num@expsign}{}%
  \renewcommand*{\si@num@mant}{}%
  \renewcommand*{\si@num@mantsign}{}%
  \renewcommand*{\si@num@err}{}%
  \renewcommand*{\si@num@xpart}{}%
```

Any "x-part" is now found, leaving the first number in `\si@num@in` and anything else in `\si@num@xpart`.

```
  \si@switchfalse
  \expandafter\si@num@findxpart\si@num@arg\@empty\@empty
```

The input is split into an mantissa and an exponent; the flag is used here to indicate if an exponent is found. The mantissa will end up in `\si@num@mant`, and the exponent in `\si@num@exp`.

```
  \si@switchfalse
  \si@num@sepmantexp{\si@num@in}%
```

The sign and value of the mantissa and exponent are separated; the mantissa is done after the exponent as this makes life easier when using the table-formatting fork. Checks are then needed, as a sign with no value is potentially-valid for the mantissa (for example -10^{10}).

```
  \si@num@sepsign{exp}%
  \si@num@sepsign{mant}%
  \ifx\@empty\si@num@exp\@empty
    \ifx\@empty\si@num@expsign\@empty\else
      \si@log@warn{Sign but no number for `\si@num@arg'}%
    \fi
    \let\si@num@expsign\@empty
  \fi
  \ifx\@empty\si@num@mant\@empty
    \ifx\@empty\si@num@mantsign\@empty\else
      \ifx\@empty\si@num@exp\@empty
        \si@log@warn{Sign but no number for `\si@num@arg'}%
        \let\si@num@mantsign\@empty
      \fi
    \fi
  \fi
```

A check for negative mantissa values is made, to allow some colour-based trickery.

```
  \renewcommand*{\si@tempa}{-}%
  \ifx\si@num@mantsign\si@tempa
    \ifsi@colourneg
      \expandafter\expandafter\expandafter\si@fam@colourcmd
    \else
```

```
      \expandafter\expandafter\expandafter\@gobble
    \fi
  \else
    \expandafter\@gobble
  \fi
  {\si@negcolour}%
```

The next stage is to process the remaining data, to find the decimal marker and reformat correctly. These two macros control this entire process. The exponent is processed first as this makes life easier when the system is used to typeset tabular material.[53]

```
  \si@num@procnum{exp}%
  \si@num@procnum{mant}%
```

If the exponent is zero, then if might need to be deleted.

```
  \si@str@ifonlychrs{\si@num@exp}{0\si@numdecimal}
    {\ifsi@allowzeroexp\else
       \renewcommand*{\si@num@exp}{}%
       \ifx\@empty\si@num@mant\@empty
         \renewcommand*{\si@num@mant}{1}%
       \fi
     \fi}{}%
```

To build up the number for typesetting, a rather complex series of tests is needed. First, the "ambiguous error" flag is set if there is an exponent and the package has been asked to check this. The same flag will already be true if a unit is present and checking is active.

```
  \ifx\@empty\si@num@exp\@empty\else
    \ifsi@trapambigerr
      \expandafter\expandafter\expandafter\si@num@ambigerrtrue
    \fi
  \fi
```

`\si@num@out` `\si@tab@out` `\si@tab@expout`

Processing now divides, as when used with the `S` column some extra steps are needed. Inside a table, the macro `\si@tab@out` holds the part of the mantissa before the decimal sign. The post-decimal part then ends up in `\i@num@out`. On the other hand, under normal circumstances the entire mantissa should be copied to `\si@num@out`.

```
  \protected@edef\si@num@out{%
    \ensuremath{{\si@num@mantsign}}\si@num@mant}%
  \renewcommand*{\si@tempa}{num}%
  \ifsi@num@intab
    \protected@edef\si@tab@out{%
      \ensuremath{{\si@num@mantsign}}\si@num@predec}%
    \protected@edef\si@num@out{\si@num@postdec}%
    \renewcommand*{\si@tempa}{tab}%
  \fi
```

Now there is the error part to handle. For tables, a check has to be made so the error ends up in the correct part of the number. The value of `\si@tempa` is used to track this., and so is set up here.

```
  \ifx\@empty\si@num@postdec\@empty\else
    \renewcommand*{\si@tempa}{num}%
```

[53] The contents of `\si@num@predec` and `\si@num@postdec` are needed for the mantissa.

```
  \fi
  \ifx\@empty\si@num@err\@empty\else
```

If there is an error, and it is begin separated, the problem arises of the potential for ambiguous values. This can only apply outside of a table, as `seperr` is disabled in tabular material.

```
    \ifsi@seperr
      \ifsi@num@ambigerr
        \protected@edef\si@num@out{%
          \ensuremath{\si@openerr}\si@num@out}%
        \si@repeatunitsfalse
        \expandafter\si@num@erropentrue
      \else
```

If there is an exponential part and an error, errors are being separated and ambiguous errors are not trapped, then there is work to do. The exponent part is added to the *error*, and deleted from the mantissa if necessary.

```
        \ifsi@trapambigerr\else
          \ifx\@empty\si@num@exp\@empty\else
            \protected@edef\si@num@err{%
              \si@num@err\expandafter\@car\si@numexp\@nil
              \si@num@expsign\si@num@exp}%
            \ifsi@repeatunits\else
              \renewcommand*{\si@num@exp}{}%
              \renewcommand*{\si@num@expsign}{}%
            \fi
          \fi
        \fi
      \fi
```

If `seperr` is not in force, the error and mantissa have to be recombined. This is handled so that the same macro deals with tables and normal processing.

```
    \else
      \expandafter\protected@edef\csname
        si@\si@tempa @out\endcsname{%
          \si@num@out\ensuremath{\si@errspace}\ensuremath
          {\si@openerr}\si@num@err\ensuremath{\si@closeerr}}%
      \renewcommand*{\si@num@err}{}%
    \fi
  \fi
```

The main reconstruction can now occur. This performs various checks on the validity of the input, and adds the necessary "filler" between the supplied data.

```
  \renewcommand*{\si@tempa}{num@out}%
  \ifsi@num@erropen
    \renewcommand*{\si@tempa}{num@ambig}%
  \fi
  \ifsi@num@intab
    \renewcommand*{\si@tempa}{tab@expout}%
  \fi
  \ifx\@empty\si@num@exp\@empty
    \ifx\@empty\si@num@mant\@empty
      \si@log@err{Invalid number format `\si@num@arg'}
        {Something is wrong with the number format; does it
         contain \MessageBreak any numbers (from the list
```

```
          `\si@numdigits')?}%
      \renewcommand*\si@num@out{}%
    \fi
  \else
    \ifx\@empty\si@num@mant\@empty\else
      \expandafter\protected@edef\csname
        si@\si@tempa\endcsname{%
          \csname si@\si@tempa\endcsname\ensuremath{{}%
          \si@expproduct{}}}%
    \fi
    \expandafter\protected@edef\csname
      si@\si@tempa\endcsname{%
        \csname si@\si@tempa\endcsname\si@expbase
        \textsuperscript{\ensuremath{\si@num@expsign}%
        \si@num@exp}}%
  \fi
```

With everything done, the result is output.

```
  \ifsi@num@intab\else
    \expandafter\si@num@out
  \fi
```

If there is anything inside the "x-part", then there is now more work to do.

```
  \ifx\@empty\si@num@err\@empty\else
    \expandafter\si@num@procerr
  \fi
  \ifsi@num@erropen
    \expandafter\si@out@num\expandafter{%
      \ensuremath{\si@closeerr}}%
      \ifx\@empty\si@num@ambig\@empty\else
        \expandafter\si@out@num\expandafter{\si@num@ambig}%
        \renewcommand*{\si@num@ambig}{}%
    \fi
  \fi
  \si@num@erropenfalse
  \ifx\@empty\si@num@xpart\@empty\else
    \expandafter\si@num@sepxpart
  \fi}
```

`\si@num@findxpart` Before processing the number, any multiplied parts have to be found and removed. As there can be more than one product sign, the building process here has no error checking.

```
\def\si@num@findxpart#1#2\@empty{%
  \si@str@ifchrstr{#1}{\si@numprod}
    {\si@switchtrue\si@seperrfalse}{}%
  \ifsi@switch
    \protected@edef\si@num@xpart{\si@num@xpart#1}%
  \else
    \protected@edef\si@num@in{\si@num@in#1}%
  \fi
  \ifx\@empty#2\@empty\else
    \si@num@findxpart#2\@empty
  \fi}
```

`\si@num@sepmantexp` `#1` : number
`\si@num@mantexp` Splitting the mantissa and exponent first checks for characters to gobble, which are simply thrown away. For any other input, there are two possibilities. If the character is an exponent marker, then the package switches from collecting the mantissa to collecting the exponent (after a sanity check). All other characters are added to either the mantissa or the exponent, as appropriate.

```
\newcommand*{\si@num@sepmantexp}[1]{%
  \expandafter\si@num@mantexp#1\@empty\@empty}
\def\si@num@mantexp#1#2\@empty{%
  \si@str@ifchrstr{#1}{\si@numgobble}
    {\si@log@debug{Gobbling '#1' in \si@num@arg}}
    {\si@str@ifchrstr{#1}{\si@numexp}
       {\ifsi@switch
          \si@log@err{Duplicate exponent marker found}
            {Only a single exponent character \MessageBreak
            (from the list '\si@numexp')\MessageBreak may
            occur in a numerical argument}%
        \else
          \si@log@debug{Exponent marker '#1' found in
            '\si@num@arg'}%
        \fi
        \si@switchtrue}%
       {\ifsi@switch
          \expandafter\si@num@addexp
        \else
          \expandafter\si@num@addmnt
        \fi
        {#1}}}%
```

If the recursion has not bottomed out, another loop occurs.

```
  \ifx\@empty#2\@empty
    \expandafter\@gobble
  \else
    \expandafter\si@num@sepmantexp
  \fi
  {#2}}
```

`\si@num@addmnt` `#1` : char
`\si@num@addexp` To allow `\expandafter` use in the above, the actual addition to the appropriate macro is handled here. Two helper macros are needed.

```
\newcommand*{\si@num@addmnt}[1]{%
  \si@num@addmntexp{#1}{mant}{mantissa}}
\newcommand*{\si@num@addexp}[1]{%
  \si@num@addmntexp{#1}{exp}{exponent}}
```

`\si@num@addmntexp` `#1` : char
`#2` : either `mant` or `exp`
`#3` : info-text
Then the business end.

```
\newcommand*{\si@num@addmntexp}[3]{%
  \si@log@debug{Adding '#1' to #3 for '\si@num@arg'}%
  \expandafter\protected@edef\csname si@num@#2\endcsname{%
    \csname si@num@#2\endcsname#1}}
```

`\si@num@sepsign` `#1` : either `mant` or `exp`

The input is now tested for a sign. If one exists, it is transferred into the `\si@num@#1sign` storage macro, with the remained of the number in `\si@num@#1`. The only check made directly here is that there is something to process.

```
\newcommand*{\si@num@sepsign}[1]{%
  \expandafter\ifx\expandafter\@empty
    \csname si@num@#1\endcsname\@empty
    \expandafter\@gobble
  \else
    \expandafter\si@num@gensign
  \fi
  {#1}}
```

`\si@num@gensign` `#1` : either `mant` or `exp`

The sign generator starts by calling the procedure to check if the input contains a valid one- or two-digit sign. The results are returned as `\si@num@sign` and `\si@num@value`.

```
\newcommand*{\si@num@gensign}[1]{%
  \expandafter\expandafter\expandafter\si@num@findsign
    \csname si@num@#1\endcsname\@empty\@empty
```

If no sign has been found, then there may be a need to add one anyway.

```
  \ifx\@empty\si@num@sign\@empty
    \ifx\@empty\si@num@value\@empty
      \expandafter\expandafter\expandafter\@gobble
    \else
      \expandafter\expandafter\expandafter\si@num@addsign
    \fi
  \else
    \expandafter\@gobble
  \fi
  {#1}%
```

The appropriate storage areas are now assigned.

```
  \expandafter\let\csname si@num@#1sign\endcsname\si@num@sign
  \expandafter\let\csname si@num@#1\endcsname\si@num@value}
```

`\si@num@findsign` `\si@num@sign` `\si@num@value`

The first one or two characters of the mantissa or exponent may contain a sign. To test for this, the first two characters of the number are split off, and examined. Two characters are used so that `\pm` and `\mp` can be represented by `+-` and `-+`, respectively. To allow the user to alter the valid signs, but retain this conversion, the generic character test is used before checking specific matches.

```
\newcommand*{\si@num@sign}{}
\def\si@num@findsign#1#2#3\@empty{%
  \si@num@delplusfalse
  \si@str@ifchrstr{#1}{\si@numsign}{%
    \si@str@ifchrstr{#2}{\si@numsign}{%
      \if +#1%
        \if -#2%
          \si@log@debug{Found sign combination +- for
            `\si@num@arg'}%
          \renewcommand*{\si@num@sign}{{\si@pm}}%
```

```
        \else
          \si@log@inf{Unknown sign combination `#1#2'}%
          \renewcommand*{\si@num@sign}{{#1#2}}%
        \fi
      \else
        \if -#1%
          \if +#2%
            \si@log@debug{Found sign combination -+ for
              `\si@num@arg'}%
            \renewcommand*{\si@num@sign}{{\mp}}%
          \else
            \si@log@inf{Unknown sign combination `#1#2'}%
            \renewcommand*{\si@num@sign}{{#1#2}}%
          \fi
        \else
          \si@log@inf{Unknown sign combination `#1#2'}%
          \renewcommand*{\si@num@sign}{{#1#2}}%
        \fi
      \fi
      \protected@edef\si@num@value{#3}}%
```

Only one valid sign character.

```
      {\si@log@debug{Found single sign character `#1' for
         `\si@num@arg'}%
       \renewcommand*{\si@num@sign}{{#1}}%
       \if +#1%
         \ifsi@retainplus\else
           \expandafter\expandafter\expandafter\si@num@killsign
         \fi
       \fi
       \protected@edef\si@num@value{#2#3}}}%
```

No valid sign, so `\@empty` is returned for the sign .

```
    {\si@log@debug{No sign found for `\si@num@arg'}%
     \renewcommand*{\si@num@sign}{}%
     \protected@edef\si@num@value{#1#2#3}}}
```

`\ifsi@num@delplus` `\si@num@killsign` A simple spin-out to remove a plus sign. A flag is set as it might be useful to know this.

```
\newif\ifsi@num@delplus
\newcommand*{\si@num@killsign}{%
  \si@num@delplustrue
  \renewcommand*{\si@num@sign}{}}
```

`\si@num@addsign` `\si@num@asign` `#1` : either `mant` or `exp`

The macro to add a sign to an unsigned number has to check whether this is a mantissa or an exponent. The result is still placed in `\si@num@sign` for ease of processing later.

```
\newcommand*{\si@num@addsign}[1]{%
  \begingroup
    \renewcommand*{\si@tempa}{#1}%
    \renewcommand*{\si@tempb}{mant}%
    \ifx\si@tempa\si@tempb
      \aftergroup\@firstoftwo
```

```
    \else
      \aftergroup\@secondoftwo
    \fi
  \endgroup
  {\ifsi@num@signmant
     \expandafter\si@num@asign
   \else
     \expandafter\@gobble
   \fi
   {mantissa}}
  {\ifsi@num@signexp
     \expandafter\si@num@asign
   \else
     \expandafter\@gobble
   \fi
   {exponent}}}
\newcommand*{\si@num@asign}[1]{%
  \let\si@num@sign\si@sign
  \si@log@debug{Adding sign \si@sign\space to #1 for
    `\si@num@arg'}}
```

`\si@num@procnum` `#1` : either `mant` or `exp`

The control macro for processing the number (plus any extra characters).

```
\newcommand*{\si@num@procnum}[1]{%
  \expandafter\ifx\expandafter\@empty
    \csname si@num@#1\endcsname\@empty
    \expandafter\@gobble
  \else
    \expandafter\si@num@finddigits
  \fi
  {#1}}
```

`\si@num@predec` `\si@num@postdec` Two new storage areas are defined.

```
\newcommand*{\si@num@predec}{}
\newcommand*{\si@num@postdec}{}
```

`\si@num@finddigits` `#1` : either `mant` or `exp`

The core digit processor divides the number into the parts before and after the decimal point marker. The temporary switch is used to indicate finding a decimal marker.

```
\newcommand*{\si@num@finddigits}[1]{%
  \renewcommand*{\si@num@predec}{}%
  \renewcommand*{\si@num@postdec}{}%
  \si@switchfalse
  \expandafter\expandafter\expandafter\si@num@digits
    \csname si@num@#1\endcsname\@empty\@empty
```

Tests are now made to see if padding zeros are needed. The trailing test needs to verify if a decimal marker was found, as well as if a zero is needed.

```
  \ifx\@empty\si@num@predec\@empty
    \ifsi@num@padlead
      \expandafter\expandafter\expandafter\si@num@addprezero
    \fi
  \fi
```

```
\ifx\@empty\si@num@postdec\@empty
  \ifsi@num@padtrail
    \ifsi@switch
      \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \si@num@addpostzero
    \fi
  \fi
\fi
```

The next checks to make concern input validity in a more mathematical sense. First, if the number is zero, then no sign should be given under any circumstances. Then leading zeros need to be removed from the input. This is slightly complicated by the potential presence of "extra" characters.

```
\si@num@unsign{#1}%
\ifx\@empty\si@num@predec\@empty
\else
  \expandafter\si@num@nozero
\fi
```

A sanity check is made to ensure that the supplied number consisted of more than a decimal marker.

```
\ifx\@empty\si@num@predec\@empty
  \ifx\@empty\si@num@postdec\@empty
    \expandafter\expandafter\expandafter\@gobble
  \else
    \expandafter\expandafter\expandafter\si@num@sepdigits
  \fi
\else
  \expandafter\si@num@sepdigits
\fi
{#1}}
```

`\si@num@digits` The `\si@num@digits` macro compares each character in the input against the list of characters valid at this stage: numbers, decimal markers and "extra" characters.

```
\def\si@num@digits#1#2\@empty{%
  \si@str@ifchrstr{#1}{\si@numdecimal}
    {\ifsi@switch
       \si@log@err{Duplicate decimal marker in `\si@num@arg'}
         {Only a single decimal marker (from the list
            `\si@numdecimal')\MessageBreak may occur in a
            numerical argument}%
     \else
       \si@log@debug{Found decimal marker `#1' in
         `\si@num@arg'}%
       \expandafter\si@switchtrue
     \fi}
```

The earlier code only checks for a sign at the start of the text. A check is therefore needed for a sign after the first two characters; if one is found, it is ignored.

```
    {\si@str@ifchrstr{#1}{\si@numsign}
       {\si@log@err{Misplaced sign character
          `#1' in `\si@num@arg'}
          {Sign characters `\si@numsign' can only
```

```
1619            occur\MessageBreak at the start of a number}}
```

The current character is added to the appropriate stack; this is "spun out" to avoid problems with expansion of the switch code.

```
1620        {\ifsi@switch
1621           \expandafter\si@num@post
1622         \else
1623           \expandafter\si@num@pre
1624         \fi
1625         {#1}}}%
1626   \ifx\@empty#2\@empty\else
1627     \si@num@digits#2\@empty\@empty
1628   \fi}
```

`\si@num@pre` `\si@num@post`

`#1` : char

Storage of the result is spun out.

```
1629 \newcommand*{\si@num@pre}[1]{%
1630   \si@num@prepost{#1}{pre}{integer}}
1631 \newcommand*{\si@num@post}[1]{%
1632   \si@num@prepost{#1}{post}{decimal}}
```

`\si@num@prepost`

`#1` : char

`#2` : either `pre` or `post`

`#3` : info-text

Then actually stored here.

```
1633 \newcommand*{\si@num@prepost}[3]{%
1634   \expandafter\protected@edef\csname si@num@#2dec\endcsname{%
1635     \csname si@num@#2dec\endcsname#1}%
1636   \si@log@debug{Adding `#1' to #3 part for `\si@num@arg'}}
```

`\si@num@addprezero` `\si@num@addpostzero`

A similar set of macros are used for the padding zeros.

```
1637 \newcommand*{\si@num@addprezero}{%
1638   \si@num@addpzero{pre}{leading}}
1639 \newcommand*{\si@num@addpostzero}{%
1640   \si@num@addpzero{post}{trailing}}
```

`\si@num@addpzero`

`#1` : either `pre` or `post`

In this case, there is no argument to be passed along.

```
1641 \newcommand*{\si@num@addpzero}[2]{%
1642   \si@log@debug{Adding #2 zero for `\si@num@arg'}%
1643   \@namedef{si@num@#1dec}{0}}
```

`\si@num@unsign`

`#1` : either `mant` or `exp`

`\si@num@nosign`

The trap for a sign with zero numerical input is made. First, a check is made to see if there is a sign to worry about. The pre- and post-decimal parts are then examined, to see if they contain something other than "0" or an extra character.

```
1644 \newcommand*{\si@num@unsign}[1]{%
1645   \expandafter\ifx\expandafter\@empty
1646     \csname si@num@#1sign\endcsname\@empty
1647     \expandafter\@gobble
1648   \else
1649     \expandafter\si@num@nosign
1650   \fi
```

```
  {#1}}
\newcommand*{\si@num@nosign}[1]{%
  \begingroup
    \si@switchtrue
    \si@str@ifonlychrs{\si@num@predec\si@num@postdec}{0}
      {\si@switchfalse}{}%
    \ifsi@switch
      \aftergroup\@gobble
    \else
      \aftergroup\@firstofone
    \fi
  \endgroup
  {\si@log@debug{Zero value: removing any sign}%
   \ifsi@ang@sign\else
     \@namedef{si@num@#1sign}{}%
   \fi}}
```

`\si@num@nozero` A very short test for a totally zero pre-decimal component.

```
\newcommand*{\si@num@nozero}{%
  \si@str@ifonlychrs{\si@num@predec}{0}
    {\renewcommand*{\si@num@predec}{0}}{}}
```

`\si@num@decimalhook` A hook is needed to attach things inside the group to happen afterwards, if the number is a decimal.

```
\newcommand*{\si@num@decimalhook}{}
```

`\si@num@sepdigits` `#1` : either `mant` or `exp`

The `\si@num@sepdigits` macro is only called if at least one of the mantissa and exponent contain something to output.

```
\newcommand*{\si@num@sepdigits}[1]{%
```

First an overall check is needed for additional characters. By altering the contents of `\si@numextra`, the same code can be shared by two different checks.

```
  \begingroup
    \let\si@numextra\si@numaddn
    \protected@edef\si@tempa{\si@num@predec\si@num@postdec}%
    \si@num@ifextra{\si@tempa}
      {\aftergroup\@gobble}
      {\aftergroup\@firstofone}%
  \endgroup
```

Separation of the error in a number from the number itself is only attempted for the mantissa.

```
  {\renewcommand*{\si@tempb}{mant}%
   \renewcommand*{\si@tempc}{#1}%
   \ifx\si@tempb\si@tempc
     \expandafter\si@num@checkerr
   \fi}%
```

If both parts of the number contain only digits, then any rounding can be attempted if there is no error part. Otherwise, the input must be left alone.

```
  \protected@edef\si@tempa{\si@num@predec\si@num@postdec}%
  \expandafter\si@str@ifonlychrs\expandafter{\si@tempa}
    {0123456789}
```

```
    {\ifx\@empty\si@num@err\@empty
       \ifsi@fixdp
         \expandafter\expandafter\expandafter\si@num@fixdp
       \fi
     \fi}{}%
```

If the pre-decimal part contains nothing except numbers, then digit separation is carried out.

```
  \si@num@ifextra{\si@num@predec}{}
    {\expandafter\si@num@int\expandafter{\si@num@predec}}%
```

A decision is made about the decimal sign, then digit separation occurs on the post-decimal part of the number.

```
  \renewcommand*{\si@tempc}{}%
  \ifx\@empty\si@num@postdec\@empty\else
    \si@num@decimalhook
    \renewcommand*{\si@tempc}{%
      \ensuremath{{\si@decimalsymbol}}}%
    \si@num@ifextra{\si@num@postdec}{}
      {\expandafter\si@num@dec\expandafter{\si@num@postdec}}%
  \fi
```

The construction is finalised by re-combining the number.

```
  \expandafter\protected@edef\csname si@num@#1\endcsname
    {\si@num@predec\si@tempc\si@num@postdec}}
```

`\si@num@ifextra` `#1` : number
`\si@num@extra` A relatively simple test for "extra" characters. Once again, a bit of group trickery is used.

```
\newcommand*{\si@num@ifextra}[1]{%
  \begingroup
    \si@switchfalse
    \expandafter\si@num@extra#1\@empty\@empty
    \ifsi@switch
      \si@log@debug{Found 'extra' characters in '#1'}%
      \aftergroup\@firstoftwo
    \else
      \aftergroup\@secondoftwo
    \fi
  \endgroup}
\def\si@num@extra#1#2\@empty{%
  \ifx\@empty#1\@empty\else
    \si@str@ifchrstr{#1}{\si@numextra}{\si@switchtrue}{}%
    \ifx\@empty#2\@empty\else
      \si@num@extra#2\@empty\@empty
    \fi
  \fi}
```

`\ifsi@num@ambigerr` When separating out an error from a number, the first step is to see if there is a
`\si@num@checkerr` decimal part to the number. If so, any error must be in that part of the decimal part; it is not possible to have an error starting in the integer part and continuing into the decimal part.

```
\newif\ifsi@num@ambigerr
\newcommand*{\si@num@checkerr}{%
```

```
  \ifx\@empty\si@num@postdec\@empty
    \expandafter\si@num@preerr
  \else
    \expandafter\si@num@posterr
  \fi}
```

`\si@num@preerr` Errors in integers are easy to handle. After finding the error, the result is simply stored in the error macro `\si@num@err`.

```
\newcommand*{\si@num@preerr}{%
  \si@num@seperr{pre}%
  \ifx\@empty\si@tempb\@empty\else
    \expandafter\renewcommand\expandafter*\expandafter
      \si@num@err\expandafter{\si@tempb}%
  \fi}
```

`\si@num@posterr` `\si@num@psterr` Life is more complex for an error in the decimal part. This is found, then it may need zero-filling or the insertion of a decimal point. This depends on whether the number of error digits is larger than the number of post-decimal digits.

```
\newcommand*{\si@num@posterr}{%
  \si@num@seperr{post}%
  \ifx\@empty\si@tempb\@empty\else
    \ifsi@seperr
      \expandafter\expandafter\expandafter\si@num@psterr
    \else
      \let\si@num@err\si@tempb
    \fi
  \fi}
\newcommand*{\si@num@psterr}{%
    \si@num@cntdigits{\si@tempb}%
    \si@tempcntb\si@tempcnta\relax
    \si@num@cntdigits{\si@num@postdec}%
    \ifnum\si@tempcnta<\si@tempcntb\relax
      \expandafter\si@num@largeerr
    \else
      \expandafter\si@num@smallerr
    \fi}
```

`\si@num@seperr` `#1` : either `pre` or `post`

`\si@num@finderr` The usual hand-off is made for searching for an error.

```
\newcommand*{\si@num@seperr}[1]{%
  \si@switchfalse
  \renewcommand*{\si@tempa}{}%
  \renewcommand*{\si@tempb}{}%
  \expandafter\expandafter\expandafter\si@num@finderr
    \csname si@num@#1dec\endcsname\@empty\@empty
  \ifx\@empty\si@tempb\@empty\else
    \expandafter\let\csname si@num@#1dec\endcsname\si@tempa
  \fi}
\def\si@num@finderr#1#2\@empty{%
```

First a check for the opening character of an error.

```
  \si@str@ifchrstr{#1}{\si@numopenerr}
```

If the switch is set when an error is found, then something is wrong.

```
    {\ifsi@switch
       \si@log@err{Invalid error in number}
         {The numerical argument \si@num@arg\space has two (or
           more)\MessageBreak error-opening characters}%
     \else
       \expandafter\si@switchtrue
     \fi}
```

The end-of-error character has to be the last item of the input, and an error needs to start before it ends.

```
    {\si@str@ifchrstr{#1}{\si@numcloseerr}
       {\ifsi@switch
          \ifx\@empty#2\@empty\else
            \si@log@err{Invalid error in number}
              {The numerical argument \si@num@arg\space has an
              error-closing before the last character}%
          \fi
        \else
          \si@log@err{Invalid error in number}
           {The numerical argument \si@num@arg\space has an
              error-closing character\MessageBreak but no
              error-opening one}%
        \fi}
```

The input must be a digit. It is stored in the appropriate area.

```
       {\ifsi@switch
          \expandafter\si@num@addtmpb
        \else
          \expandafter\si@num@addtmpa
        \fi
        {#1}}}%
  \ifx\@empty#2\@empty\else
    \si@num@finderr#2\@empty
  \fi}
```

`\si@num@addtmpa` `\si@num@addtmpb`

`#1` : number

Some quick methods for adding to the temporary macros with `\if` expansion.

```
\newcommand*{\si@num@addtmpa}[1]{\si@num@addtmp{a}{#1}}
\newcommand*{\si@num@addtmpb}[1]{\si@num@addtmp{b}{#1}}
```

`\si@num@addtmp`

`#1` : either `a` or `b`

`#2` : number

The addition is committed.

```
\newcommand*{\si@num@addtmp}[2]{%
  \expandafter\protected@edef\csname si@temp#1\endcsname{%
    \csname si@temp#1\endcsname#2}}
```

`\si@num@cntdigits` `\si@num@cntdgt`

`#1` : number

A recursive system for counting the number of characters in a given input; only used here to count digits in a number.

```
\newcommand*{\si@num@cntdigits}[1]{%
  \si@tempcnta\z@\relax
  \expandafter\si@num@cntdgt#1\@empty\@empty}
\def\si@num@cntdgt#1#2\@empty{%
```

```
  \ifx\@empty#1\@empty\else
    \advance\si@tempcnta\@ne\relax
  \fi
  \ifx\@empty#2\@empty\else
    \expandafter\si@num@cntdgt#2\@empty
  \fi}
```

`\si@num@smallerr` `\si@num@serr` For handling an error with no more digits than the post-decimal part of a number, zero-padding is undertaken before adding a decimal sign and (possibly) leading zero.

```
\newcommand*{\si@num@smallerr}{%
  \si@tempcntb\si@tempcnta\relax
  \si@num@serr
  \protected@edef\si@num@err{%
    \ifsi@num@padlead0\fi\expandafter\@car\si@numdecimal\@nil
    \si@tempb}}
\newcommand*{\si@num@serr}{%
  \si@num@cntdigits{\si@tempb}%
  \ifnum\si@tempcnta=\si@tempcntb\relax\else
    \protected@edef\si@tempb{0\si@tempb}%
    \expandafter\si@num@serr
  \fi}
```

`\si@num@largeerr` `\si@num@lerr` `\si@num@movedigit` When the error has more digits than the decimal part, some shuffling is needed.

```
\newcommand*{\si@num@largeerr}{%
  \renewcommand*{\si@tempa}{}%
  \si@tempcntb\si@tempcnta\relax
  \si@num@lerr
  \protected@edef\si@num@err{%
    \si@tempa\ensuremath{\si@decimalsymbol}\si@tempb}}
\newcommand*{\si@num@lerr}{%
  \si@num@cntdigits{\si@tempb}%
  \ifnum\si@tempcnta=\si@tempcntb\relax\else
    \expandafter\si@num@movedigit\si@tempb\@empty\@empty
    \si@num@lerr
  \fi}
\def\si@num@movedigit#1#2\@empty{%
  \protected@edef\si@tempa{\si@tempa#1}%
  \protected@edef\si@tempb{#2}}
```

`\si@num@fixdp` The test for fixing decimal places starts by counting up the number of decimal digits. The number is then padded, rounded or left alone.

```
\newcommand*{\si@num@fixdp}{%
  \si@num@cntdigits{\si@num@postdec}%
  \ifx\@empty\si@num@postdec\@empty
    \si@tempcnta\z@\relax
  \fi
  \ifnum\si@tempcnta>\si@num@dp\relax
    \expandafter\si@num@round
  \else
    \ifnum\si@tempcnta<\si@num@dp\relax
      \expandafter\expandafter\expandafter\si@num@pad
    \fi
  \fi}
```

\si@num@pad
\si@num@pd

Padding a number is a relatively easy matter. The number of digits is increased by adding "0" recursively.

```
\newcommand*{\si@num@pad}{%
  \si@log@debug{Padding to \the\si@num@dp\space digits}%
  \loop\ifnum\si@tempcnta<\si@num@dp\si@num@pd\repeat}
\newcommand*{\si@num@pd}{%
  \advance\si@tempcnta\@ne\relax
  \protected@edef\si@num@postdec{\si@num@postdec0}}
```

\si@num@round
\si@num@prernd
\si@num@postrnd

Rounding a number is more complicated. The main macro here relies on several others, and is simply a set-up and hand-off system.

```
\newcommand*{\si@num@prernd}{}
\newcommand*{\si@num@postrnd}{}
\newcommand*{\si@num@round}{%
  \si@log@debug{Rounding to \the\si@num@dp\space digits}%
  \si@num@reverse{\si@num@postdec}%
  \si@num@reverse{\si@num@predec}%
  \let\si@num@prernd\si@num@predec
  \let\si@num@postrnd\si@num@postdec
  \renewcommand*{\si@num@predec}{}%
  \renewcommand*{\si@num@postdec}{}%
  \si@switchfalse
  \si@num@rnd}
```

\si@num@reverse
\si@num@rev

`#1` : storage-macro

The initial stage is to reverse the entire number, to make life easier. This is then undone by the other macros as the output is constructed.

```
\newcommand*{\si@num@reverse}[1]{%
  \renewcommand*{\si@tempa}{}%
  \expandafter\si@num@rev#1\@empty\@empty
  \let#1\si@tempa}
\def\si@num@rev#1#2\@empty{%
  \edef\si@tempa{#1\si@tempa}%
  \ifx\@empty#2\@empty\else
    \si@num@rev#2\@empty\@empty
  \fi}
```

\si@num@rnd
\si@num@rndpre
\si@num@rndpost

The core looping macro of the system simply divides the flow between rounding before and after the decimal. The two routines are quite similar, but have subtly different requirements. Both take one character at a time from the input, increment if there is a carry digit, check its value, and add to the output string. This is very similar to the same routines used in other macro packages to achieve the same thing.

```
\newcommand*{\si@num@rnd}{%
  \ifnum\si@tempcnta>\z@\relax
    \expandafter\si@num@rndpost
  \else
    \expandafter\si@num@rndpre
  \fi}
\newcommand*{\si@num@rndpre}{%
  \expandafter\edef\expandafter\si@tempa\expandafter{%
    \expandafter\@car\si@num@prernd\@nil}%
  \expandafter\edef\expandafter\si@num@prernd\expandafter{%
```

```
    \expandafter\@cdr\si@num@prernd\@nil}%
  \si@tempcntb\si@tempa\relax
  \ifsi@switch
    \advance\si@tempcntb\@ne\relax
  \fi
  \si@switchfalse
  \ifnum\si@tempcntb=10\relax
    \si@tempcntb\z@\relax
    \expandafter\expandafter\expandafter\si@switchtrue
  \fi
  \edef\si@num@predec{\the\si@tempcntb\si@num@predec}%
  \ifx\@empty\si@num@prernd\@empty
    \ifsi@switch
      \edef\si@num@predec{1\si@num@predec}%
    \fi
  \else
    \expandafter\si@num@rnd
  \fi}
\newcommand*{\si@num@rndpost}{%
  \expandafter\edef\expandafter\si@tempa\expandafter{%
    \expandafter\@car\si@num@postrnd\@nil}%
  \expandafter\edef\expandafter\si@num@postrnd\expandafter{%
    \expandafter\@cdr\si@num@postrnd\@nil}%
  \si@tempcntb\si@tempa\relax
  \ifsi@switch
    \advance\si@tempcntb\@ne\relax
  \fi
  \si@switchfalse
  \advance\si@num@dp\@ne\relax
  \ifnum\si@tempcnta>\si@num@dp\relax
    \advance\si@num@dp\m@ne\relax
  \else
    \advance\si@num@dp\m@ne\relax
    \ifnum\si@tempcnta>\si@num@dp\relax
      \ifnum\si@tempcntb>4\relax
        \expandafter\expandafter\expandafter\si@switchtrue
      \fi
    \else
      \ifnum\si@tempcntb=10\relax
        \si@tempcntb\z@\relax
        \expandafter\expandafter\expandafter\si@switchtrue
      \fi
      \edef\si@num@postdec{\the\si@tempcntb\si@num@postdec}%
    \fi
  \fi
  \advance\si@tempcnta\m@ne\relax
  \si@num@rnd}
```

`\si@num@int` `#1` : integer-part

The formatting code for separating thousands is taken more-or-less directly from SIstyle. A few changes are made to fit the various conventions here. Following on from the code above, `\si@tempa` is used to store the integer part of the number, and `\si@tempb` is used for the decimal part.

```
\newcommand*{\si@num@int}[1]{%
```

```
  \renewcommand*{\si@num@predec}{}%
  \ifsi@sepfour
    \si@num@intfmt{}#1\@empty\@empty\@empty
  \else
    \si@num@iffive{#1}
      {\si@num@intfmt{}#1\@empty\@empty\@empty}
      {\renewcommand*{\si@num@predec}{#1}}%
  \fi}
```

`\si@num@iffive` `#1` : number
`\si@num@five` A test is needed for the presence of more than four characters.

```
\newcommand*{\si@num@iffive}[1]{%
  \si@num@five#1\@empty\@empty\@empty\@empty\@empty\end}
\def\si@num@five#1#2#3#4#5\end{%
   \ifx\@empty#5\@empty
      \expandafter\@secondoftwo
   \else
      \expandafter\@firstoftwo
   \fi}
```

`\si@num@intfmt` The business end of the integer formatter.
`\si@num@fiint`

```
\newcommand*{\si@num@intfmt}[4]{%
  \ifx\@empty#2\@empty
    \si@num@intsep#1\relax
  \else
    \ifx\@empty#3\@empty
      \si@num@intsep\@empty\@empty#1#2\relax
    \else
      \ifx\@empty#4\@empty
        \si@num@intsep\@empty#1#2#3\relax
      \else
        \si@num@fiint{#1#2#3#4}%
      \fi
    \fi
  \fi}
\def\si@num@fiint#1\fi\fi\fi{\fi\fi\fi\si@num@intfmt{#1}}
```

`\si@num@intsep` For adding separation to integers, an extra function is needed.

```
\newcommand*{\si@num@intsep}[4]{%
  \protected@edef\si@num@predec{\si@num@predec#1#2#3}%
  \if\relax#4\relax\else
    \protected@edef\si@num@predec{%
      \si@num@predec\ensuremath{\noexpand\si@digitsep}}%
    \expandafter\si@num@intsep\expandafter#4%
  \fi}
```

`\si@num@dec` `#1` : decimal-part
`\si@num@decfmt` Formatting a decimal uses a similar mechanism, but with a few alterations needed.

```
\newcommand*{\si@num@dec}[1]{%
  \renewcommand*{\si@num@postdec}{}%
  \ifsi@sepfour
    \si@num@decfmt#1\@empty\@empty\@empty\@empty
```

```
  \else
    \si@num@iffive{#1}
      {\si@num@decfmt#1\@empty\@empty\@empty\@empty}
      {\protected@edef\si@num@postdec{\si@num@postdec#1}}%
  \fi}
\newcommand*{\si@num@decfmt}[4]{%
  \protected@edef\si@num@postdec{\si@num@postdec#1#2#3}%
  \ifx\@empty#4\@empty%
  \else
    \protected@edef\si@num@postdec{%
      \si@num@postdec\ensuremath{\noexpand\si@digitsep}}%
    \expandafter\si@num@decfmt\expandafter#4%
  \fi}
```

`\si@num@procerr` Any error is recycled to to formatted correctly. The ± sign, and any unit, are also added.

```
\newcommand*{\si@num@procerr}{%
  \si@num@addunit
  \ensuremath{\si@pm}%
  \expandafter\si@num\expandafter{\si@num@err}}
```

`\si@num@sepxpart` A similar approach for numbers containing products, except the first token of the input has to be deleted.

```
\newcommand*{\si@num@sepxpart}{%
  \si@num@addunit
  \ensuremath{\times}%
  \expandafter\expandafter\expandafter\si@num\expandafter
    \expandafter\expandafter{%
      \expandafter\@cdr\si@num@xpart\@nil}}
```

`\si@num@addunit` A short macro to add units if needed.

```
\newcommand*{\si@num@addunit}{%
  \si@unt@numtrue
  \ifx\@empty\si@unt@unitarg\@empty\else
    \ifsi@repeatunits
      \si@unt@printunit{\si@unt@unitarg}%
    \fi
  \fi}
```

21.11 Formatting angles

`\ang [#1]` : keyval options
`#2` : angle, either in decimal or deg;min;sec format
The approach used here is similar to that in SIstyle, but has been modified in a few ways.

```
\si@newrobustcmd*{\ang}[2][]{%
  \begingroup
    \sisetup{#1}%
    \si@fam@mode
    \si@log@debug{Processing \string\ang\space input `#2'}%
    \@makeother{\;}%
    \makeatletter
    \scantokens{\si@ang@parse#2;;;\@nil}}
```

`\si@ang@parse` With the correct catcodes in place, processing can take place strip out the semi-colons. Here, the input must either contain no semi-colons or two semi-colons.

```
\def\si@ang@parse#1;#2;#3;#4\@nil{%
  \let\ifsi@ang@fixdp\ifsi@fixdp
  \si@fixdpfalse
  \si@ifmtarg{#4}
    {\si@log@debug{Angle argument contains no
       semi-colons:\MessageBreak decimal angle}%
     \si@ang@dec{#1}{}{}}
    {\si@log@debug{Angle argument contains
       semi-colons:\MessageBreak degree-minute-second angle}%
     \renewcommand*{\si@tempa}{#4}%
     \renewcommand*{\si@tempb}{;;}%
     \ifx\si@tempa\si@tempb\else
       \ifsi@strictarc
         \renewcommand*{\si@tempb}{;}%
         \ifx\si@tempa\si@tempb
           \si@log@err{Insufficient semi-colons in argument
             of \string\ang}{The argument of
             \string\ang\space  must contain either no
             semi-colons or exactly two}%
         \else
           \si@log@err{Excess semi-colons in argument of
             \string\ang}{The argument of \string\ang\space
             must contain either no semi-colons or exactly
             two}%
         \fi
       \fi
     \fi
     \si@ang@arc{#1}{#2}{#3}}}
```

`\si@ang@dec` Two tests are needed, in case the input format requires conversion.

`\si@ang@arc`

```
\newcommand*{\si@ang@dec}{%
  \let\si@ang@fix\@gobble
  \ifsi@ang@toarc
    \expandafter\si@ang@dectoarc
  \else
    \sisetup{padangle=none}\expandafter\si@ang@typeset
  \fi}
\newcommand*{\si@ang@arc}{%
  \let\si@ang@fix\si@ang@arcfix
  \ifsi@ang@todec
    \expandafter\si@ang@arctodec
  \else
    \expandafter\si@ang@typeset
  \fi}
```

`\ifsi@ang@fixdp` `#1` : number

`\si@ang@fix` `\si@ang@arcfix` A check is needed so that rounding and zero filling are only applied to the seconds of an arc angle.

```
\newif\ifsi@ang@fixdp
\newcommand*{\si@ang@fix}[1]{}
\newcommand*{\si@ang@arcfix}[1]{%
```

```
  \renewcommand*{\si@tempa}{second}%
  \renewcommand*{\si@tempb}{#1}%
  \ifx\si@tempa\si@tempb
    \ifsi@ang@fixdp
      \expandafter\expandafter\expandafter\si@fixdptrue
    \else
      \expandafter\expandafter\expandafter\si@fixdpfalse
    \fi
  \else
    \expandafter\si@fixdpfalse
  \fi}
```

`\si@ang@ifnum` `#1` : number

A test is required to check that the provided data consists of numbers which can actually be processed by TEX. This is achieved by using `\si@num@ifvalid` with a fixed list of valid characters.

```
\newcommand*{\si@ang@ifnum}[1]{%
  \begingroup
    \renewcommand*{\si@numvalid}{0123456789,.+-}%
    \ifx\@empty#1\@empty
      \aftergroup\@firstoftwo
    \else
      \si@num@ifvalid{#1}
        {\aftergroup\@firstoftwo}
        {\aftergroup\@secondoftwo}%
    \fi
  \endgroup}
```

`\si@ang@arctodec` `#1` : degrees
`#2` : minutes
`#3` : seconds

The business end of converting arcs to decimal angles is relatively straightforward, as it only needs one calculation. This has to be divided up, so that disaster does not strike if there are empty arguments; a check is also needed for the sign of the angle, so that the maths makes sense.

```
\newcommand*{\si@ang@arctodec}[3]{%
  \let\si@ang@fix\@gobble
  \ifnum\si@num@dp>\thr@@\relax
    \si@num@dp\thr@@\relax
  \fi
  \si@fixdptrue
  \si@ang@ifnum{#1}
    {\si@ang@ifnum{#2}
      {\si@ang@ifnum{#3}
        {\si@tempdima\z@\relax
         \renewcommand*{\si@tempa}{+}%
         \ifx\@empty#1\@empty\else
           \si@tempdima #1pt\relax
         \fi
         \ifdim\si@tempdima<\z@\relax
           \renewcommand*{\si@tempa}{-}%
         \fi
         \ifx\@empty#2\@empty\else
```

```
            \si@tempdima\dimexpr\si@tempdima\si@tempa
              #2pt/60\relax
          \fi
          \ifdim\si@tempdima<\z@\relax
            \renewcommand*{\si@tempa}{-}%
          \else
          \fi
          \ifx\@empty#3\@empty\else
            \si@tempdima\dimexpr\si@tempdima\si@tempa
              #3pt/3600\relax
          \fi
          \sisetup{numdecimal=.}%
          \expandafter\si@ang@typeset\expandafter{%
            \strip@pt\si@tempdima}{}{}}
        {\si@ang@notnum{#1}{#2}{#3}}}
      {\si@ang@notnum{#1}{#2}{#3}}}
    {\si@ang@notnum{#1}{#2}{#3}}}
```

`\si@ang@dectoarc` `#1` : number

`\si@ang@arcdeg` `\si@ang@arcmin` `\si@ang@arcsec`

The conversion macros for decimal to arc angles. Life is more complex here than above, even without the need for checks on the input. A number of separation steps are needed, each of which needs a separate macro.

```
\newcommand*{\si@ang@dectoarc}[1]{%
  \let\si@ang@fix\si@ang@arcfix
  \si@ang@fixdptrue
  \ifnum\si@num@dp>\@ne\relax
    \si@num@dp\@ne\relax
  \fi
  \si@ang@ifnum{#1}
    {\si@tempdima\z@\relax
     \ifx\@empty#1\@empty\else
       \si@tempdima #1pt\relax
     \fi
     \si@ang@sepint{deg}%
     \si@tempdima\dimexpr\si@tempdima *60\relax
     \si@ang@sepint{min}%
     \edef\si@tempa{\the\dimexpr\si@tempdima *60\relax}%
     \expandafter\newcommand\expandafter*\expandafter{%
       \expandafter\si@ang@arcsec\expandafter}\expandafter{%
       \expandafter\si@ang@strippt\si@tempa}%
```

A check is made for "0.0" seconds, which should be converted to simply "0".

```
     \si@tempdima\z@\relax
     \edef\si@tempa{\the\si@tempdima}%
     \expandafter\renewcommand\expandafter*\expandafter{%
       \expandafter\si@tempa\expandafter}\expandafter{%
       \expandafter\si@ang@strippt\si@tempa}%
     \ifx\si@tempa\si@ang@arcsec
       \renewcommand*{\si@ang@arcsec}{0}%
     \fi
```

To avoid adding zeros where they are not required, each part of the angle is now checked.

```
     \renewcommand*{\si@tempa}{0}%
```

```
      \ifx\si@ang@arcdeg\si@tempa
        \si@temptoks{{}}%
      \else
        \si@temptoks{{\si@ang@arcdeg}}%
      \fi
      \ifx\si@ang@arcmin\si@tempa
        \si@temptoks\expandafter{\the\si@temptoks{}}%
      \else
        \si@temptoks\expandafter{\the\si@temptoks{%
          \si@ang@arcmin}}%
      \fi
      \ifx\si@ang@arcsec\si@tempa
        \si@temptoks\expandafter{\the\si@temptoks{}}%
      \else
        \si@temptoks\expandafter{\the\si@temptoks{%
          \si@ang@arcsec}}%
      \fi
      \expandafter\si@ang@typeset\the\si@temptoks}
    {\si@ang@notnum{#1}{}{}}}
```

`\si@ang@sepint` `#1` : either `deg` or `min`
`\si@ang@sint` Support macros for the conversion from decimal to arc angles.

```
\si@ang@strippt\newcommand*{\si@ang@sepint}[1]{%
  \expandafter\si@ang@sint\the\si@tempdima\@empty
  \expandafter\let\csname si@ang@arc#1\endcsname\si@tempa}
\def\si@ang@sint#1.#2\@empty{%
  \renewcommand*{\si@tempa}{#1}%
  \si@tempdima 0.#2\relax}
\begingroup
  \catcode`P=12
  \catcode`T=12
  \lowercase{
    \renewcommand*{\si@tempa}{%
      \def\si@ang@strippt##1PT{##1}}}
  \expandafter\endgroup
\si@tempa
```

`\si@ang@notnum` `#1` : degrees
`#2` : minutes
`#3` : seconds
Not a TEX number: complain.

```
\newcommand*{\si@ang@notnum}[3]{%
  \si@log@warn{Angle `#1;#2;#3' is not a pure
    number:\MessageBreak output will be as given}%
  \si@ang@typeset{#1}{#2}{#3}}
```

`\ifsi@ang@sign` A flag is needed to leave signs along for angles, where a zero value may still need a sign.

```
\newif\ifsi@ang@sign
```

`\si@ang@typeset` `#1` : degrees
`#2` : minutes
`#3` : seconds

The \si@ang@set macro does the work of assigning the degrees, minutes and seconds, and actually typesetting the result.

```
\newcommand*{\si@ang@typeset}[3]{%
```

\si@ang@degs
\si@ang@mins
\si@ang@secs

First, the three macros that will contain the measures must exist.

```
  \ifsi@ang@padlarge
    \newcommand*{\si@ang@degs}{0\si@sym@degree}%
    \newcommand*{\si@ang@mins}{0\si@sym@minute}%
    \newcommand*{\si@ang@secs}{0\si@sym@second}%
  \else
    \newcommand*{\si@ang@degs}{}%
    \newcommand*{\si@ang@mins}{}%
    \newcommand*{\si@ang@secs}{}%
  \fi
```

\si@ang@decimalsymbol

The current definition of \si@decimalsymbol needs to be saved.

```
  \protected@edef\si@ang@decimalsymbol{\si@decimalsymbol}%
```

\si@ang@movesign

Either the signs need to be moved, or this needs to be killed off.

```
  \ifsi@astroang
    \let\si@ang@movesign\si@ang@astrosign
  \else
    \let\si@ang@movesign\@gobble
  \fi
```

\si@ang@secnum
\si@ang@minnum

The arguments are now examined in reverse order. If they are empty, then nothing is done. Otherwise, the larger measures are zero-filled, if this has been requested. Some steps are needed to allow for addition of signs to numbers.

```
  \newcommand*{\si@ang@secnum}{\si@ang@num{second}}%
  \newcommand*{\si@ang@minnum}{\si@ang@num{minute}}%
   \si@ifnotmtarg{#3}
     {\si@log@debug{Found seconds `#3'}%
      \si@ang@ifnum{#3}
        {\ifdim #3 pt=\z@\relax\else
           \si@ang@signtrue
         \fi}{}%
      \renewcommand*{\si@ang@secs}
       {\si@ang@secnum{#3}\si@sym@second}%
      \renewcommand*{\si@ang@mins}
        {\si@ang@pad{0\si@sym@minute}}%
      \renewcommand*{\si@ang@degs}
        {\si@ang@pad{0\si@sym@degree}}}%
   \si@ifnotmtarg{#2}
     {\si@log@debug{Found minutes `#2'}%
      \si@ang@ifnum{#2}
        {\ifdim #2 pt=\z@\relax\else
           \si@ang@signtrue
         \fi}{}%
      \renewcommand*{\si@ang@secnum}{%
        \si@ang@signlessnum{second}}%
      \renewcommand*{\si@ang@mins}
       {\si@ang@minnum{#2}\si@sym@minute}%
      \renewcommand*{\si@ang@degs}
```

```
        {\si@ang@pad{0\si@sym@degree}}%
    \si@ifnotmtarg{#1}
      {\si@log@debug{Found degrees `#1'}%
       \renewcommand*{\si@ang@secnum}{%
         \si@ang@signlessnum{second}}%
       \renewcommand*{\si@ang@minnum}{%
         \si@ang@signlessnum{minute}}%
       \renewcommand*{\si@ang@degs}
```

The group here is needed to get the mechanism to move the symbol to work properly.

```
       {\si@ang@num{degree}{#1}%
        \si@sym@degree}}%
    \si@out@num
      {\si@ang@degs\si@anglesep\si@ang@mins\si@anglesep
         \si@ang@secs}%
```

The group opened by `\ang` is closed.

```
    \endgroup}
```

`\si@ang@pad` `#1` : number

Padding is only added if requested; the zero is a literal.

```
\newcommand*{\si@ang@pad}[1]{\ifsi@ang@padsmall #1\fi}
```

`\si@ang@num` `#1` : one of `degree`, `minute` or `second`

`\si@ang@signlessnum` `#2` : number

Modified versions of `\num`, one to typeset angles without a leading sign and the other with.

```
\newcommand*{\si@ang@num}[2]{%
  \begingroup
    \si@ang@fix{#1}%
    \si@ang@movesign{#1}%
    \si@num{#2}%
  \endgroup}
\newcommand*{\si@ang@signlessnum}[2]{%
  \begingroup
    \si@ang@fix{#1}%
    \si@ang@movesign{#1}%
    \sisetup{addsign=none}%
    \si@num{#2}%
  \endgroup}
```

`\si@ang@killdegree` `\si@ang@killminute` `\si@ang@killsecond`

A mechanism is needed to handle moving the angle unit signs for the `astroang` option. First, some support macros are needed.

```
\newcommand*{\si@ang@killdegree}{\let\si@sym@degree\relax}
\newcommand*{\si@ang@killminute}{\let\si@sym@minute\relax}
\newcommand*{\si@ang@killsecond}{\let\si@sym@second\relax}
```

`\si@ang@astrosign` `#1` : one of `degree`, `minute` or `second`

The method needs two steps, producing the sign over the decimal sign and preventing duplicate symbols appearing. This is based on a suggestion from Morten Høgholm, but using TEX internals as `\makebox` does not work here.

Note the need to correct for `\scriptspace` (thanks to Donald Arseneau for that).

```
\newcommand*{\si@ang@astrosign}[1]{%
  \renewcommand*{\si@decimalsymbol}{%
    \setbox\si@tempboxa=\hbox{%
      \ensuremath{{\si@ang@decimalsymbol}}}%
    \si@tempdima\wd\si@tempboxa\relax
    \setbox\si@tempboxb=\hbox to\z@{%
      \hss\unhbox\si@tempboxa\hss}%
    \setbox\si@tempboxa=\hbox{%
      \csname si@sym@#1\endcsname\hskip-\scriptspace}%
    \si@tempdimb\wd\si@tempboxa\relax
    \setbox\si@tempboxc=\hbox to\z@{%
      \hss\unhbox\si@tempboxa\hss}%
    \setbox\si@tempboxd=\hbox{%
      \usebox\si@tempboxb\usebox\si@tempboxc}%
    \ifdim\si@tempdima>\si@tempdimb\relax
      \setbox\si@tempboxa=\hbox to\si@tempdima{%
        \hss\unhbox\si@tempboxd\hss}%
    \else
      \setbox\si@tempboxa=\hbox to\si@tempdimb{%
        \hss\unhbox\si@tempboxd\hss}%
    \fi
    \usebox\si@tempboxa%
    \ifdim\si@tempdima>\si@tempdimb\relax\else
      \hskip\scriptspace
    \fi}%
  \renewcommand*{\si@num@decimalhook}{\expandafter\aftergroup
    \csname si@ang@kill#1\endcsname}}%
```

21.12 Tabular material

The automatic formatting and alignment of numerical data in columns is handled here. The various other packages that work in this area are basically ripped-off here. The letters `D`, `N` and `R` are already taken by the other packages for numerical alignment, and so `S` (= siunitx) is chosen for the alignment of numerical material. The package also provides a second column type, `s`, for units (the letter is taken from `\si`).

`\NC@list` The first part of the job is to create the basic apparatus for the columns using the array package. To prevent any issues with the content of optional arguments to the new columns, so rearrangement is needed. The siunitx columns have to come *before* any other column definitions. This is achieved by saving `\NC@list`, creating the columns then restoring the list with the appropriate extra parts.

```
\edef\si@tempa{%
  \noexpand\NC@do S\noexpand\NC@do s\the\NC@list}
\newcolumntype{S}{}
\newcolumntype{s}{}
\NC@list\expandafter{\si@tempa}
```

`\NC@rewrite@S [#1]`: keyval options

`\NC@rewrite@s` Following the numprint approach, the `\NC@rewrite@...` macros are rewritten to collect the cell contents. This means messing with the internal macros of

another package, but there is no other way to do this. As array is a standard package from the tools bundle, this should be reasonably safe. Here the begin and end code needed is added to the existing list if `\@temptokena`, with the start and end macros unexpanded. Argument `#1` contains any user setup options for this column. Passing an argument at this stage will cause issues, so each column type needs its own `begin` and `end` macros.

```
\renewcommand*{\NC@rewrite@S}[1][]{%
  \edef\si@tempa{\the\@temptokena
    >{\noexpand\si@tab@begin@S[#1]}c%
    <{\noexpand\si@tab@end@S}}%
  \@temptokena\expandafter{\si@tempa}%
  \NC@find}
\renewcommand*{\NC@rewrite@s}[1][]{%
  \edef\si@tempa{\the\@temptokena
    >{\noexpand\si@tab@begin@s[#1]}c%
    <{\noexpand\si@tab@end@s}}%
  \@temptokena\expandafter{\si@tempa}%
  \NC@find}
```

`\si@tab@begin@S [#1]`: keyval options

`\si@tab@begin@s`
`\si@tab@gettok`

At this stage, the appropriate token gathering macro is activated, and the common starting macro is called. For the `S` column, the `seperr` is turned off, and an error is set to be raised by any "x-part" input.

```
\newcommand*{\si@tab@begin@S}[1][]{%
  \si@log@debug{Processing S column cell contents}%
  \let\si@tab@gettok\si@tab@gettok@S
  \si@seperrfalse
  \renewcommand*{\si@num@sepxpart}{%
    \si@log@err{Multiple numbers not allowed in
       tables\MessageBreak Only the first number used}
       \@ehb}%
  \si@tab@begin[#1]}
\newcommand*{\si@tab@begin@s}[1][]{%
  \si@log@debug{Processing s column cell contents}%
  \let\si@tab@gettok\si@tab@gettok@s
  \si@tab@begin[#1]}
```

`\si@tab@toks`
`\si@tab@pretoks`
`\si@tab@posttoks`

Some storage is needed for the data to build up. In common with rccol and numprint, token registers are used for this (thus leaving problematic input to be handled later).

```
\newtoks\si@tab@toks
\newtoks\si@tab@pretoks
\newtoks\si@tab@posttoks
```

`\si@tab@begin [#1]`: keyval options

The macro for gathering up input is common to both column types. It uses the method for rccol; the cell contents are collected by a second macro, which then stores all of the data in an appropriate token store.

```
\newcommand*{\si@tab@begin}[1][]{%
  \begingroup
    \sisetup{#1}%
    \si@tab@toks{}%
```

```
    \si@tab@pretoks{}%
    \si@tab@posttoks{}%
    \si@switchfalse
    \si@tab@gettok}
```

`\si@tab@next` A macro is needed for recursion when collecting cell contents.

```
\newcommand*{\si@tab@next}{}
```

`\si@tab@gettok@S` `#1` : token

The cell-content collection system is inherited from rccol. First, the current input is compared with a fixed list of macros which need custom handling.

```
\newcommand*{\si@tab@gettok@S}[1]{%
  \ifx\tabularnewline#1\relax
    \let\si@tab@next\si@tab@newline@S
  \else
    \ifx\end#1\relax
      \let\si@tab@next\end
    \else
      \ifx\si@tab@end@S#1\relax
        \let\si@tab@next\si@tab@end@S
      \else
        \ifx\endtabular#1\relax
          \let\si@tab@next\endtabular
        \else
          \ifx\csname#1\relax
            \let\si@tab@next\csname
          \else
            \ifx\relax#1\relax
              \let\si@tab@next\relax
            \else
```

If the code gets to here, then recursion should occur.

```
              \let\si@tab@next\si@tab@gettok@S
```

Cells always contain `\ignorespaces` and `\unskip`. These can both be filtered out here and are saved to either the pre-number or post-number token store, as appropriate. This is done without fixing the destination in case the user adds addition copies to the input.

```
            \ifx\ignorespaces#1\relax
              \si@tab@othertok{#1}%
            \else
              \ifx\unskip#1\relax
                \si@tab@othertok{#1}%
              \else
```

If the current token is a valid numerical character, then the flag is set and the input is saved into the “processing” token register.

```
                \si@str@ifchrstr{#1}{\si@numvalid}
                   {\si@switchtrue
                    \si@log@debug{Found numerical cell
                      contents `#1'}%
                    \si@tab@toks=\expandafter{%
                      \the\si@tab@toks#1}}
```

For non-numerical content, some testing is needed. A check is made to see if the current "token" is a single item. If it was wrapped in one set of braces before being picked up here, they will have been removed. So the input may now have more than one character to it. If it is a single token, testing for expendability first requires a check for another set of braces. This is handed off using `\futurelet`.

```
                        {\si@tab@ifonechar{#1}
                          {\futurelet\si@tempa\si@tab@bracetest#1}%
                          {\si@tab@othertok{#1}}}%
                    \fi
                  \fi
                \fi
              \fi
            \fi
          \fi
        \fi
      \fi
    \si@tab@next}
```

`\si@tab@othertok` `#1` : token

Tokens which are not part of a number can occur before or after a number. This is all sorted out here, with some debugging information for good measure.

```
\newcommand*{\si@tab@othertok}[1]{%
  \si@log@debug{Found other cell contents `\unexpanded{#1}'}%
  \ifsi@switch
    \si@log@debug{Adding to post-numeral part}%
    \si@tab@posttoks=\expandafter{\the\si@tab@posttoks#1}%
  \else
    \si@log@debug{Adding to pre-numeral part}%
    \si@tab@pretoks=\expandafter{\the\si@tab@pretoks#1}%
  \fi}
```

`\si@tab@ifonechar` `#1` : token

`\si@tab@onechar` A quick check for a single character.

```
\newcommand*{\si@tab@ifonechar}[1]{\si@tab@onechar#1\@empty}
\def\si@tab@onechar#1#2\@empty{%
  \ifx\@empty#2\@empty
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
```

`\si@tab@bracetest` For the `\futurelet` test, a separate macro is needed. If the current token starts with a brace, then it is added to the "other" stack. If not, a test can be made on its definition.

```
\newcommand*{\si@tab@bracetest}{%
  \ifx\si@tempa\bgroup
    \expandafter\si@tab@othertok
  \else
    \expandafter\si@tab@exptest
  \fi}
```

`\si@tab@exptest` `#1` : token
`\si@tab@meaning` `\si@tab@strip` The meaning of the token is compared to that for a short macro. If the two are of the same type, expansion is attempted. In this case, the current pre-number tokens are also typeset.

```
\newcommand*{\si@tab@exptest}[1]{%
  \edef\si@tempa{\si@tab@meaning{#1}}%
  \ifx\si@tempa\si@tab@macro
    \si@log@debug{Expanding cell contents `\unexpanded{#1}'}%
    \the\si@tab@pretoks
    \si@tab@pretoks{}%
    \protected@edef\si@tab@next{\noexpand\si@tab@gettok@S#1}%
  \else
    \si@tab@othertok{#1}%
  \fi}
\newcommand*{\si@tab@meaning}[1]{%
  \expandafter\si@tab@strip\meaning#1:->:->\@empty}
\def\si@tab@strip#1:->#2:->#3\@empty{#1}
```

`\si@tab@macro` A supporter macro for the above which always expands to "macro" with everything with category code 12.

```
\newcommand*{\si@tab@macro}{}
\edef\si@tab@macro{\si@tab@meaning{\si@tab@macro}}
```

`\si@tab@gettok@s` `#1` : token
The situation for the `s` column is less complex, as there is only one store and expansion is not an issue.

```
\newcommand*{\si@tab@gettok@s}[1]{%
  \ifx\tabularnewline#1\relax
    \let\si@tab@next\si@tab@newline@s
  \else
    \ifx\end#1\relax
      \let\si@tab@next\end
    \else
      \ifx\si@tab@end@s#1\relax
        \let\si@tab@next\si@tab@end@s
      \else
        \ifx\endtabular#1\relax
          \let\si@tab@next\endtabular
        \else
          \ifx\csname#1\relax
            \let\si@tab@next\csname
          \else
            \ifx\relax#1\relax
              \let\si@tab@next\relax
            \else
              \let\si@tab@next\si@tab@gettok@s
              \ifx\ignorespaces#1\relax\else
                \ifx\unskip#1\relax\else
                  \si@tab@toks=\expandafter{%
                    \the\si@tab@toks#1}%
                  \si@log@debug{Found cell contents `#1'}%
                \fi
              \fi
```

```
            \fi
          \fi
        \fi
      \fi
    \fi
  \fi
  \si@tab@next}
```

`\si@tab@end@S` `\si@tab@rfill` `\si@tab@lfill` `\si@tab@end@s`

The end macros are similar, but with some minor differences. In both cases, the appropriate filling is carried out. For the `S` column, this depends on the cell contents, whereas for the `s` column the fill is always the same. Output of a number in an `S` column is only attempted if one was found, otherwise the cell contents will all be in `\si@tab@pretoks`.

```
\newcommand*{\si@tab@end@S}{%
    \ifsi@switch
      \let\si@tab@lfill\si@tab@lfill@S
      \let\si@tab@rfill\si@tab@rfill@S
    \else
      \let\si@tab@rfill\si@tab@rfill@t
      \let\si@tab@lfill\si@tab@lfill@t
    \fi
    \si@tab@lfill\relax
    \ifsi@switch
      \the\si@tab@pretoks
      \si@tab@numout
      \the\si@tab@posttoks
    \else
      \the\si@tab@pretoks
    \fi
    \si@tab@rfill\relax
  \endgroup}
\newcommand*{\si@tab@end@s}{%
    \si@tab@lfill@s\relax
    \ignorespaces
    \expandafter\si\expandafter{\the\si@tab@toks}%
    \unskip
    \si@tab@rfill@s\relax
  \endgroup}
```

`\si@tab@newline@S` `\si@tab@newline@s` `\si@tab@end`

If the column is the final one read, then some work is needed with the `\tabularnewline` macro. Output has to happen *before* the new line, then the ending macro is made safe before calling the LaTeX line end. If the user makes use of the primitive `\cr` then this problem does not arise as `\si@tab@end@...` is called correctly.

```
\newcommand*{\si@tab@newline@S}{%
  \si@tab@end@S
  \hfil\relax
  \let\si@tab@end\si@tab@end@S
  \renewcommand*{\si@tab@end@S}{\let\si@tab@end@S\si@tab@end}%
  \tabularnewline}
\newcommand*{\si@tab@newline@s}{%
  \si@tab@end@s
  \hfil\relax
```

```
  \let\si@tab@end\si@tab@end@s
  \renewcommand*{\si@tab@end@s}{\let\si@tab@end@s\si@tab@end}%
  \tabularnewline}
```

`\si@tempcnta` `\si@tempcntb` The second part of the tabular code is concerned with typesetting numbers in `S` columns with the appropriate alignment. Counters are needed for the digit-counting system.

```
\newcount\si@tempcnta
\newcount\si@tempcntb
```

`\si@tab@numout` If a number is found, then some secondary processing is needed to format it correctly.

```
\newcommand*{\si@tab@numout}{%
  \si@num@intabtrue
  \ifsi@tab@fixed
    \ifsi@tabautofit
      \si@num@dp\si@tab@mantpostcnt\relax
      \expandafter\expandafter\expandafter\si@fixdptrue
    \fi
  \fi
  \expandafter\si@num\expandafter{\the\si@tab@toks}%
  \si@tab@format}
```

`\si@tab@prebox` `\si@tab@postbox` `\si@tab@midbox` `\si@tab@expbox` The various boxes needed for the column centring are declared Unlike the dcolumn original, private boxes are used here. `\si@tempboxa` is used when a space to measure one of the constituents is needed; it is never used for output.

```
\newbox\si@tab@prebox
\newbox\si@tab@midbox
\newbox\si@tab@postbox
\newbox\si@tab@expbox
```

`\si@tab@format` The formatting set up is taken from dcolumn, but with control of the output form the stored information. The choice of a variable (decimal-centred) column or fixed width boxes is made.

```
\newcommand*{\si@tab@format}{%
  \ifsi@tab@fixed
    \expandafter\si@tab@fixed
  \else
    \expandafter\si@tab@unfixed
  \fi
```

A hack to get the correct colour everywhere without too much work.

```
  \ifsi@colourvalues
    \si@fam@colourcmd{\si@valuecolour}%
  \fi
  \box\si@tab@prebox\box\si@tab@midbox\box\si@tab@postbox%
    \box\si@tab@expbox}
```

`\si@tab@unfixed` When the width of the contents is not fixed, the system creates a block in which the decimal marker is always at the centre. This is achieved by placing the pre- and post-decimal parts of the number in boxes. The wider one is then used to set up the column width, by resizing the other one.

```
\newcommand*\si@tab@unfixed{%
```

```
\si@log@debug{Using variable width S column}%
\ifx\@empty\si@num@out\@empty
  \setbox\si@tab@midbox=\hbox
    {\phantom{\ensuremath{{\si@decimalsymbol}}}}%
\else
  \setbox\si@tab@midbox=\hbox
    {\ensuremath{{\si@decimalsymbol}}}%
\fi
\protected@edef\si@num@out{\si@num@out\si@tab@expout}%
\setbox\si@tab@prebox=\hbox
  {\expandafter\si@out@num\expandafter{\si@tab@out}}%
\setbox\si@tab@postbox=\hbox
  {\expandafter\si@out@num\expandafter{\si@num@out}}%
\ifdim\wd\si@tab@prebox>\wd\si@tab@postbox\relax
  \setbox\si@tab@postbox=\hbox to\wd\si@tab@prebox%
    {\unhbox\si@tab@postbox\hfill}%
\else
  \setbox\si@tab@prebox=\hbox to\wd\si@tab@postbox%
    {\hfill\unhbox\si@tab@prebox}%
\fi
\setbox\si@tab@expbox=\hbox{}}
```

`\si@tab@predim` Some storage dimensions are declared.

`\si@tab@postdim` `\si@tab@expdim` `\si@tempdima` `\si@tempdimb`

```
\newdimen\si@tab@predim
\newdimen\si@tab@postdim
\newdimen\si@tab@expdim
\newdimen\si@tempdima
\newdimen\si@tempdimb
```

`\si@tab@sp` A short macro to control superscript.

```
\newcommand*{\si@tab@sp}{}
```

`\si@tab@fixed` The column is not centred on the decimal marker; the user specifies how many characters on each side are allowed for. First, the width of a character is measured, and stored.

```
\newcommand*{\si@tab@fixed}{%
  \si@log@debug{Using fixed-width S column}%
  \let\si@tab@sp\relax
  \setbox\si@tab@midbox=\hbox{}%
  \setbox\si@tab@expbox=\hbox{}%
  \setbox\si@tempboxa=\hbox{\si@out@num{1}}%
  \si@tempdima\wd\si@tempboxa\relax
```

The width for the two output boxes is set up.

```
  \si@tab@predim\the\si@tab@mantprecnt\si@tempdima\relax
  \si@tab@sepcorr{mantpre}{pre}%
  \si@tab@postdim\si@tab@mantpostcnt\si@tempdima\relax
  \setbox\si@tempboxa=\hbox{\ensuremath{{\si@decimalsymbol}}}%
  \ifnum\si@tab@mantpostcnt=\z@\relax\else
    \advance\si@tab@postdim\wd\si@tempboxa\relax
  \fi
  \si@tab@sepcorr{mantpost}{post}%
```

If space is needed for an exponent, it needs to be allowed for in the exponent box dimension. First, the digits of the two parts are checked for; the width of a character is altered to be superscript.

```
\setbox\si@tempboxa=\hbox{\si@out@num{^{1}}}%
\si@tempdima\wd\si@tempboxa\relax
\ifnum\si@tab@expprecnt>\z@\relax
  \si@tab@expdim\si@tab@expprecnt\si@tempdima\relax
  \si@tab@sepcorr{exppre}{exp}%
\fi
\let\si@tab@sp\sp
\ifnum\si@tab@exppostcnt>\z@\relax
  \advance\si@tab@expdim\si@tab@exppostcnt\si@tempdima\relax
  \setbox\si@tempboxa=\hbox{%
    \ensuremath{^{\si@decimalsymbol}}}%
  \advance\si@tab@expdim\wd\si@tempboxa\relax
  \si@tab@sepcorr{exppost}{exp}%
\fi
```

Space is reserved for signs.

```
\setbox\si@tempboxa=\hbox{\ensuremath{-}}%
\ifsi@tab@mantsign
  \advance\si@tab@predim\wd\si@tempboxa\relax
\fi
\setbox\si@tempboxa=\hbox{\ensuremath{^{-}}}%
\ifsi@tab@expsign
  \advance\si@tab@expdim\wd\si@tempboxa\relax
\fi
```

Now, if there is space to be saved for an exponent under any circumstances, the space for the "$\times 10$" part is needed. This is done by adding both counters together, then using this result for the logic.

```
\si@tempcnta\si@tab@expprecnt\relax
\advance\si@tempcnta\si@tab@exppostcnt\relax
\ifnum\si@tempcnta>\z@\relax
  \setbox\si@tempboxa=\hbox{\ensuremath{%
    {}\si@expproduct{}\si@expbase}}%
  \advance\si@tab@expdim\wd\si@tempboxa\relax
\fi
```

Finally for the box dimensions, if the exponent is not aligned, the space reserved for it is added to the post box.

```
\ifsi@tabalignexp\else
  \advance\si@tab@postdim\si@tab@expdim\relax
\fi
```

With the boxes set up, the contents can be sorted out. A bit of shuffling may be needed, depending on the treatment of exponents.

```
\setbox\si@tab@prebox=\hbox to\si@tab@predim{\hss\hfill
  \expandafter\si@out@num\expandafter{\si@tab@out}}%
\ifx\@empty\si@num@out\@empty
  \setbox\si@tab@postbox=\hbox to\si@tab@postdim
    {\expandafter\si@out@num\expandafter{\si@num@out}\hfil}%
\else
  \ifsi@tabalignexp\else
    \protected@edef\si@num@out{\si@num@out\si@tab@expout}%
```

```
    \fi
    \setbox\si@tab@postbox=\hbox to\si@tab@postdim
      {\ensuremath{{\si@decimalsymbol}}\expandafter\si@out@num
        \expandafter{\si@num@out}\hfil}%
  \fi
  \ifx\@empty\si@tab@expout\@empty
    \ifsi@tabalignexp
      \setbox\si@tab@expbox=\hbox to\si@tab@expdim{\hfil}%
    \fi
  \else
    \ifsi@tabalignexp
      \setbox\si@tab@expbox=\hbox to\si@tab@expdim
        {\expandafter\si@out@num\expandafter{\si@tab@expout}%
           \hfil}%
    \fi
  \fi}
```

`\si@tab@sepcorr` `#1` : one of `mantpre`, `mantpost`, `exppre` or `exppost`
`#2` : one of `pre`, `post` or `exp`

A spacing correction is needed *if* the number of digits to be allowed for will lead to the introduction of a separator. A counter and dimension are needed for the testing.

```
\newcommand*{\si@tab@sepcorr}[2]{%
  \expandafter\si@tempcnta\expandafter\the
    \csname si@tab@#1cnt\endcsname\relax
```

Calculate how many groups of three there are, then allow for not separating four characters if `\ifsi@sepfour` is false.

```
  \divide\si@tempcnta\thr@@\relax
  \ifsi@sepfour\else
    \expandafter\ifnum\expandafter\the
      \csname si@tab@#1cnt\endcsname=4\relax
      \si@tempcnta\z@\relax
    \fi
  \fi
```

The width of the separators is measured, and the correct number of separator widths are added to the box dimension.

```
  \setbox\si@tempboxa=\hbox{%
    \ensuremath{\si@tab@sp{\si@digitsep}}}%
  \expandafter\advance\csname si@tab@#2dim\endcsname
    \si@tempcnta\wd\si@tempboxa}
```

21.13 Units

`\SI [#1]` : keyval options
`#2` : number

There are two types of user macros for the units system; those for defining new units, prefixes and powers, and those for using them. There are two macros for using units, `\SI` and `\si`, which work in very similar ways. argument to `\SI`

```
\si@newrobustcmd*{\SI}[2][]{%
  \@ifnextchar[%]
    {\si@SI[#1]{#2}}
    {\si@SI[#1]{#2}[]}}
```

`\si [#1]` : keyval options
`#2` : unit

`\si` is just an alias for `\SI` with no number; everything is handed off into an internal macro. The internal macro also handles the optional prefix

```
\si@newrobustcmd*{\si}[2][]{\si@SI[#1]{}[]{#2}}
```

`\newunit [#1]` : keyval options
`\renewunit` `#2` : unit
`\provideunit` `#3` : symbol

The `\newunit` and `\renewunit` macros create the new unit macros. To allow a mechanism for checking an existing definition, these macros simply carry out the appropriate tests, before handing off to the internal macro. `\@ifdefinable` is not used here as a customised error is desirable. Other than that, the code here gives very similar results to `\newcommand` and `\renewcommand`. Finally, `\provideunit` adds the unit definition only if it does not already exist.

```
\newcommand*{\newunit}[3][]{%
  \si@ifdefinable{#2}
    {\si@unt@defunit[#1]{#2}{#3}}
    {\si@log@err{Unit \string#2 already defined!}\@eha}}
\newcommand*{\renewunit}[3][]{%
  \si@ifdefinable{#2}
    {\si@log@err{Unit \string#2 undefined}\@ehc
     \si@unt@defunit[#1]{#2}{#3}}
    {\si@log@inf{Redefining unit \string#2}%
     \si@unt@defunit[#1]{#2}{#3}}}
\newcommand*{\provideunit}[3][]{%
  \si@ifdefinable{#2}
    {\si@unt@defunit[#1]{#2}{#3}}
    {}}
```

`\newprefix [#1]` : binary
`\renewprefix` `#2` : multiple
`\provideprefix` `#3` : powers-ten
`#4` : symbol

The multiples of units are defined here; very similar code is used to the `\newunit`, *etc.*, macros. The multiple prefixes cannot take an optional argument, and must represent some power. Hence the arguments required are different.

```
\newcommand*{\newprefix}[4][]{%
  \si@ifdefinable{#2}
    {\si@unt@defprefix[#1]{#2}{#3}{#4}}
    {\si@log@err{Prefix \string#2 already defined!}\@eha}}
\newcommand*{\renewprefix}[4][]{%
  \si@ifdefinable{#2}
    {\si@log@err{Prefix \string#2 undefined}\@ehc
     \si@unt@defprefix[#1]{#2}{#3}{#4}}
    {\si@log@inf{Redefining prefix \string#2}%
     \si@unt@defprefix[#1]{#2}{#3}{#4}}}
\newcommand*{\provideprefix}[4][]{%
  \si@ifdefinable{#2}
    {\si@unt@defprefix[#1]{#2}{#3}{#4}}
    {}}
```

\newpower [#1]: post
\renewpower #2 : number
\providepower #3 : power

Here power multiples for units are set up. As with units and multiples, a layered approach is used to keep things easy to maintain.

```
\newcommand*{\newpower}[3][]{%
  \si@ifdefinable{#2}
    {\si@unt@defpower[#1]{#2}{#3}}
    {\si@log@err{Power \string#2 already defined!}\@eha}}
\newcommand*{\renewpower}[3][]{%
  \si@ifdefinable{#2}
    {\si@log@err{Power \string#2 undefined}\@ehc
     \si@unt@defpower[#1]{#2}{#3}}
    {\si@log@inf{Redefining power \string#2}%
     \si@unt@defpower[#1]{#2}{#3}}}
\newcommand*{\providepower}[3][]{%
  \si@ifdefinable{#2}
    {\si@unt@defpower[#1]{#2}{#3}}
    {}}
```

\ifsi@unt@num A flag is needed to tell the processor whether there is a number, to get the correct spacing.

```
\newif\ifsi@unt@num
```

\si@unt@unitarg A storage macro for the argument of the unit macro.

```
\newcommand*{\si@unt@unitarg}{}
```

\si@SI [#1]: keyval options
\si@unt@SIopts #2 : unit
[#3]: preunit
#4 : unit

The internal processing starts with \si@SI, which processes the second optional argument to \SI (which is empty for \si). Everything is set up in a group, and processing begins by handling the options.

```
\newcommand*{\si@unt@SIopts}{}
\def\si@SI[#1]#2[#3]#4{%
  \begingroup
    \let\fg\SIfg
    \si@ifnotmtarg{#1}
      {\sisetup{#1}%
       \renewcommand*{\si@unt@SIopts}{#1}}%
```

The prefix unit is handled before any processing of the number; the flags are set to get spacing correct.

```
    \si@unt@numfalse
    \si@xspacefalse
    \si@ifnotmtarg{#3}
      {\si@log@debug{Prefix unit found}%
       \si@unt@printunit{#3}}%
```

The numerical argument may be empty, in which case no extra space should be produced.

```
    \si@ifnotmtarg{#4}
```

```
      {\renewcommand*{\si@unt@unitarg}{#4}}%
    \si@ifnotmtarg{#2}
      {\si@log@debug{Number found in \string\SI\space
         argument}%
       \ifsi@repeatunits\else
         \ifsi@trapambigerr
           \expandafter\expandafter\expandafter
             \si@num@ambigerrtrue
         \fi
       \fi
       \num{#2}%
       \si@unt@numtrue}%
```

If there is a unit, a check is needed in case the units need to have a power added.

```
    \si@ifnotmtarg{#4}
      {\si@ifmtarg{#2}
         {\si@unt@printunit{#4}}
         {\si@tempcnta\z@\relax
         \ifsi@addunitpower
           \si@unt@countx{#2}%
         \fi
          \ifnum\si@tempcnta>\z@\relax
            \advance\si@tempcnta\@ne\relax
            \edef\si@tempa{\noexpand\tothe{\si@tempcnta}}%
            \renewcommand*{\si@tempb}{#4}%
            \expandafter\expandafter\expandafter
            \si@unt@printunit\expandafter\expandafter
            \expandafter{%
              \expandafter\si@tempb\si@tempa}%
          \else
            \si@unt@printunit{#4}%
          \fi}}%
  \endgroup}
```

`\si@unt@countx` A short macro to count up any multiplication in numerical input.

```
\newcommand*{\si@unt@countx}[1]{%
  \si@tempcnta\z@\relax
  \expandafter\si@unt@cntx#1\@empty\@empty}
\def\si@unt@cntx#1#2\@empty{%
  \si@str@ifchrstr{#1}{\si@numprod}
    {\advance\si@tempcnta\@ne\relax}
    {}%
  \ifx\@empty#2\@empty\else
    \si@unt@cntx#2\@empty\@empty
  \fi}
```

`\si@unt@ifliteral` `#1` : unit

`\ifsi@unt@littest` The next stage of the processor is to determine whether or not the argument of the unit macro is processable. For literal arguments, this is not the case, and the argument is typeset "as is". On the other hand, any units, *etc*., declared by the package will work with the processor, and so need to be executed before typesetting the result.

```
\newif\ifsi@unt@littest
\newcommand*{\si@unt@ifliteral}[1]{%
```

```
  \begingroup
    \si@unt@littesttrue
```

The test relies on any non-processable test having some width; hopefully, this should be the case.

```
    \setbox\si@tempboxa=\hbox{\si@unt@out{#1}}%
    \ifdim\wd\si@tempboxa>\z@\relax
      \aftergroup\@firstoftwo
    \else
      \aftergroup\@secondoftwo
    \fi
  \endgroup}
```

`\ifsi@unt@litout` `#1 : unit`
`\si@unt@printunit`

The printing macro uses the above test to determine how to act. It then carries out the appropriate action: either typesetting or executing. A flag is also provided so that any macro units inside a partially-literal argument will work (this is also needed to emulate unitsdef).

```
\newif\ifsi@unt@litout
\newcommand*{\si@unt@printunit}[1]{%
  \si@unt@ifliteral{#1}
```

The unit includes one or more literal items; typeset using the unit typesetting macro.

```
    {\si@log@debug{Literal items found in unit
      argument:\MessageBreak outputting without further
      processing}%
     \si@unt@litouttrue
     \si@unt@addvaluesep
     \si@unt@out{#1}}
```

For processable output, the argument is executed; the macros are all designed for this.

```
    {\si@log@debug{Macro unit found:\MessageBreak
       processing to format output}%
     \si@unt@init
     \advance\si@unt@depthcnt\@ne\relax
     #1%
     \si@unt@final}}
```

`\si@unt@addvaluesep`
`\si@unt@addvalsep`
`\si@unt@litvalsep`
`\si@unt@stackvalsep`

To ensure no problems pop up with expansion, adding the value–unit space is handled by a macro.

```
\newcommand*{\si@unt@addvaluesep}{%
  \ifsi@unt@num
    \expandafter\si@unt@addvalsep
  \fi}
\newcommand*{\si@unt@addvalsep}{%
  \ifsi@unt@litout
    \expandafter\si@unt@litvalsep
  \else
    \expandafter\si@unt@stackvalsep
  \fi}
\newcommand*{\si@unt@stackvalsep}{%
  \protected@edef\si@unt@spstack{\si@valuesep}}
```

```
\newcommand*{\si@unt@litvalsep}{%
  \nobreak\ensuremath{\si@valuesep}\nobreak}
```

\si@unt@spstack
\si@unt@stacka
\si@unt@stackb
\si@unt@unitcnta
\si@unt@unitcntb
\si@unt@depthcnt
\ifsi@unt@first
\ifsi@unt@first
\si@unt@init

The initialisation macro sets up the various switches, and clears the storage areas for the formatted output. There are two stacks, as when typesetting as fractions, the two parts of the number have to be stored separately. The depth counter is used to tell when recursion ends in the processor. The "first" switch is needed as the depth counter will not be at one for items processed by \SI.

```
\newcommand*{\si@unt@spstack}{}
\newcommand*{\si@unt@stacka}{}
\newcommand*{\si@unt@stackb}{}
\newcount\si@unt@unitcnta
\newcount\si@unt@unitcntb
\newcount\si@unt@depthcnt
\newif\ifsi@unt@first
\si@unt@depthcnt\m@ne\relax
\newcommand*{\si@unt@init}{%
  \begingroup
    \si@unt@litoutfalse
    \si@unt@litprefixfalse
    \si@unt@firsttrue
    \si@unt@perfalse
    \si@unt@perseenfalse
    \si@unt@prepowerfalse
    \si@unt@depthcnt\z@\relax
    \si@unt@powerdim\z@\relax
    \si@unt@unitcnta\z@\relax
    \si@unt@unitcntb\z@\relax
    \si@unt@prefixcnt\z@\relax
    \renewcommand*{\si@unt@spstack}{}%
    \renewcommand*{\si@unt@stacka}{}%
    \renewcommand*{\si@unt@stackb}{}%
    \renewcommand*{\si@unt@holdstacka}{}%
    \renewcommand*{\si@unt@holdstackb}{}%
    \renewcommand*{\si@unt@lastadda}{space}%
    \renewcommand*{\si@unt@lastaddb}{space}}
```

\si@unt@final

The finalisation macro finishes off the output and resets the flags.

```
\newcommand*{\si@unt@final}{%
    \si@unt@third
    \si@unt@stackout
  \endgroup
  \ifsi@xspace
    \expandafter\expandafter\expandafter\xspace
  \fi}
```

\si@unt@defunit [#1]: keyval options
#2 : unit
#3 : symbol

The internal macro for defining a unit does not check for redefinition; that is done by the user macros.

```
\newcommand*{\si@unt@defunit}[3][]{%
  \si@log@debug{Declaring unit \string#2 with \MessageBreak
    meaning \string#3}%
```

The optional argument needs to be saved. The macro name is reversed so that life is easier with the expansions here.

```
\si@ifnotmtarg{#1}
  {\expandafter\@namedef\expandafter{%
    \expandafter\@gobble\string#2@opt@unt@si}{#1}}%
```

The unit macro itself is now defined. The definition simply selects the correct path for the rest of the processing to go down.

```
\protected\def#2{%
  \ifsi@allowoptarg
    \expandafter\si@unt@withopt
  \else
    \expandafter\si@unt@noopt
  \fi
  {#2}{#3}}}
```

`\si@unt@withopt` `\si@unt@noopt`
`#1` : unit
`#2` : symbol

To allow the correct expansion, the potential optional argument to a unit macro has to come last. So `\@ifnextchar` is needed to detect it and pass data through. To keep variation down, when the argument is not allowed, the empty `[]` is supplied.

```
\newcommand*{\si@unt@withopt}[2]{%
  \@ifnextchar[%]
    {\si@unt@opt{#1}{#2}}
    {\si@unt@opt{#1}{#2}[]}}
\newcommand*{\si@unt@noopt}[2]{\si@unt@opt{#1}{#2}[]}
```

`\si@unt@opt`
`#1` : unit
`#2` : symbol
`[#3]` : number

The optional argument to the unit macro (if present) is converted to a normal one for ease. The correct route for processing is then picked.

```
\def\si@unt@opt#1#2[#3]{%
  \ifsi@unt@littest
    \expandafter\si@gobblethree
  \else
```

For literal output, the second argument is all that is needed.

```
    \ifsi@unt@litout
      \expandafter\expandafter\expandafter\@gobbletwo
    \else
      \expandafter\expandafter\expandafter\si@unt@unit
    \fi
  \fi
  {#3}{#1}{#2}}
```

`\si@gobblethree` LaTeX does not have a `\@gobblethree` macro, but one is needed.

```
\long\def\si@gobblethree #1#2#3{}
```

`\si@unt@defprefix` `[#1]` : binary
`\ifsi@unt@litprefix` `#2` : multiple
`#3` : powers-ten

`#4` : symbol

As with units, multiples are defined by an internal macro.

```
\newif\ifsi@unt@litprefix
\si@unt@litprefixtrue
\newcommand*{\si@unt@defprefix}[4][]{%
  \si@log@debug{Declaring multiple \string#1 with\MessageBreak
    meaning \string#4}%
```

The optional argument is saved, using `\def` as no check is made on an existing definition of the storage macro.

```
  \expandafter\expandafter\expandafter\def\expandafter
    \csname\expandafter\@gobble\string#2@opt@si\endcsname{#1}%
  \protected\def#2{%
    \ifsi@unt@littest
      \expandafter\si@gobblethree
    \else
      \ifsi@unt@litout
        \expandafter\expandafter\expandafter\@gobbletwo
      \else
        \ifsi@unt@litprefix
          \expandafter\expandafter\expandafter\expandafter
            \expandafter\expandafter\expandafter\@gobbletwo
        \else
          \expandafter\expandafter\expandafter\expandafter
            \expandafter\expandafter\expandafter\si@unt@prefix
        \fi
      \fi
    \fi
    {#2}{#3}{#4}}}
```

`\si@unt@defpower [#1]: post`

`#2` : number

`#3` : power

The definition of powers is complicated by the need to handle both those given before units (such as `\cubic`) and those given after (*e.g.* `\cubed`). This means that an optional argument is needed.

```
\newcommand*{\si@unt@defpower}[3][]{%
  \si@log@debug{Declaring power \string#2 with\MessageBreak
    meaning \string#3}%
```

Once again the optional argument is saved.

```
  \expandafter\expandafter\expandafter\def\expandafter
    \csname\expandafter\@gobble\string#2@opt@si\endcsname{#1}%
  \protected\def#2{%
    \ifsi@unt@littest
      \expandafter\@gobbletwo
    \else
```

The literal output here does not need to gobble anything.

```
      \ifsi@unt@litout
        \expandafter\expandafter\expandafter\si@unt@litpower
      \else
        \expandafter\expandafter\expandafter\si@unt@power
      \fi
```

```
    \fi
    {#2}{#3}}}
```

`\si@unt@unithook` `#1` : number
`\si@unt@unit` `#2` : unit
`#3` : symbol

The macro for units is actually a processor, rather than typesetting anything, which is handled elsewhere. The first argument to the macro is optional, but does not have square brackets to keep things simple with gobbling.

```
\newcommand*{\si@unt@unithook}{}
\newcommand*{\si@unt@unit}[3]{%
```

When the count is minus one at the start of the processor, then the unit is begin used on its own: initialisation occurs.

```
  \ifnum\si@unt@depthcnt=\m@ne\relax
    \expandafter\si@unt@init
  \fi
  \advance\si@unt@depthcnt\@ne\relax
  \si@log@debug{Unit processing: level \the\si@unt@depthcnt,
    \MessageBreak unit \string#2}%
  \si@unt@firstorsecond{#1}{#2}%
```

The core of the `\si@unt@unit` macro is testing if the symbol for the unit is a literal value or another macro. Depending on the result, the symbol is either used as a literal or executed.

```
  \si@unt@ifliteral{#3}
    {\si@unt@addtostack{unit}{#3}%
     \ifsi@unt@prepower
       \expandafter\si@unt@stkpower
     \fi}
    {#3}%
```

The counter is now stepped down, before checking if this is the end of a compound unit.

```
  \advance\si@unt@depthcnt\m@ne\relax
  \ifnum\si@unt@depthcnt=\z@\relax
    \expandafter\si@unt@final
  \fi}
```

`\si@unt@firstorsecond` `#1` : number
`#2` : unit-macro

At this stage, the flag will be set for the first item to be processed whichever route the unit has been called by.

```
\newcommand*{\si@unt@firstorsecond}[2]{%
  \ifsi@unt@first
    \expandafter\si@unt@first
  \else
    \expandafter\si@unt@second
  \fi
  {#1}{#2}}%
```

`\si@unt@first` `#1` : number
`#2` : unit-macro

For the first unit in the input, some extra tasks are needed. First, the optional argument for the unit macro needs to be tested.

```
\newcommand*{\si@unt@first}[2]{%
  \si@ifnotmtarg{#1}
    {\num{#1}%
     \si@unt@numtrue}%
  \si@unt@unithook
```

To avoid filling up the macro list with useless values, the ε-TEX primitive `\ifcsname` is employed here (it also avoids complex expansion issues). If some options exist, they are set.

```
  \ifcsname\expandafter\@gobble\string#2@opt@unt@si\endcsname
    \expandafter\si@unt@setopts
  \else
    \expandafter\@gobble
  \fi
  {#2}%
  \si@unt@addvaluesep
  \si@unt@firstfalse}
```

`\si@unt@setopts` `#1` : unit

`\si@unt@setSIopts` A rather long set of `\expandafter` commands to get the options to set safely.

```
\newcommand*{\si@unt@setopts}[1]{%
  \expandafter\expandafter\expandafter\expandafter\expandafter
    \expandafter\expandafter\si@temptoks\expandafter
    \expandafter\expandafter\expandafter\expandafter
    \expandafter\expandafter{\expandafter%
      \csname\expandafter\@gobble\string#1@opt@unt@si%
      \endcsname}%
  \expandafter\sisetup\expandafter{\the\si@temptoks}%
  \si@log@debug{Applying options `\the\si@temptoks'
    for\MessageBreak unit \string#1}%
```

The user options are reloaded, if defined, to ensure that they still work as expected.

```
  \@ifundefined{si@unt@SIopts}{}
    {\ifx\@empty\si@unt@SIopts\@empty\else
       \expandafter\expandafter\si@unt@setSIopts
     \fi}}
\newcommand*{\si@unt@setSIopts}{%
  \expandafter\si@temptoks\expandafter{\si@unt@SIopts}%
  \expandafter\sisetup\expandafter{\the\si@temptoks}}
```

`\si@unt@second` `#1` : number

`\si@unt@third` `#2` : unit

For everything apart from the first item to be processed, spacing may need to be added to separated different units. The macro is divided into two, so that everything except the space can be added in finalisation.

```
\newcommand*{\si@unt@second}[2]{%
  \si@ifnotmtarg{#1}
    {\si@log@warn{Optional argument to unit macro\MessageBreak
       allowed only for outer unit}}%
  \si@unt@third
  \si@unt@addtostack{space}{\ensuremath{\si@unitsep}}}
```

```
\newcommand*{\si@unt@third}{%
  \ifsi@unt@prepower\else
    \expandafter\si@unt@stkpower
  \fi
```

A check is made to avoid adding -1 to prefixes. If `frac` is active, then the `b` stack will be in use, otherwise it will be `a`.

```
  \renewcommand*{\si@tempa}{prefix}%
  \expandafter\ifx
    \csname si@unt@lastadd\si@unt@checkstack\endcsname\si@tempa
  \else
    \expandafter\si@unt@spacecheck
  \fi
  \ifsi@unt@per
    \expandafter\si@unt@perseentrue
  \fi}
```

`\si@unt@spacecheck` A check to prevent adding -1 at the very beginning of the unit, where there is a space on the stack.

```
\newcommand*{\si@unt@spacecheck}{%
  \renewcommand*{\si@tempa}{space}%
  \expandafter\ifx
    \csname si@unt@lastadd\si@unt@checkstack\endcsname\si@tempa
  \else
    \expandafter\si@unt@reciptest
  \fi}
```

`\si@unt@prefix` `#1` : multiple
`#2` : power-ten
`#3` : symbol

Actual output of the prefixes.

```
\newcommand*{\si@unt@prefix}[3]{%
  \si@unt@firstorsecond{}{#1}%
  \ifsi@prefixsymbolic
    \expandafter\si@unt@addprefix
  \else
    \expandafter\si@unt@countprefix
  \fi
  {#1}{#2}{#3}}
```

`\si@unt@addprefix` `#1` : gobbled
`#2` : gobbled
`#3` : symbol

To add the prefix, a little translation is needed.

```
\newcommand*{\si@unt@addprefix}[3]{%
  \si@unt@addtostack{prefix}{#3}}
```

`\si@unt@prefixcnt` A storage area is created.

```
\newcount\si@unt@prefixcnt
```

`\si@unt@countprefix` `#1` : multiple
`\si@unt@invprefix` `#2` : powers-ten
`#3` : gobbled

To count the prefix numeral, the symbol is thrown away. First, a check is made for binary units.

```
\newcommand*{\si@unt@countprefix}[3]{%
  \renewcommand*{\si@tempa}{binary}%
  \expandafter\expandafter\expandafter\ifx\expandafter
    \csname\expandafter\@gobble\string#1@opt@si\endcsname
    \si@tempa
    \expandafter\sisetup
  \else
    \expandafter\@gobble
  \fi
  {prefixbase=two}%
  \si@tempcnta#2\relax
  \ifsi@unt@per
    \expandafter\si@unt@invprefix
  \fi
  \advance\si@unt@prefixcnt\si@tempcnta\relax}
\newcommand*{\si@unt@invprefix}{%
  \si@tempcntb\si@tempcnta\relax
  \si@tempcnta -\si@tempcntb\relax}
```

`\si@unt@litpower` `#1` : gobbled
`#2` : number
For literal power output, the number can't simply be dumped, so a macro is needed.

```
\newcommand*{\si@unt@litpower}[2]{\textsuperscript{#2}}
```

`\ifsi@unt@prepower` If a power is seen before a unit, tracking is needed.

```
\newif\ifsi@unt@prepower
```

`\si@unt@power` `#1` : power
`#2` : number
The handling of powers starts by ensuring that "pre" powers follow `\per` cleanly. Then a check is needed for inversion.

```
\newcommand*{\si@unt@power}[2]{%
  \renewcommand*{\si@tempa}{post}%
  \expandafter\expandafter\expandafter\ifx\expandafter
    \csname\expandafter\@gobble\string#1@opt@si\endcsname
    \si@tempa
    \expandafter\@gobbletwo
  \else
    \expandafter\si@unt@firstorsecond
  \fi
  {}{\power}%
  \si@unt@powerdim #2 pt\relax
  \ifsi@frac\else
    \ifsi@unt@per
      \expandafter\expandafter\expandafter\si@unt@invpower
    \fi
  \fi
  \renewcommand*{\si@tempa}{post}%
  \si@unt@prepowertrue
  \expandafter\expandafter\expandafter\ifx\expandafter
```

```
    \csname\expandafter\@gobble\string#1@opt@si\endcsname
    \si@tempa
    \expandafter\si@unt@stackpower
  \else
    \si@log@debug{Power \strip@pt\si@unt@powerdim\space saved
      to be added after\MessageBreak next unit}%
  \fi}
```

`\si@unt@powerdim` To do sign-inversion on the power, a dimension is used (this allows non-integer values to be handled).

```
\newdimen\si@unt@powerdim
```

`\si@unt@stackpower` `\si@unt@stkpower` `\si@unt@stkpwr` Adding powers to the stack should also clear the power list. If the number is already zero, then of course nothing happens.

```
\newcommand*{\si@unt@stackpower}{%
  \si@unt@prepowerfalse
```

A trap is used for -1 added to the denominator of a fraction.

```
  \si@unt@stkpower
  \ifsi@stickyper\else
    \si@unt@perfalse
    \si@unt@perseenfalse
  \fi}
```

The `\si@unt@stkpower` macro needs to be called from a few places, so is spun out from the above.

```
\newcommand*{\si@unt@stkpower}{%
  \ifdim\si@unt@powerdim=\m@ne pt\relax
    \ifsi@frac\else
      \expandafter\expandafter\expandafter\si@unt@stkpwr
    \fi
  \else
    \expandafter\si@unt@stkpwr
  \fi}
```

Finally, the actual adding (set up to avoid problems with the `\if` above).

```
\newcommand*{\si@unt@stkpwr}{%
  \ifdim\si@unt@powerdim=\z@\relax\else
    \renewcommand*{\si@tempa}{unit}%
    \expandafter\ifx
      \csname si@unt@lastadd\si@unt@checkstack\endcsname
      \si@tempa
      \si@log@debug{Adding power
        \strip@pt\si@unt@powerdim\space to output stack}%
      \si@unt@addtostack{power}{^{\num[fixdp=false]{%
        \strip@pt\si@unt@powerdim}}}%
    \fi
  \fi
  \si@unt@powerdim\z@\relax}
```

`\si@unt@invpower` A macro to change the sign of the current power.

```
\newcommand*{\si@unt@invpower}{%
  \si@tempdima\si@unt@powerdim\relax
  \si@unt@powerdim -\si@tempdima\relax
```

The power might end up as "1", which is not wanted. So it is chucked away.

```
\ifdim\si@unt@powerdim=\p@\relax
  \si@unt@powerdim\z@\relax
\fi}
```

\ifsi@unt@per
\ifsi@unt@perseen
\per
\si@per
\si@unt@per

The \per macro sets the correct flags; almost everything else is done elsewhere. There is always the case of two \per instructions; so the flag is flipped rather than set arbitrarily. The second flag is needed so that \per can give powers of -1 properly.

```
\newif\ifsi@unt@per
\newif\ifsi@unt@perseen
\si@newrobustcmd*{\si@per}{%
  \ifsi@unt@littest\else
    \ifsi@unt@litout
      \expandafter\expandafter\expandafter /%
    \else
      \ifsi@unt@litprefix
        \expandafter\expandafter\expandafter\expandafter
          \expandafter\expandafter\expandafter /%
      \else
        \expandafter\expandafter\expandafter\expandafter
          \expandafter\expandafter\expandafter\si@unt@per
      \fi
    \fi
  \fi}
\newcommand*{\si@unt@per}{%
  \si@unt@firstorsecond{}{\per}%
  \ifsi@unt@per
    \ifsi@stickyper\else
      \expandafter\expandafter\expandafter\si@unt@perfalse
    \fi
  \else
    \expandafter\si@unt@pertrue
  \fi}
\let\per\si@per
```

\si@unt@reciptest
\si@unt@recip

A test is needed for adding -1 when needed. The second macro is fired only if the power should be reciprocal.

```
\newcommand*{\si@unt@reciptest}{%
  \ifsi@unt@per
    \ifsi@unt@perseen
      \expandafter\expandafter\expandafter\si@unt@recip
    \fi
  \fi}
\newcommand*{\si@unt@recip}{%
  \si@unt@powerdim\m@ne pt\relax
  \si@unt@stackpower}
```

\si@unt@lastadda
\si@unt@lastaddb

Items cannot be added directly to the stacks (except the spacing stack, a) as the fractional handling may need to add the item to either storage area. First, a track is kept of what has been added at each stage.

```
\newcommand*{\si@unt@lastadda}{}
\newcommand*{\si@unt@lastaddb}{}
```

`\si@unt@addtostack` `#1` : type
`#2` : token

By indicating the type of data to be added to the stack, problems can be avoided.

```
\newcommand*{\si@unt@addtostack}[2]{%
  \renewcommand*{\si@tempa}{#1}%
```

Two consecutive items cannot be of the same type; there must be spaces between units, units between prefixes, *etc*.

```
  \expandafter\ifx
    \csname si@unt@lastadd\si@unt@checkstack\endcsname\si@tempa
    \expandafter\@gobbletwo
  \else
    \expandafter\si@unt@preplussp
  \fi
  {#1}{#2}}
```

`\si@unt@preplussp` `#1` : type
`#2` : stack
`#3` : token
`#4` : gobbled

The space added after a prefix needs to be ignored.

```
\newcommand*{\si@unt@preplussp}[2]{%
  \renewcommand*{\si@tempa}{prefix+space}%
  \edef\si@tempb{%
    \csname si@unt@lastadd\si@unt@checkstack\endcsname+#1}%
  \ifx\si@tempa\si@tempb
    \expandafter\@gobbletwo
  \else
    \expandafter\si@unt@stack
  \fi
  {#1}{#2}}
```

`\si@unt@stack` `#1` : type
`#2` : token

The macro for actually doing the stacking up.

```
\newcommand*{\si@unt@stack}[2]{%
  \expandafter\renewcommand\expandafter*\expandafter{%
    \csname si@unt@lastadd\si@unt@checkstack\endcsname}{#1}%
```

A count is kept of the number of *units* added to each stack.

```
  \renewcommand*{\si@tempa}{#1}%
  \renewcommand*{\si@tempb}{unit}%
  \ifx\si@tempa\si@tempb
    \expandafter\si@unt@inccnt
  \fi
```

If a space is added, it is actually held until the next add.

```
  \renewcommand*{\si@tempb}{space}%
  \ifx\si@tempa\si@tempb
    \expandafter\si@unt@holdspace
  \else
    \expandafter\si@unt@addstack
  \fi
  {#2}}
```

\si@unt@inccnt The appropriate counter is added to.

```
\newcommand*{\si@unt@inccnt}{%
  \expandafter\advance
    \csname si@unt@unitcnt\si@unt@checkstack\endcsname
    \@ne\relax}
```

\si@unt@holdstacka The stacked material needs somewhere to live.

\si@unt@holdstackb

```
\newcommand*{\si@unt@holdstacka}{}
\newcommand*{\si@unt@holdstackb}{}
```

\si@unt@holdspace `#1` : token

\si@unt@addstack Depending on the nature of the addition, it is either held or added to the stack. For the "b" space stack, a check is made to ensure that a space cannot be added before the first item.

```
\newcommand*{\si@unt@holdspace}[1]{%
  \renewcommand*{\si@tempa}{b}%
  \edef\si@tempb{\si@unt@checkstack}%
  \ifx\si@tempa\si@tempb
    \ifx\@empty\si@unt@stackb\@empty
    \else
      \expandafter\protected@edef
      \csname si@unt@holdstack\si@unt@checkstack\endcsname{#1}%
    \fi
  \else
    \expandafter\protected@edef
    \csname si@unt@holdstack\si@unt@checkstack\endcsname{#1}%
  \fi}
\newcommand*{\si@unt@addstack}[1]{%
  \expandafter\protected@edef
    \csname si@unt@stack\si@unt@checkstack\endcsname
    {\csname si@unt@stack\si@unt@checkstack\endcsname
     \csname si@unt@holdstack\si@unt@checkstack\endcsname#1}%
  \expandafter\renewcommand\expandafter*\expandafter{%
    \csname si@unt@holdstack\si@unt@checkstack\endcsname}{}}
```

\si@unt@stackout The stack contents are actually typeset here. First the spacing between units and values is added.

```
\newcommand*{\si@unt@stackout}{%
  \si@unt@litouttrue
  \ifsi@frac
    \expandafter\si@unt@fracout
  \else
    \expandafter\si@unt@normout
  \fi}
```

\si@unt@checkstack Which stack is in use needs to be tested.

```
\newcommand*{\si@unt@checkstack}{%
  \ifsi@frac
    \ifsi@unt@per
      \expandafter\expandafter\expandafter b%
    \else
      \expandafter\expandafter\expandafter a%
    \fi
```

```
  \else
    \expandafter a%
  \fi}
```

`\si@unt@spaceout` The space before a unit might not be needed, so it crops up a few times in the output routine.

```
\newcommand*{\si@unt@spaceout}{%
   \ensuremath{\si@unt@spstack}}
```

`\si@unt@prefixout` If the prefix counter is not zero, then there is something to typeset.

```
\newcommand*{\si@unt@prefixout}{%
  \ifnum\si@unt@prefixcnt=\z@\relax\else
    \ifsi@unt@num
      \si@out{\ensuremath{{}\si@prefixproduct{}}}%
    \fi
    \si@unt@stackvalsep
    \let\si@expbase\si@prefixbase
    \num[fixdp=false]{e\the\si@unt@prefixcnt}%
  \fi}
```

`\si@unt@normout` The normal output mode is set up here; nothing much needs to be done as there is no need for complex checks.

```
\newcommand*{\si@unt@normout}{%
  \si@unt@prefixout
  \si@unt@spaceout
  \expandafter\si@unt@out\expandafter{\si@unt@stacka}}
```

`\si@unt@fracout` For fractions, some checks are needed.

```
\newcommand*{\si@unt@fracout}{%
  \si@unt@notambig
  \ifx\@empty\si@unt@stacka\@empty
    \ifx\@empty\si@unt@stackb\@empty
      \ifsi@unt@litout\else
        \si@log@err{Empty fractional unit}{The unit
          argument\MessageBreak given does not contain any
          symbols}%
      \fi
    \else
```

With an empty numerator, no space is added

```
      \ifsi@slash
        \si@unt@prefixout
        \si@frac{}{\si@unt@stackb}%
      \else
        \si@unt@prefixout
        \si@unt@spaceout
        \si@frac{1}{\si@unt@stackb}%
      \fi
    \fi
  \else
```

If the denominator is empty, then the usual output system can be used.

```
    \ifx\@empty\si@unt@stackb\@empty
      \si@unt@normout
```

```
    \else
      \si@unt@prefixout
      \si@unt@spaceout
      \si@frac{\si@unt@stacka}{\si@unt@stackb}%
    \fi
  \fi}
```

`\si@unt@notambig` `\si@unt@notabg` A trap is set for adding brackets to units using a slash, when more than one unit is in the denominator.

```
\newcommand*{\si@unt@notambig}{%
  \ifnum\si@unt@unitcntb>\@ne\relax
    \ifsi@slash
        \ifsi@trapambigfrac
          \expandafter\expandafter\expandafter\expandafter
            \expandafter\expandafter\expandafter\si@unt@notabg
        \fi
    \fi
  \fi}
\newcommand*{\si@unt@notabg}{%
  \protected@edef\si@unt@stackb{\si@openfrac\si@unt@stackb
    \si@closefrac}}
```

`\si@unt@out` `#1` : unit

The final part of the units system is the output routine. This has to cope with units not only as macros but also as direct input (SIstyle-type input). Non-Latin characters are also handled cleanly. The `\scantokens` system deals with everything except full stops; these are left out so that a single level system can be used *via* a token register.

```
\begingroup
  \catcode`\~=\active
  \catcode`\.=\active
  \gdef\si@unt@out#1{%
    \si@temptoks{#1}%
    \si@unt@fullstop
    \def.{\ensuremath{\si@unitsep}}%
    \def~{\ensuremath{\si@unitspace}}%
    \expandafter\protected@edef\expandafter\si@tempa
      \expandafter{\the\si@temptoks}%
    \begingroup
      \si@unt@nonlatin
      \makeatletter
      \endlinechar\m@ne
      \expandafter\si@out\expandafter{%
        \expandafter\scantokens\expandafter{\si@tempa}}%
    \endgroup}
\endgroup
```

`\si@unt@fullstop` `\si@unt@stp` Two macros modified from kvsetkeys to deal with a single level of active full stops only.

```
\begingroup
  \catcode`\.=\active
  \catcode`\&=12\relax
  \begingroup
```

```
    \lccode`\.=`\.\relax
    \lccode`\&=`\.\relax
  \lowercase{\endgroup
    \gdef\si@unt@fullstop{%
      \si@temptoks\expandafter{\expandafter}\expandafter
      \si@unt@stp\the\si@temptoks&\@nil}
  \gdef\si@unt@stp#1&#2\@nil{%
    \edef\si@tempa{\the\si@temptoks}%
    \ifx\si@tempa\@empty
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi
      {\si@temptoks{#1}}
      {\si@temptoks\expandafter{\the\si@temptoks.#1}}%
    \si@ifmtarg{#2}
      {}
      {\si@unt@stp#2\@nil}}}
\endgroup
```

`\si@unt@nonlatin` To handle non-Latin symbols in the input, a single macro is provided. If XeTeX is in use, this can be detected immediately.

```
\newcommand*{\si@unt@nonlatin}{}
\ifdefined\XeTeXrevision
  \renewcommand*{\si@unt@nonlatin}{%
    \catcode176=\active
    \catcode181=\active
    \catcode197=\active
    \si@unt@sym{176}{\si@sym@degree}%
    \si@unt@sym{181}{\si@sym@mu}%
    \si@unt@sym{197}{\si@sym@ringA}}%
\fi
```

If inputenc has been loaded, then a check is made that the encoding is correct. If all is well, the non-Latin symbols are handled. The degree symbol is character 176, the micro symbol is character 181 and ring capital A is character 197 in latin1.

```
\AtBeginDocument{
  \@ifpackageloaded{inputenc}
    {\@for\si@tempa:=latin1,latin5,latin9\do{
       \ifx\inputencodingname\si@tempa
         \renewcommand*{\si@unt@nonlatin}{%
           \catcode176=\active
           \catcode181=\active
           \catcode197=\active
           \si@unt@sym{176}{\si@sym@degree}%
           \si@unt@sym{181}{\si@sym@mu}%
           \si@unt@sym{197}{\si@sym@ringA}}%
       \fi}}
    {}}
```

`\si@unt@sym` `#1` : charcode

A macro for declaring symbols: a copy of `\DeclareInputMath` from inputenc.

```
\newcommand*{\si@unt@sym}[1]{%
  \bgroup
```

```
    \uccode`\~#1%
    \uppercase{%
      \egroup
      \def~}}
```

\kilogram
\metre
\meter
\mole
\kelvin
\candela
\second
\ampere

With the system set up, the basic unit macros are implemented. The only units defined whatever options are given are the base SI units.

```
\newunit{\kilogram}{kg}
\newunit{\metre}{m}
\newunit{\meter}{\metre}
\newunit{\mole}{mol}
\newunit{\second}{s}
\newunit{\ampere}{A}
\newunit{\kelvin}{K}
\newunit{\candela}{cd}
```

\Square
\ssquare
\squared
\cubic
\cubed

Unlike multiples (which can be skipped if needed), the basic powers are also always defined.

```
\newpower{\Square}{2}
\newpower{\ssquare}{2}
\newpower[post]{\squared}{2}
\newpower{\cubic}{3}
\newpower[post]{\cubed}{3}
```

\tothe
\raiseto

The user macros for arbitraty powers are simple calling macros to a common internal macro.

```
\newcommand*{\tothe}{\si@tothe{\tothe}}
\newcommand*{\raiseto}{\si@tothe{\raiseto}}
```

\si@tothe
\tothe@opt@si
\raiseto@opt@si

`#1` : either `\tothe` or `\raiseto`
`#2` : number
A macro for arbitrary powers, which comes after the unit and so needs to be marked as such. Two fake option-storage macros are created so that everything works correctly.

```
\newcommand*{\si@tothe}[2]{%
  \ifsi@unt@littest
    \expandafter\@gobbletwo
  \else
    \ifsi@unt@litout
      \expandafter\expandafter\expandafter\si@unt@litpower
    \else
      \expandafter\expandafter\expandafter\si@unt@power
    \fi
  \fi
  {#1}{#2}}
\newcommand*{\tothe@opt@si}{post}
\newcommand*{\raiseto@opt@si}{}
```

21.14 Locales

\si@loc@load
\si@loc@sisetup

`#1` : locale

When loading a locale, the setup is saved rather than applied. Anything other than simple settings should be inside `\addtolocale`, which is already defined.

```
\newcommand*{\si@loc@load}[1]{%
  \let\si@loc@sisetup\sisetup
  \renewcommand*{\sisetup}[1]{%
    \expandafter\gdef\csname si@loc@#1\endcsname{##1}}%
  \si@loadfile{#1}%
  \let\sisetup\si@loc@sisetup}
```

`\si@loc@set` `#1` : locale

Setting the locale transfers the settings to `\sisetup`, and runs any extra macros.

```
\newcommand*{\si@loc@set}[1]{%
  \ifcsname si@loc@#1\endcsname
    \si@log@inf{Setting locale to `#1'}%
    \expandafter\expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\si@temptoks
      \expandafter\expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter{%
        \expandafter\csname si@loc@#1\endcsname}%
    \expandafter\sisetup\expandafter{\the\si@temptoks}%
    \ifcsname si@loc@#1@extra\endcsname
      \csname si@loc@#1@extra\endcsname
    \fi
  \else
    \ifcsname si@loc@#1@extra\endcsname
      \si@log@inf{Setting locale to `#1'}%
      \csname si@loc@#1@extra\endcsname
    \else
      \si@log@warn{Unknown locale `#1'}%
    \fi
  \fi}
```

`\si@loc@ltol` `#1` : list of locales and languages

The necessary loading for language modules occurs.

```
\newcommand*{\si@loc@ltol}[1]{%
  \def\si@tempa##1:##2\@nil{\si@loc@load{##1}}
  \@for\si@tempb:=#1\do{%
    \expandafter\si@tempa\si@tempb:\@nil}
  \AtBeginDocument{
    \@ifpackageloaded{babel}
      {\def\si@tempa##1:##2:##3\@nil{%
        \expandafter\addto\expandafter{%
          \csname extras##2\endcsname}%
          {\si@loc@set{##1}}}%
       \@for\si@tempb:=#1\do{%
         \expandafter\si@tempa\si@tempb::\@nil}%
       \expandafter\selectlanguage\expandafter{\languagename}}
      {\si@log@warn{babel not loaded \MessageBreak
        loctolang option ignored}}}}
\AtBeginDocument{
  \@ifpackageloaded{babel}
    {\renewcommand*{\si@loc@ltol}[1]{%
       \def\si@tempa##1:##2\@nil{\si@loc@load{##1}}%
```

```
      \@for\si@tempb:=#1\do{%
        \expandafter\si@tempa\si@tempb:\@nil}%
      \def\si@tempa##1:##2:##3\@nil{%
        \expandafter\addto\expandafter{%
          \csname extras##2\endcsname}%
          {\si@loc@set{##1}}}%
      \@for\si@tempb:=#1\do{%
        \expandafter\si@tempa\si@tempb::\@nil}}}
   {\renewcommand*{\si@loc@ltol}[1]{%
      \si@log@warn{babel not loaded \MessageBreak
        loctolang option ignored}}}}
```

`\addtolocale` `#1` : locale
`#2` : commands
Arbitrary macros may need to be added to the locale.

```
\newcommand*{\addtolocale}[2]{%
  \si@addtocsname{si@loc@#1@extra}{#2}}
```

21.15 Output routine

`\si@out` `#1` : text
With all of the setup done, the text can finally be typeset. This is done inside a `\text` block, which takes care of `\ensuremath`, *etc*. First of all, the various catcode settings needed to get maths-in-text mode are made. `\makeatletter` is needed so that `\scantokens` still allows internal macros to work.

```
\begingroup
  \catcode`\^=\active
  \catcode`\-=\active
  \gdef\si@out#1{%
    \begingroup
      \catcode`\^=\active
      \makeatletter
      \endlinechar\m@ne
```

The various font families can now be set up. First a check is made in case there are nested calls to `\si@out@text`.

```
      \ifsi@fam@set\else
        \expandafter\si@fam@set
      \fi
      \text{%
        \si@colourcmd{\si@colour}%
        \si@fam@italic\si@fam@bold\si@fam@text
```

The correct mode is selected, and the input is handed off for typesetting.

```
        \ifsi@textmode
          \expandafter\si@out@text
        \else
          \expandafter\si@out@maths
        \fi
        {\scantokens{#1}}}%
    \endgroup
    \check@mathfonts}
```

`\si@out@text` `\si@out@maths` `#1` : text

Output occurs with the correct changes to superscript behaviour.

```
\gdef\si@out@text#1{%
  \let^\si@out@sp
  \let\textsuperscript\si@out@sp
  \catcode`\-=\active\relax
  \let-\si@out@minus
  #1}
\gdef\si@out@maths#1{%
  \let^\sp
  \let\textsuperscript\sp
  $\si@fam@maths{#1}$}
\endgroup
```

`\si@out@sp` `#1` : text

`\textsuperscript` gives slightly different alignment of numbers to using `^` in text mode. To avoid this, a slightly different definition is used. Elsewhere `\textsuperscript` is used, as the code above sorts out the text/maths mode issues.

```
\newcommand*{\si@out@sp}[1]{\ensuremath{^{\text{#1}}}}
```

`\si@out@minus` An active minus sign is needed.

```
\newcommand*{\si@out@minus}{\ensuremath{-}}
```

`\ifsi@out@num` A flag is needed to control output settings. This will be false unless inside `\si@out@num`.

```
\newif\ifsi@out@num
```

`\si@out@num` `#1` : number

For numerical output, the default fonts are controlled slightly differently to text output.

```
\newcommand*{\si@out@num}[1]{%
  \begingroup
    \si@out@numtrue
    \si@out{#1}%
  \endgroup}
```

21.16 Finalisation

`\si@extension` `\si@fileprefix` To keep the code easy to maintain, the reusable filename components are macros rather than literals.

```
\newcommand*{\si@extension}{cfg}
\newcommand*{\si@fileprefix}{si-}
```

`\si@ifl@aded` `#1` : package

`\si@ifloaded` A bit of borrowing from the LaTeX kernel. A copy of `\@ifl@aded` is needed as things aren't always done in the preamble by siunitx.

```
\newcommand*{\si@ifl@aded}{}
\let\si@ifl@aded\@ifl@aded
\newcommand*{\si@ifloaded}[1]{%
  \si@ifl@aded\si@extension{\si@fileprefix#1}}
```

`\si@loadfile` `#1` : file

Loading configuration files is handled here.

```
\newcommand*{\si@loadfile}[1]{%
  \si@ifloaded{#1}{}
    {\si@InputIfFileExists{\si@fileprefix#1.\si@extension}
      {}
      {\si@log@err{Failed to load file
         \si@fileprefix#1.\si@extension}
         {The configuration file requested could not be
          found}}}}
```

`\ifsi@outerinput` At various points, the package needs to know if the currently-read configuration file is the outer one.

```
\newif\ifsi@outerinput\si@outerinputtrue
```

`\si@InputIfFileExists` To allow reading configuration files after the start of the document, a private copy of `\InputIfFileExists` is needed. The main macro here simply checks if the modified version is needed.

```
\let\si@InputIfFileExists\InputIfFileExists
\AtBeginDocument{
  \renewcommand*{\si@InputIfFileExists}{%
    \ifsi@outerinput
      \expandafter\si@outerinput
    \else
      \expandafter\InputIfFileExists
    \fi}}
```

`\si@outerinput` `#1` : file
`#2` : true-code
`#3` : false-code

The outer call to load a file needs to set `@` to a letter and turn of printing using `\nullfont`.

```
\newcommand*{\si@outerinput}[3]{%
  \makeatletter
  \nullfont
  \si@outerinputfalse
  \IfFileExists{#1}%
    {#2\@addtofilelist{#1}\@@input \@filef@und\normalsize}
    {#3}%
  \normalsize
  \makeatother}
```

`\requiresiconfigs` `#1` : config-file

The configuration files depend on each other.

```
\newcommand*{\requiresiconfigs}[1]{%
  \@for\si@tempb:=#1\do{\si@loadfile{\si@tempb}}}
```

`\si@loademfile` `#1` : file

For emulation files, an additional check is made.

```
\newcommand*{\si@loademfile}[1]{%
  \@ifpackageloaded{#1}
    {\si@log@err{Emulation clash for package `#1'}
```

```
      {You have asked for emulation of package
       `#1'\MessageBreak
       (perhaps by giving siunitx a back-compatibility
       option)\MessageBreak but the package is already
       loaded!}}
    {\si@loadfile{#1}}}
```

`\si@emclash` `#1` : package
`#2` : package

A macro for emulation clashes.

```
\newcommand*{\si@emclash}[2]{%
  \si@log@err{Emulation clash: `#1' and `#2'}
    {You have asked for emulation of package `#1'\MessageBreak
     but have already loaded emulation of `#2'}}
```

`\si@emulating` `#1` : package
`#2` : version

For packages that are emulated, the LaTeX mechanism to prevent re-loading is used. The list of packages to check at the start of the document also has to be altered.

```
\newcommand*{\si@emulating}[2]{%
  \@namedef{ver@#1.sty}{#2 siunitx emulation of #1}%
  \let\si@tempa\si@blockpkgs
  \renewcommand*{\si@blockpkgs}{}%
  \@for\si@tempb:=\si@tempa\do{%
    \renewcommand*{\si@tempa}{#1}%
    \ifx\si@tempa\si@tempb\else
      \lowercase{\edef\si@tempa{#1}}%
      \lowercase{\edef\si@tempc{\si@tempb}}%
      \ifx\si@tempa\si@tempc
        \@namedef{ver@\si@tempc.sty}{#2 siunitx emulation of
          #1}%
      \else
        \si@addtolist{\si@blockpkgs}{\si@tempb}%
      \fi
    \fi}%
  \let\si@tempa\si@checkpkgs
  \renewcommand*{\si@checkpkgs}{}%
  \renewcommand*{\si@tempb}{#1}%
  \@for\si@tempc:=\si@tempa\do{%
    \ifx\si@tempb\si@tempc\else
      \si@addtolist{\si@checkpkgs}{\si@tempc}%
    \fi}}
```

With the siunitx kernel macros defined, the package can now run through finalisation. First, the default key values are set. The user options are then processed.

```
\sisetup{
  addsign=none,
  allowzeroexp=false,
  angformat=unchanged,
  astroang=false,%(
  closeerr=),%(
  closefrac=),
```

```
  colour=black,
  colourall=false,
  colourneg=false,
  decimalsymbol=fullstop,
  detectdisplay=true,
  digitsep=thin,
  dp=3,
  eVcorra=0.3ex,
  eVcorrb=0ex,
  errspace=none,
  fixdp=false,
  inlinebold=text,
  load=default,
  mathsrm=mathrm,
  mathssf=mathsf,
  mathstt=mathtt,
  mode=maths,
  negcolour=red,
  noload={},
  numaddn={},%(
  numcloseerr=),%
  numdecimal={.,},
  numdigits=0123456789,
  numexp=eEdD,
  numgobble={},
  numopenerr=(,%)
  numprod=x,
  numsign=+-\pm\mp,
  obeybold=false,
  obeyitalic=false,
  obeymode=false,
  openerr=(,%)
  openfrac=(,%)
  padangle=small,
  padnumber=lead,
  per=reciprocal,
  prefixbase=ten,
  prefixproduct=times,
  prefixsymbolic=true,
  prespace=false,
  redefsymbols=true,
  repeatunits=true,
  retainplus=false,
  seperr=false,
  sepfour=false,
  sign=plus,
  slash=slash,
  stickyper=false,
  strictarc=true,
  tabalignexp=true,
  tabautofit=false,
  tabformat=3.2,
  tabnumalign=centredecimal,
  tabtextalign=centre,
```

```
  tabunitalign=left,
  textrm=rmfamily,
  textsf=sffamily,
  texttt=ttfamily,
  tightpm=false,
  trapambigerr=true,
  trapambigfrac=true,
  unitsep=thin,
  valuesep=thin,
  xspace=false}
\ProcessOptionsX[si]<key>
```

A check is now made so that emulation takes place one file at a time, and that each file is loaded only once.

```
\ifx\@empty\si@emulate\@empty\else
  \@for\si@tempa:=\si@emulate\do{%
    \si@loademfile{\si@tempa}}
\fi
```

`\si@expanddefault` For turning the list of default choices into a loadable list.

```
\newcommand*{\si@expanddefault}[2]{%
  \expandafter\ifx\expandafter\@empty\csname si@#1\endcsname
    \@empty
  \else
    \renewcommand*{\si@tempb}{default}%
    \renewcommand*{\si@tempc}{}%
    \expandafter\@for\expandafter\si@tempa\expandafter
      :\expandafter=\csname si@#1\endcsname\do{%
      \ifx\si@tempa\si@tempb
        \si@addtolist{\si@tempc}{#2}%
      \else
        \si@addtolist{\si@tempc}{\si@tempa}%
      \fi}
    \expandafter\edef\csname si@#1\endcsname{\si@tempc}%
    \expandafter\si@addtolist\expandafter{%
      \csname si@no#1\endcsname}%
      {default}%
    \renewcommand*{\si@tempc}{}%
    \expandafter\@for\expandafter\si@tempa\expandafter
      :\expandafter=\csname si@#1\endcsname\do{%
      \si@switchfalse
      \expandafter\@for\expandafter\si@tempb\expandafter
        :\expandafter=\csname si@no#1\endcsname\do{%
        \ifx\si@tempa\si@tempb
          \si@switchtrue
        \fi
        \ifsi@switch\else
          \si@addtolist{\si@tempc}{\si@tempa}%
        \fi}}
    \@for\si@tempa:=\si@tempc\do{%
      \si@loadfile{\si@tempa}}%
  \fi}
```

The configuration and abbreviation files are loaded.

```
\si@expanddefault{load}{prefix,named,addn,prefixed,accepted,%
  physical,abbr}
```

The very last job is to load a local configuration file, if one exists, and restore catcodes.

```
\IfFileExists{siunitx.cfg}
  {\si@log@inf{Local configuration file found}%
   \InputIfFileExists{siunitx.cfg}{}{}}
  {}
\si@catcodes
```

22 Loadable modules

To keep the package relatively clear, and to make maintenance easier, the only units defined in the package itself are the base units. Everything else is a loadable module (similar to the approach in unitsdef).

22.1 Multiple prefixes

\yocto \zepto \atto \femto \pico \nano \micro \Micro \milli \centi \deci

The various SI multiple prefixes are defined here. First the small powers.

```
\ProvidesFile{si-prefix.cfg}
  [2008/09/03 v1.0l siunitx: SI Multiple prefixes]
\newprefix{\yocto}{-24}{y}
\newprefix{\zepto}{-21}{z}
\newprefix{\atto}{-18}{a}
\newprefix{\femto}{-15}{f}
\newprefix{\pico}{-12}{p}
\newprefix{\nano}{-9}{n}
```

Some testing is needed for unitsdef compatibility.

```
\ifsi@old@OHM
  \newprefix{\Micro}{-6}{\si@sym@mu}
\else
  \ifsi@gensymb\else
    \newprefix{\micro}{-6}{\si@sym@mu}
  \fi
\fi
\newprefix{\milli}{-3}{m}
\newprefix{\centi}{-2}{c}
\newprefix{\deci}{-1}{d}
```

\deca \hecto \kilo \mega \giga \tera \peta \exa \zetta \yotta

Now the large ones.

```
\newprefix{\deca}{1}{da}
\newprefix{\hecto}{2}{h}
\newprefix{\kilo}{3}{k}
\newprefix{\mega}{6}{M}
\newprefix{\giga}{9}{G}
\newprefix{\tera}{12}{T}
\newprefix{\peta}{15}{P}
\newprefix{\exa}{18}{E}
\newprefix{\zetta}{21}{Z}
\newprefix{\yotta}{24}{Y}
```

\deka Apparently, "deka" is common in the US for deca.

```
\newprefix{\deka}{1}{da}
```

\gram As the base unit of mass is the kilogram, rather than the gram, a bit of extra work
\kilogram is needed; by default the package only defines \kilogram, but with the prefixes available, this is altered to be \kilo\gram. For that, the \gram must be defined first.

```
\newunit{\gram}{g}
\renewunit{\kilogram}{\kilo\gram}
```

22.2 Derived units with specific names

\becquerel Derived units with specific names and symbols are defined. Litre is an awkward
\coulomb one, but here the UK standard is used.

```
\ProvidesFile{si-named.cfg}
  [2008/09/03 v1.0l siunitx: SI Named units]
\newunit{\becquerel}{Bq}
\newunit{\coulomb}{C}
\newunit{\farad}{F}
\newunit{\Gray}{Gy}
\newunit{\ggray}{Gy}
\newunit{\hertz}{Hz}
\newunit{\henry}{H}
\newunit{\joule}{J}
\newunit{\katal}{kat}
\newunit{\lumen}{lm}
\newunit{\lux}{lx}
\newunit{\newton}{N}
```

\ohm Some testing is needed for unitsdef compatibility.

```
\ifsi@old@OHM
  \newunit{\Ohm}{\si@sym@Omega}
\else
  \ifsi@gensymb\else
    \providеunit{\ohm}{\si@sym@Omega}
  \fi
\fi
\newunit{\pascal}{Pa}
\newunit{\siemens}{S}
\newunit{\sievert}{Sv}
\newunit{\tesla}{T}
\newunit{\volt}{V}
\newunit{\watt}{W}
\newunit{\weber}{Wb}
```

\celsius The degree celsius is a named unit.

```
\ifsi@old@OHM
  \newunit{\Celsius}{\si@sym@celsius}
\else
  \ifsi@gensymb\else
    \newunit{\celsius}{\si@sym@celsius}
```

```
  \fi
\fi
```

\radian The radian and steradian are officially derived units.

\steradian
```
\newunit{\radian}{rad}
\newunit{\steradian}{sr}
```

22.3 Units with prefixes

As in unitsdef, some commonly used prefixed units are set up. This requires `si-prefix.cfg` and `si-named.cfg`.

```
\ProvidesFile{si-prefixed.cfg}
  [2008/09/03 v1.0l siunitx: SI Prefixed units]
\requiresiconfigs{prefix,named,accepted,physical}
```

\picometre Extra distances.

\nanometre \micrometre \millimetre \centimetre \decimetre \kilometre
```
\newunit{\picometre}{\pico\metre}
\newunit{\nanometre}{\nano\metre}
\newunit{\micrometre}{\micro\metre}
\newunit{\millimetre}{\milli\metre}
\newunit{\centimetre}{\centi\metre}
\newunit{\decimetre}{\deci\metre}
\newunit{\kilometre}{\kilo\metre}
```

\femtogram Extra masses.

\picogram \nanogram \microgram \milligram
```
\newunit{\femtogram}{\femto\gram}
\newunit{\picogram}{\pico\gram}
\newunit{\nanogram}{\nano\gram}
\newunit{\microgram}{\micro\gram}
\newunit{\milligram}{\milli\gram}
```

\femtomole Now some moles.

\picomole \nanomole \micromole \millimole
```
\newunit{\femtomole}{\femto\mole}
\newunit{\picomole}{\pico\mole}
\newunit{\nanomole}{\nano\mole}
\newunit{\micromole}{\micro\mole}
\newunit{\millimole}{\milli\mole}
```

\attosecond Prefixed seconds.

\femtosecond \picosecond \nanosecond \microsecond \millisecond
```
\newunit{\attosecond}{\atto\second}
\newunit{\femtosecond}{\femto\second}
\newunit{\picosecond}{\pico\second}
\newunit{\nanosecond}{\nano\second}
\newunit{\microsecond}{\micro\second}
\newunit{\millisecond}{\milli\second}
```

\picoampere The last prefixed base units are amperes.

\nanoampere \microampere \milliampere \kiloampere
```
\newunit{\picoampere}{\pico\ampere}
\newunit{\nanoampere}{\nano\ampere}
\newunit{\microampere}{\micro\ampere}
\newunit{\milliampere}{\milli\ampere}
\newunit{\kiloampere}{\kilo\ampere}
```

More electricity-related units.

```
\newunit{\millivolt}{\milli\volt}
\newunit{\kilovolt}{\kilo\volt}
\newunit{\milliwatt}{\milli\watt}
\newunit{\kilowatt}{\kilo\watt}
\newunit{\megawatt}{\mega\watt}
\newunit{\femtofarad}{\femto\farad}
\newunit{\picofarad}{\pico\farad}
\newunit{\nanofarad}{\nano\farad}
\newunit{\microfarad}{\micro\farad}
\newunit{\millifarad}{\milli\farad}
\newunit{\millisiemens}{\milli\siemens}
```

For resistance, checks are needed again for the definition of `\ohm`.

```
\ifsi@old@OHM
  \newunit{\kiloohm}{\kilo\Ohm}
  \newunit{\megaohm}{\mega\Ohm}
  \newunit{\gigaohm}{\giga\Ohm}
\else
  \ifsi@gensymb\else
  \newunit{\kiloohm}{\kilo\ohm}
  \newunit{\megaohm}{\mega\ohm}
  \newunit{\gigaohm}{\giga\ohm}
  \fi
\fi
```

Volumes (unlike unitsdef, with litre and metre spelled correctly). `\millilitre` and `\microlitre` are defined as they are the two officially-sanctioned prefixes for the litre.

```
\newunit{\microlitre}{\micro\litre}
\newunit{\millilitre}{\milli\litre}
\newunit{\cubicmetre}{\metre\cubed}
\newunit{\cubiccentimetre}{\centi\metre\cubed}
\newunit{\centimetrecubed}{\centi\metre\cubed}
\newunit{\cubicmicrometre}{\micro\metre\cubed}
\newunit{\cubicmillimetre}{\milli\metre\cubed}
\newunit{\cubicdecimetre}{\cubic\deci\metre}
```

Areas, with metre spelled corrected; `\are` and `\hectare` are in the "temporarily accepted" file.

```
\newunit{\squaremetre}{\Square\metre}
\newunit{\squarecentimetre}{\Square\centi\metre}
\newunit{\centimetresquared}{\centi\metre\squared}
\newunit{\squarekilometre}{\Square\kilo\metre}
```

Some energy is needed by now! Notice the definition of `\kilowatthour`

```
\newunit{\millijoule}{\milli\joule}
\newunit{\kilojoule}{\kilo\joule}
\newunit{\megajoule}{\mega\joule}
\newunit{\millielectronvolt}{\milli\electronvolt}
\newunit{\kiloelectronvolt}{\kilo\electronvolt}
\newunit{\megaelectronvolt}{\mega\electronvolt}
\newunit{\gigaelectronvolt}{\giga\electronvolt}
\newunit{\teraelectronvolt}{\tera\electronvolt}
\newunit{\kilowatthour}
```

```
\newunit{\teraelectronvolt}{\tera\electronvolt}
\newunit[unitsep=none]{\kilowatthour}{\kilo\watt\hour}
```

\millihertz Frequencies.

```
\newunit{\millihertz}{\milli\hertz}
\newunit{\kilohertz}{\kilo\hertz}
\newunit{\megahertz}{\mega\hertz}
\newunit{\gigahertz}{\giga\hertz}
\newunit{\terahertz}{\tera\hertz}
```

\millinewton A few more from various areas.

```
\newunit{\millinewton}{\milli\newton}
\newunit{\kilonewton}{\kilo\newton}
\newunit{\hectopascal}{\hecto\pascal}
\newunit{\megabecquerel}{\mega\becquerel}
\newunit{\millisievert}{\milli\sievert}
```

22.4 Abbreviated units

\pA \nA The abbreviated units are sorted in one file. To allow back-compatibility with unitsdef, each one is inside an `\if` block for the appropriate option. First currents.

```
\ProvidesFile{si-abbr.cfg}
  [2008/09/03 v1.01 siunitx: Abbreviated units]
\requiresiconfigs{prefix,named,accepted,physical}
\newunit{\pA}{\pico\ampere}
\newunit{\nA}{\nano\ampere}
\newunit{\micA}{\micro\ampere}
\newunit{\mA}{\milli\ampere}
\newunit{\kA}{\kilo\ampere}
```

\Hz Then frequencies.

```
\newunit{\Hz}{\hertz}
\newunit{\mHz}{\milli\hertz}
\newunit{\kHz}{\kilo\hertz}
\newunit{\MHz}{\mega\hertz}
\newunit{\GHz}{\giga\hertz}
\newunit{\THz}{\tera\hertz}
```

\fmol Amounts of substance.

```
\newunit{\fmol}{\femto\mole}
\newunit{\pmol}{\pico\mole}
\newunit{\nmol}{\nano\mole}
\newunit{\micmol}{\micro\mole}
\newunit{\mmol}{\milli\mole}
```

\kV Potentials.

```
\newunit{\kV}{\kilo\volt}
\newunit{\mV}{\milli\volt}
```

\ml Volumes and areas

```
\provideunit{\ml}{\milli\litre}
\provideunit{\micl}{\micro\litre}
\dmc
\cms
```

```
\newunit{\cmc}{\centi\metre\cubed}
\newunit{\dmc}{\deci\metre\cubed}
\newunit{\cms}{\centi\metre\squared}
```

`\fg` Masses.

```
\newunit{\kg}{\kilo\gram}
```

`\kg` `\fg` There is a name clash with babel here in French; hopefully there will not be too many complaints.

```
\AtBeginDocument{\provideunit{\fg}{\femto\gram}}
\newunit{\SIfg}{\femto\gram}
\newunit{\pg}{\pico\gram}
\newunit{\nanog}{\nano\gram}
\newunit{\micg}{\micro\gram}
\newunit{\mg}{\milli\gram}
\newunit{\amu}{\atomicmass}
```

`\kJ` Energies.

```
\newunit{\kJ}{\kilo\joule}
\newunit{\eV}{\electronvolt}
\newunit{\meV}{\milli\electronvolt}
\newunit{\keV}{\kilo\electronvolt}
\newunit{\MeV}{\mega\electronvolt}
\newunit{\GeV}{\giga\electronvolt}
\newunit{\TeV}{\tera\electronvolt}
\newunit[unitsep=none]{\kWh}{\kilo\watt\hour}
```

`\picom` Lengths.

```
\newunit{\picom}{\pico\metre}
\newunit{\nm}{\nano\metre}
\newunit{\micm}{\micro\metre}
\newunit{\mm}{\milli\metre}
\newunit{\cm}{\centi\metre}
\newunit{\dm}{\deci\metre}
\newunit{\km}{\kilo\metre}
```

`\Sec` Finally, times.

```
\newunit{\Sec}{\second}
\newunit{\as}{\atto\second}
\newunit{\fs}{\femto\second}
```

`\ns` `\mics` The letter class (and others) define `\ps` for postscripts, so `\provideunit` is best here.

```
\provideunit{\ps}{\pico\second}
\newunit{\ns}{\nano\second}
\newunit{\mics}{\micro\second}
\newunit{\ms}{\milli\second}
```

22.5 Additional (temporary) SI units

`\angstrom` `\are` Some units are "temporarily" acceptable for use in the SI system. These are defined here, although some are in very general use.

```
\ProvidesFile{si-addn.cfg}
```

`\hectare` `\barn` `\BAR` `\bbar` `\millibar` `\gal` `\curie` `\roentgen` `\rad` `\rem`

```
  [2008/09/03 v1.01 siunitx: SI Additional units]
\newunit{\angstrom}{\si@sym@ringA}
\newunit{\are}{a}
\newunit{\hectare}{\hecto\are}
\newunit{\barn}{b}
\newunit{\BAR}{bar}
\newunit{\bbar}{bar}
\newunit{\millibar}{\milli\BAR}
\newunit{\gal}{Gal}
\newunit{\curie}{Ci}
\newunit{\roentgen}{R}
\newunit{\rad}{rad}
\newunit{\rem}{rem}
```

22.6 Units accepted for use with SI

The units which are accepted but do not fit in the above, plus `\percent` which seems to fit into this category.

```
\minute
  \hour
   \Day
  \dday
\degree
\Degree
\arcmin
\arcsec
 \litre
 \liter
 \tonne
 \neper
   \bel
\percent
\ProvidesFile{si-accepted.cfg}
  [2008/09/03 v1.01 siunitx: SI Accepted units]
\newunit{\minute}{min}
\newunit{\hour}{h}
\newunit{\Day}{d}
\newunit{\dday}{d}
\ifsi@old@OHM
  \newunit[valuesep=none]{\Degree}{\si@sym@degree}
\else
  \ifsi@gensymb\else
    \newunit[valuesep=none]{\degree}{\si@sym@degree}
  \fi
\fi
\newunit[valuesep=none]{\arcmin}{\si@sym@minute}
\newunit[valuesep=none]{\arcsec}{\si@sym@second}
\newunit{\litre}{l}
\newunit{\liter}{L}
\newunit{\tonne}{t}
\newunit{\neper}{Np}
\newunit{\bel}{B}
\newunit{\percent}{\%}
```

22.7 Units based on physical measurements

```
\si@eVspacea
\si@eVspaceb
\electronvolt
\atomicmassunit
\atomicmass
```

A few units based on physical measurements exist. For `\eV`, the need for a negative kern does make things a bit complicated.

```
\ProvidesFile{si-physical.cfg}
  [2008/09/03 v1.01 siunitx: SI Physically-measured units]
\newcommand*{\si@eVspacea}{\text{\kern-\si@eVcorra}}%
\newcommand*{\si@eVspaceb}{\text{\kern-\si@eVcorrb}}%
\newunit{\electronvolt}{e\protect\si@eVspacea V\protect%
  \si@eVspaceb}
\newunit{\atomicmass}{u}
```

```
\newunit{\atomicmassunit}{u}
```

23 Additional configurations

To provide flexibility for people in specific areas, specialised units can be set up. These are then stored separately to ease use.

23.1 Synthetic chemistry

\mmHg
\molar
\Molar
\torr
\dalton

Some useful units for synthetic chemists; although \mmHg and \Molar are outside of the SI system, they are used a lot. These are set up using \provideunit as people may have their own definitions.

```
\ProvidesFile{si-synchem.cfg}
  [2008/09/03 v1.0l siunitx: Units for synthetic chemists]
\requiresiconfigs{prefix}
\newunit{\mmHg}{mmHg}
\newunit{\molar}{\mole\per\cubic\deci\metre}
\newunit{\Molar}{\textsc{m}}
\newunit{\torr}{Torr}
\newunit{\dalton}{Da}
```

23.2 High-energy physics

Some units inspired by hepunits.

```
\ProvidesFile{si-hep.cfg}
  [2008/09/03 v1.0l siunitx: Units for high-energy physics]
\requiresiconfigs{prefix,named}
```

\micron
\mrad
\gauss

These specific to high-energy physics, but are not defined elsewhere in siunitx.[54]

```
\provideunit{\micron}{\micro\metre}
\newunit{\mrad}{\milli\rad}
\newunit{\gauss}{G}
```

\clight
\eVperc

The speed of light is used in units for the area, although of course it is not strictly a unit. However, this makes it easier to use in other units.

```
\newunit{\clight}{\ensuremath{\mathnormal{c}}}
\newunit{\eVperc}{\eV\per\clight}
```

\nanobarn
\picobarn
\femtobarn
\attobarn
\zeptobarn
\yoctobarn

Various prefixed barns.

```
\newunit{\nanobarn}{\nano\barn}
\newunit{\picobarn}{\pico\barn}
\newunit{\femtobarn}{\femto\barn}
\newunit{\attobarn}{\atto\barn}
\newunit{\zeptobarn}{\zepto\barn}
\newunit{\yoctobarn}{\yocto\barn}
```

[54] It is not clear from hepunits, but the assumption is that \mrad represents millirad and not milliradian.

\nb Abbreviations for the same, but using \provideunit to avoid any clashes.

```
\pb3858 \provideunit{\nb}{\nano\barn}
\fb3859 \provideunit{\pb}{\pico\barn}
\ab3860 \provideunit{\fb}{\femto\barn}
\zb3861 \provideunit{\ab}{\atto\barn}
\yb3862 \provideunit{\zb}{\zepto\barn}
   3863 \provideunit{\yb}{\yocto\barn}
```

23.3 Astronomy

\parsec For astronomy, the \parsec unit is needed.

```
\lightyear3864 \ProvidesFile{si-astro.cfg}
          3865   [2008/09/03 v1.0l siunitx: Units for astronomy]
          3866 \newunit{\parsec}{pc}
          3867 \newunit{\lightyear}{ly}
```

23.4 Binary units

\kibi \mebi The binary units, as specified by the IEC and made available by Slunits. First, the binary prefixes.

```
\gibi3868 \ProvidesFile{si-binary.cfg}
\tebi3869   [2008/09/03 v1.0l siunitx: Binary units]
\pebi3870 \newprefix[binary]{\kibi}{10}{Ki}
\exbi3871 \newprefix[binary]{\mebi}{20}{Mi}
\zebi3872 \newprefix[binary]{\gibi}{30}{Gi}
\yobi3873 \newprefix[binary]{\tebi}{40}{Ti}
     3874 \newprefix[binary]{\pebi}{50}{Pi}
     3875 \newprefix[binary]{\exbi}{60}{Ei}
     3876 \newprefix[binary]{\zebi}{70}{Zi}
     3877 \newprefix[binary]{\yobi}{80}{Yi}
```

\bit Now the units.

```
\byte3878 \newunit{\bit}{bit}
     3879 \newunit{\byte}{B}
```

24 Loadable locales

Some short files to provide the correct settings for various places.

24.1 United Kingdom

This is identical to the USA locale, and contains the package default values.[55]

```
3880 \ProvidesFile{si-UK.cfg}
3881   [2008/09/03 v1.0l siunitx: UK locale]
3882 \sisetup{
3883   unitsep=thin,
3884   expproduct=times,
3885   valuesep=thin,
3886   decimalsymbol=fullstop,
```

[55] The package author is in the UK.

```
  digitsep=thin,
  sepfour=false}
```

24.2 United States

The same as for the UK.

```
\ProvidesFile{si-USA.cfg}
  [2008/09/03 v1.01 siunitx: USA locale]
\sisetup{
  unitsep=thin,
  expproduct=times,
  valuesep=thin,
  decimalsymbol=fullstop,
  digitsep=thin,
  sepfour=false}
```

24.3 Germany

Germany, hopefully.

```
\ProvidesFile{si-DE.cfg}
  [2008/09/03 v1.01 siunitx: Germany locale]
\sisetup{
  unitsep=cdot,
  valuesep=thin,
  decimalsymbol=comma,
  expproduct=cdot,
  digitsep=thin,
  sepfour=false}
```

24.4 South Africa

Design decisions taken from the same section of SIstyle.

```
\ProvidesFile{si-ZA.cfg}
  [2008/09/03 v1.01 siunitx: South Africa locale]
\sisetup{
  unitsep=cdot,
  valuesep=thin,
  expproduct=times,
  decimalsymbol=comma,
  digitsep=thin,
  sepfour=false}
```

25 Emulation code

Each emulation mode loads an appropriate definition file. This then alters the package defaults, and defines new macros provided by the emulated package.

25.1 units

The very first thing to do here is a reload check, as things could go wrong with unitsdef emulation.

```
\si@ifloaded{units}{\endinput}{}
```

The units package is quite easy to emulate, as it only has a few options and macros. There is also no error checking in units for conflicting options, so users probably expect none.

```
\ProvidesFile{si-units.cfg}
  [2008/09/03 v1.01 siunitx: Emulation of units]
\si@emulating{units}{1998/08/04 v0.9b}
\si@ifloaded{SIunits}
  {\si@emclash{units}{SIunits}\endinput}{}
\si@ifloaded{sistyle}
  {\si@emclash{units}{sistyle}\endinput}{}
```

To emulate units, `\per` must give fractions.

```
\sisetup{per=fraction,fraction=nice,obeybold,inlinebold=maths,
  ,obeymode}
\ifsi@old@tight
  \sisetup{valuesep=thin}
\fi
\ifsi@old@loose
  \sisetup{valuesep=space}
\fi
\ifsi@old@ugly
  \sisetup{fraction=ugly}
\fi
```

`\unit [#1]` : number
`#2` : unit

The units version of `\unit` is similar to `\SI`. Here and in `\unitfrac` the `\num` macro is used; thus the number given really has to be a number. However, if users are using siunitx rather than units they should expect more checking of input. As the units package uses the current mode, this has to be detected.

```
\si@newrobustcmd*{\unit}[2][]{%
  \ifmmode
    \SI{#1}{#2}%
  \else
    \SI[obeyfamily,obeyitalic]{#1}{#2}%
  \fi}
```

`\unitfrac [#1]` : number
`#2` : numerator
`#3` : denominator

`\unitfrac` is a bit more of a hack.

```
\si@newrobustcmd*{\unitfrac}[3][]{%
  \begingroup
    \si@fam@mode%
    \ifmmode\else
      \sisetup{obeyfamily,obeyitalic}%
    \fi
    \si@ifnotmtarg{#1}
      {\num{#1}\ensuremath{\si@valuesep}}%
    \si@frac{#2}{#3}
  \endgroup}
```

25.2 unitsdef

The package begins with the usual identification of what is happening. Although `si-units.cfg` makes the same checks, the error will make more sense if it comes here, in the event of a clash.

```
\ProvidesFile{si-unitsdef.cfg}
  [2008/09/03 v1.01 siunitx: Emulation of unitsdef]
\si@emulating{unitsdef}{2005/01/04 v0.2}
\si@ifloaded{SIunits}
  {\si@emclash{unitsdef}{SIunits}\endinput}{}
\si@ifloaded{sistyle}
  {\si@emclash{unitsdef}{sistyle}\endinput}{}
```

Emulation of units is needed for unitsdef to work.

```
\requiresiconfigs{units}
```

The unitsdef package loads some packages that siunitx does not. In particular, it loads textcomp and fontenc. This could be important for output, and so the same is done here.

```
\RequirePackage{textcomp}
\RequirePackage[T1]{fontenc}
```

The multitude of package options for unitsdef need to be handled.

```
\sisetup{mode=text,allowoptarg,prespace}
\ifsi@old@noxspace
  \sisetup{xspace=false}
\fi
```

The various options for loading unit abbreviations have to be handled. Here, any request to avoid abbreviations prevents any loading.

```
\ifsi@old@noabbr
  \sisetup{noload=abbr}
\fi
\ifsi@old@nofrequncyabbr
  \sisetup{noload=abbr}
\fi
\ifsi@old@nomolabbr
  \sisetup{noload=abbr}
\fi
\ifsi@old@novoltageabbr
  \sisetup{noload=abbr}
\fi
\ifsi@old@novolumeabbr
  \sisetup{noload=abbr}
\fi
\ifsi@old@noweightabbr
  \sisetup{noload=abbr}
\fi
\ifsi@old@noenergyabbr
  \sisetup{noload=abbr}
\fi
\ifsi@old@nolengthabbr
  \sisetup{noload=abbr}
\fi
\ifsi@old@notimeabbr
```

```
  \sisetup{noload=abbr}
\fi
```

`\unitvaluesep` To emulate the `\unitvaluesep` macro, a hack is needed of the original xkeyval macro for `valuesep`, as well of course as a definition of the macro itself.

```
\newcommand*{\unitvaluesep}{\,}
\renewcommand*{\si@valuesep}{\text{\unitvaluesep}}
\define@choicekey*+[si]{key}{valuesep}[\si@tempa]
  {space,thin,med,medium,thick,none}
  {\renewcommand*\unitvaluesep\@nameuse{si@fix@##1}%
   \si@log@debug{Option valuesep set to ##1}}
  {\si@log@debug{Option valuesep set to ##1}%
   \renewcommand*\unitvaluesep{##1}}
```

`\unitsignonly` `\ilu` `\arc` Some rather straight-forward definitions, with just a bit of fun to get the spacing correct.

```
\let\unitsignonly\si
\si@newrobustcmd*{\ilu}[2][]{%
  \begingroup
    #1\unitvaluesep%
    \unit{#2}%
  \endgroup}
\let\arc\ang
```

`\unitSIdef` `\si@unitSIdef` The unitsdef package uses a different approach to setting the font inside its version of `\SI`. The problem is the same as for `\unitvaluesep`, but with the added problem that siunitx uses `\csname` ... `\endcsname`.

```
\newcommand*{\unitSIdef}{\upshape}
\newcommand*{\si@unitSIdef}{\unitSIdef\selectfont}
\sisetup{textrm=si@unitSIdef}
```

`\per` Rather awkwardly, unitsdef uses `\per` in a different way to siunitx.

```
\let\per\relax
\si@newrobustcmd*{\per}[2]{%
  \begingroup
    \si@xspacefalse
    \renewcommand*{\unitvaluesep}{}%
    \unitfrac{#1}{#2}%
  \endgroup}
```

`\unittimes` `\unitsep` `\unitsuperscript` Some pretty straight-forward stuff again; notice that the automatic analyser for units has to be turned off for this to work.

```
\newcommand*{\unittimes}{\ensuremath{\cdot}}
\newcommand*{\unitsep}{\,}
\renewcommand*{\si@unt@unithook}{\si@unt@litouttrue}
\sisetup{unitsep=none}
\newcommand*{\unitsuperscript}{\tothe}
```

`\newnosepunit` `\renewnosepunit` Simple aliases.

```
\newcommand*{\newnosepunit}{\newunit[valuesep=none]}
\newcommand*{\renewnosepunit}{\renewunit[valuesep=none]}
```

\setTextOmega
\setMathOmega
\setTextmu
\setMathmu
\setTextCelsius
\setMathCelsius
\setMathDegree
\setTextDegree

Controlling symbols is a simple translation job; as only one setting is used by siunitx in text mode, a bit of extra work is needed.

```
\newcommand*{\setTextOmega}[2]{%
  \renewcommand*{\si@textOmega}{%
    \begingroup
    \edef\si@tempa{\sfdefault}%
    \ifx\f@family\si@tempa
      \expandafter#2%
    \else
      \expandafter#1%
    \fi
  \endgroup}}
\newcommand*{\setMathOmega}[1]{\sisetup{mathsOmega=#1}}
\newcommand*{\setTextmu}[2]{%
  \renewcommand*{\si@textmu}{%
    \begingroup
    \edef\si@tempa{\sfdefault}%
    \ifx\f@family\si@tempa
      \expandafter#2%
    \else
      \expandafter#1%
    \fi
  \endgroup}}
\newcommand*{\setMathmu}[1]{\sisetup{mathsmu=#1}}
\newcommand*{\setTextCelsius}[2]{%
  \renewcommand*{\si@textcelsius}{%
    \begingroup
    \edef\si@tempa{\sfdefault}%
    \ifx\f@family\si@tempa
      \expandafter#2%
    \else
      \expandafter#1%
    \fi
  \endgroup}}
\newcommand*{\setMathCelsius}[1]{\sisetup{mathscelsius=#1}}
\newcommand*{\setMathDegree}[2]{%
  \renewcommand*{\si@textdegree}{%
    \begingroup%
    \edef\si@tempa{\sfdefault}%
    \ifx\f@family\si@tempa
      \expandafter#2%
    \else
      \expandafter#1%
    \fi
  \endgroup}}
\newcommand*{\setTextDegree}[1]{\sisetup{textdegree=#1}}
```

The `ohm` and `OHM` options are checked, and some sanity is ensured. This needs to happen before loading the configuration files.

```
\ifsi@old@OHM
  \ifsi@old@ohm
    \si@log@inf{Both 'ohm' and 'OHM' options given\MessageBreak
      Using default behaviour for unitsdef}
    \expandafter\expandafter\expandafter\si@old@OHMfalse
```

```
  \fi
\fi
```

\liter
\ton
\days

Tonne is spelled as "ton" by unitsdef, which is wrong in the UK at least (1 ton = 40 cwt = 2240 lb!).

```
\ifsi@old@liter
  \ifsi@old@LITER
    \si@log@inf{Both 'liter' and 'LITER' options
      given\MessageBreak Using default behaviour for unitsdef}
  \else
    \renewunit{\liter}{l}
  \fi
\fi
\newunit{\ton}{t}
\newunit{\days}{d}
```

\picometer
\nanometer
\micrometer
\millimeter
\centimeter
\decimeter
\kilometer

Extra distances.

```
\newunit{\picometer}{\pico\meter}
\newunit{\nanometer}{\nano\meter}
\newunit{\micrometer}{\micro\meter}
\newunit{\millimeter}{\milli\meter}
\newunit{\centimeter}{\centi\meter}
\newunit{\decimeter}{\deci\meter}
\newunit{\kilometer}{\kilo\meter}
```

\femtoliter
\picoliter
\nanoliter
\microliter
\milliliter
\centiliter
\deciliter
\hectoliter
\cubicmeter
\cubicmicrometer
\cubicmillimeter

Volumes with US spellings.

```
\newunit{\femtoliter}{\femto\liter}
\newunit{\picoliter}{\pico\liter}
\newunit{\nanoliter}{\nano\liter}
\newunit{\microliter}{\micro\liter}
\newunit{\milliliter}{\milli\liter}
\newunit{\centiliter}{\centi\liter}
\newunit{\deciliter}{\deci\liter}
\newunit{\hectoliter}{\hecto\liter}
\newunit{\cubicmeter}{\meter\cubed}
\newunit{\cubicmicrometer}{\micro\meter\cubed}
\newunit{\cubicmillimeter}{\milli\meter\cubed}
```

\squaremeter
\squarecentimeter
\squarekilometer
\ar
\hectar

Areas, including the mis-spellings for \are and \hectare.

```
\newunit{\squaremeter}{\Square\meter}
\newunit{\squarecentimeter}{\Square\centi\meter}
\newunit{\squarekilometer}{\Square\kilo\meter}
\newunit{\ar}{a}
\newunit{\hectar}{\hecto\ar}
```

\kv
\mv

The code for unitsdef has the capitalisation wrong for \kV and \mV.

```
\ifsi@old@noabbr
\else
  \ifsi@old@novoltageabbr\else
    \newunit{\kv}{\kilo\volt}
    \newunit{\mv}{\milli\volt}
  \fi
\fi
```

\sek \fg \fl \pl \nl \micl \ml \cl \dl \hl

There are some slightly different abbreviations, plus some which are not officially allowed.

```
\ifsi@old@noabbr\else
  \ifsi@old@notimeabbr\else
    \newunit{\sek}{\second}
  \fi
  \ifsi@old@noweightabbr\else
    \newunit{\fg}{\femto\gram}
  \fi
  \ifsi@old@novolumeabbr\else
    \newunit{\fl}{\femto\liter}
    \newunit{\pl}{\pico\liter}
    \newunit{\nl}{\nano\liter}
    \newunit{\micl}{\micro\liter}
    \newunit{\ml}{\milli\liter}
    \newunit{\cl}{\centi\liter}
    \newunit{\dl}{\deci\liter}
    \newunit{\hl}{\hecto\liter}
  \fi
\fi
```

\calory \kilocalory

unitsdef spells calorie incorrectly, and it is also not an SI unit.

```
\newunit{\calory}{cal}
\newunit{\kilocalory}{\kilo\calory}
```

\uBar

unitsdef uses `\ubar` for bar.

```
\newunit{\uBar}{ba}
```

\gensymbohm \gensymbcelsius \gensymbmicro \gensymbdegree

If the options relating to gensymb are given, then the package *has* to be loaded. The definitions are then renamed; a slight awkward feature is that the hyphen character needs to be a letter. To avoid needing to worry about this again, a second switch is set up.

```
\catcode`\-=11\relax
\ifsi@old@redef-gensymb
  \expandafter\si@gensymbtrue
\fi
\catcode`\-=12\relax
\ifsi@gensymb
  \RequirePackage{gensymb}
  \AtBeginDocument{
    \let\gensymbohm\ohm
    \let\gensymbcelsius\celsius
    \let\gensymbmicro\micro
    \let\gensymbdegree\degree
    \let\ohm\@undefined
    \let\celsius\@undefined
    \let\micro\@undefined
    \let\degree\@undefined
    \ifsi@old@OHM\else
      \newunit{\ohm}{\si@sym@Omega}
      \newunit{\celsius}{\si@sym@celsius}
      \newprefix{\micro}{\si@sym@mu}{-6}
      \newunit{\degree}{\si@sym@degree}
```

```
    \fi}
\fi
```

The configuration files can now be loaded.

```
\requiresiconfigs{prefix,named,addn,accepted}
```

The `noconfig` option could be ignored, but it costs little to let is be used.

```
\ifsi@old@noconfig\else
  \InputIfFileExists{unitsdef.cfg}
    {\si@log@inf{unitsdef config file loaded}}
    {\si@log@inf{unitsdef config file not found}}
\fi
```

25.3 SIstyle

After setting the necessary defaults, the emulation code defines the macros in SIstyle as given in the manual for that package.

```
\ProvidesFile{si-sistyle.cfg}
  [2008/09/03 v1.01 siunitx: Emulation of SIstyle]
\si@emulating{sistyle}{2006/12/20 v2.3}
\sisetup{%
  sepfour=true,
  obeyfamily,
  obeyitalic=true,
  numsign=+-,
  strictarc=false,
  unitsep=cdot}
```

`\SIobeyboldtrue` `\SIobeyboldfalse` Some simple switches, but not using `\newif`.

```
\newcommand*{\SIobeyboldtrue}{\sisetup{obeybold=true}}
\newcommand*{\SIobeyboldfalse}{\sisetup{obeybold=false}}
```

`\num` `\si@sis@num` `\si@sis@numstar` To get the correct behaviour for `\num`, some redefinitions are needed to handle to optional `*`.

```
\let\num\relax
\si@newrobustcmd*{\num}{%
  \@ifstar
    {\si@sis@numstar}
    {\si@sis@num}}
\newcommand*{\si@sis@num}[2][]{%
  \begingroup%
    \sisetup{#1}%
    \expandafter\si@out@num\expandafter{\si@num{#2}}%
  \endgroup}
\newcommand*{\si@sis@numstar}[2][]{%
  \begingroup%
    \sisetup{mode=text,obeybold}%
    \sisetup{#1}%
    \expandafter\si@out@num\expandafter{\si@num{#2}}%
  \endgroup}
```

`\pnt` The `\pnt` macro is needed as `.` is active inside `\SI`. The name is exactly the same as in SIstyle, but the implementation is different. This is not defined by the

main package as there are better ways of including numbers in the output than this.

```
\newcommand*{\pnt}{\ensuremath{\si@decimalsymbol}}
```

`\SIgroupfourtrue` Switches for grouping four characters.

`\SIgroupfourfalse`

```
\newcommand*{\SIgroupfourtrue}{\sisetup{sepfour=true}}
\newcommand*{\SIgroupfourfalse}{\sisetup{sepfour=false}}
```

`\SIunitsep` Whatever is given here is passed through to `\sisetup`.

`\SIunitspace` `\SIunitdot`

```
\newcommand*{\SIunitsep}[1]{\sisetup{valuesep={#1}}}
\newcommand*{\SIunitspace}[1]{\sisetup{unitspace={#1}}}
\newcommand*{\SIunitdot}[1]{\sisetup{unitsep={#1}}}
```

`\SIdecimalsymbol` The same is true here, with the appropriate translation.

`\SIthousandsep` `\SIproductsign` `\SIdecimalsign`

```
\newcommand*{\SIdecimalsymbol}[1]{\sisetup{decimalsymbol={#1}}}
\newcommand*{\SIthousandsep}[1]{\sisetup{digitsep={#1}}}
\newcommand*{\SIproductsign}[1]{\sisetup{expproduct={#1}}}
\newcommand*{\SIdecimalsign}[1]{\sisetup{decimalsymbol={#1}}}
```

`\si@sis@savefont` `#1` : setting
`#2` : argument
The font definitions need a bit of extra work doing. As both settings here have `@` as a letter, all should be fine.

```
\newcommand*{\si@sis@savefont}[2]{%
  \@namedef{si@sis@#1}{#2}%
  \sisetup{#1=si@sis@#1}}
```

`\SImathrm` The font control macros have to ensure that a macro name is passed to `\sisetup`.

`\SImathsf` `\SImathtt`

```
\newcommand*{\SImathrm}[1]{\si@sis@savefont{mathrm}{#1}}
\newcommand*{\SImathsf}[1]{\si@sis@savefont{mathsf}{#1}}
\newcommand*{\SImathtt}[1]{\si@sis@savefont{mathtt}{#1}}
```

`\SIdefaultMfam` The same for the default keys.

`\SIdefaultNfam` `\SIdefaultTfam`

```
\newcommand*{\SIdefaultMfam}[1]{\si@sis@savefont{mathrm}{#1}}
\newcommand*{\SIdefaultNfam}[1]{\si@sis@savefont{mathnumrm}{#1}}
\newcommand*{\SIdefaultTfam}[1]{\si@sis@savefont{textrm}{#1}}
```

`\ensureupmath` The `\ensureupmath` command guarantees processing by the font-matching system. The argument cannot be processed here, so care is needed.

```
\si@newrobustcmd*{\ensureupmath}[1]{%
  \begingroup
    \sisetup{mode=maths,obeyitalic=false}%
    \si@out{#1}%
  \endgroup}
```

`\degC` A few extra symbol names are needed.

`\degF` `\arcdeg`

```
\newcommand*{\degC}{\si@sym@celsius}
\newcommand*{\arcdeg}{\si@sym@degree}
\newcommand*{\degF}{\si@sym@degree F}
```

\AddToSIstyle Finally, the locale control.

```
      \SIstyle 4220 \newcommand*{\SIstyle}[1]{\sisetup{locale=#1}}
 \SIstyleToLang 4221 \newcommand*{\SIstyleToLang}[2]{\sisetup{loctolang=#1:#2}}
\si@sis@addtolocale 4222 \newcommand*{\AddToSIstyle}{%
                4223   \si@switchfalse
                4224   \@ifstar
                4225     {\si@switchtrue
                4226      \si@sis@addtolocale}
                4227     {\si@sis@addtolocale}}
                4228 \newcommand*{\si@sis@addtolocale}[2]{%
                4229   \ifsi@switch
                4230     \expandafter\let\csname si@loc@#1@extra\endcsname\relax
                4231   \fi
                4232   \addtolocale{#1}{#2}}
```

25.4 SIunits

SIunits emulation starts in much the same way.

```
                4233 \ProvidesFile{si-SIunits.cfg}
                4234   [2008/09/03 v1.01 siunitx: Emulation of SIunits]
                4235 \si@emulating{SIunits}{2007/12/02 v1.36}
                4236 \sisetup{
                4237   unitsep=thick,
                4238   valuesep=thick,
                4239   prefixproduct=\si@valuesep,
                4240   trapambigfrac=false,
                4241   stickyper}
                4242 \requiresiconfigs{prefix,named,accepted,physical}
```

\reciprocal A few very simple translations, using the internal version of \per to allow changes of output style.

```
          \rp
         \per 4243 \newcommand*{\reciprocal}{\sisetup{per=reciprocal}\si@per}
         \usk 4244 \let\rp\reciprocal
       \power 4245 \renewcommand*{\per}{\sisetup{per=slash}\si@per}
   \rpsquare 4246 \newcommand*{\usk}{}
    \rpcubic 4247 \newcommand*{\power}[1]{#1\tothe}
     \fourth 4248 \newcommand*{\rpsquare}{\sisetup{per=reciprocal}\si@per\Square}
   \rpfourth 4249 \newcommand*{\rpcubic}{\sisetup{per=reciprocal}\si@per\cubic}
             4250 \newpower{\fourth}{4}
             4251 \newcommand*{\rpfourth}{\sisetup{per=reciprocal}\si@per\fourth}
```

\rpsquared Here, some low-level switch changing is needed.

```
    \rpcubed 4252 \newcommand*{\rpsquared}{%
             4253   \sisetup{per=reciprocal}\si@unt@pertrue\si@unt@perseentrue%
             4254     \squared}
             4255 \newcommand*{\rpcubed}{%
             4256   \sisetup{per=reciprocal}\si@unt@pertrue\cubed}
```

\SIsetup The various package spacing options are processed. They also have to be correctly handled by the \SIsetup macro.

```
\si@siu@setup 4257 \newcommand*{\SIsetup}[1]{%
             4258   \@for\si@tempa:=#1\do{%
             4259     \@ifundefined{ifsi@old@#1}
```

```
      {\si@log@warn{Unknown SIunits option `#1'}}
      {\csname si@old@#1true\endcsname}}
  \si@siu@setup}
\newcommand*{\si@siu@setup}{%
  \ifsi@old@cdot
    \sisetup{unitsep=cdot}%
  \fi
  \ifsi@old@thickspace
    \sisetup{unitsep=thick}%
  \fi
  \ifsi@old@mediumspace
    \sisetup{unitsep=medium}%
  \fi
  \ifsi@old@thinspace
    \sisetup{unitsep=thin}%
  \fi
  \ifsi@old@thickqspace
    \sisetup{valuesep=thick}%
  \fi
  \ifsi@old@mediumqspace
    \sisetup{valuesep=medium}%
  \fi
  \ifsi@old@thinqspace
    \sisetup{valuesep=thin}%
  \fi}
\si@siu@setup
```

`\square` `\squaren` SIunits does slightly different things about the clash with `\square`, and either redefines this macro or provides `\squaren`.

```
\ifsi@old@squaren
  \newpower{\squaren}{2}
\fi
\AtBeginDocument{%
  \@ifundefined{square}
    {\newpower{\square}{2}}
    {\ifsi@old@amssymb
       \renewpower{\square}{2}
     \else
       \ifsi@old@squaren\else
         \si@log@warn{\string\square\space already
           defined\MessageBreak SIunits mode may cause
           errors}%
       \fi
     \fi}}
```

`\gray` The potential clash with PStricks is also handled differently; here, `\Gray` will already be defined by the siunitx kernel.

```
\AtBeginDocument{
  \@ifundefined{gray}
    {\newunit{\gray}{Gy}}
    {\ifsi@old@pstricks
       \renewunit{\gray}{Gy}
     \else
       \ifsi@old@Gray\else
```

```
          \si@log@warn{\string\gray\space already
            defined\MessageBreak SIunits mode may cause
            errors}%
        \fi
      \fi}}
```

`\unit` The `\unit` macro is defined.

`\unita`
```
\ifsi@old@italian
  \let\unita\SI
\else
  \let\unit\SI
\fi
```

The miscellaneous options are moped up.

```
\ifsi@old@textstyle
  \sisetup{mode=text}
\fi
\ifsi@old@binary
  \sisetup{alsoload=binary}
\fi
\ifsi@old@noams
  \AtBeginDocument{%
    \renewcommand*{\si@textmu}{\ensuremath\si@mathsmu}}
\fi
```

`\arcminute` `\arcsecond` `\rperminute` `\ton` `\degreecelsius` The unit macros defined by SIunits that are not defined by siunitx (by default).

```
\newunit[valuesep=none]{\arcminute}{\si@sym@minute}
\newunit[valuesep=none]{\arcsecond}{\si@sym@second}
\newunit{\rperminute}{r/min}
\newunit{\ton}{t}
\newunit{\degreecelsius}{\celsius}
```

`\addunit` This is an alias for `\newunit`.

```
\let\addunit\newunit
```

`\addprefix` A little more work for `\addprefix`.

```
\newcommand*{\addprefix}[2]{\newprefix{#1}{#2}}
```

`\si@siu@newunit` `[#1]` : power
`\si@siu@power` `#2` : numerator
`\si@siu@newunithook` `#3` : denominator

To save some code, making new units which need a `...np` variant is handled by a dedicated macro.

```
\newcommand*{\si@siu@newunit}[3][]{%
```

A test is needed to sort out `\square`.

```
  \renewcommand*{\si@tempa}{#1}%
  \renewcommand*{\si@tempb}{square}%
  \renewcommand*{\si@siu@power}{}%
  \ifx\@empty\si@tempa\@empty\else
    \ifx\si@tempa\si@tempb
      \renewcommand*{\si@siu@power}{\ssquare}%
    \else
```

```
      \edef\si@siu@power{%
        \expandafter\noexpand\csname #1\endcsname}%
    \fi
  \fi
```

The necessary information is now stored in temporary macros.

```
  \edef\si@tempa{%
    \expandafter\noexpand\csname #2per#1#3\endcsname}%
  \edef\si@tempb{%
    \expandafter\noexpand\csname #2\endcsname\noexpand\per
    \expandafter\noexpand\si@siu@power
    \expandafter\noexpand\csname #3\endcsname}%
  \expandafter\expandafter\expandafter\newunit\expandafter%
    \expandafter\expandafter{\expandafter\si@tempa\expandafter}%
      \expandafter{\si@tempb}
  \edef\si@tempa{%
    \expandafter\noexpand\csname #2per#1#3np\endcsname}%
  \edef\si@tempb{%
    \expandafter\noexpand\csname #2\endcsname\noexpand
    \reciprocal\expandafter\noexpand\si@siu@power
    \expandafter\noexpand\csname #3\endcsname}%
  \expandafter\expandafter\expandafter\newunit\expandafter
    \expandafter\expandafter{\expandafter\si@tempa\expandafter}%
      \expandafter{\si@tempb}%
  \si@siu@newunithook[#1]{#2}{#3}}
\providecommand*{\si@siu@newunithook}[3][]{}
```

The basic units are now defined; these only have a single csname in each of the numerator and denominator.

```
\si@siu@newunit{gray}{second}
\si@siu@newunit[square]{metre}{second}
\si@siu@newunit{joule}{mole}
\si@siu@newunit[cubic]{mole}{metre}
\si@siu@newunit[square]{radian}{second}
\si@siu@newunit{radian}{second}
\si@siu@newunit[cubic]{squaremetre}{metre}
\si@siu@newunit[cubic]{katal}{metre}
\si@siu@newunit{coulomb}{mol}
\si@siu@newunit[square]{ampere}{metre}
\si@siu@newunit[cubic]{kilogram}{metre}
\si@siu@newunit[cubic]{coulomb}{metre}
\si@siu@newunit{volt}{metre}
\si@siu@newunit[square]{coulomb}{squaremetre}
\si@siu@newunit{farad}{metre}
\si@siu@newunit[square]{watt}{metre}
\si@siu@newunit[square]{joule}{metre}
\si@siu@newunit[cubic]{newton}{metre}
\si@siu@newunit{newton}{kilogram}
\si@siu@newunit{joule}{kelvin}
\si@siu@newunit{joule}{kilogram}
\si@siu@newunit{coulomb}{kilogram}
\si@siu@newunit{squaremetre}{second}
\si@siu@newunit[square]{squaremetre}{second}
\si@siu@newunit[square]{candela}{metre}
```

```
\si@siu@newunit{ampere}{metre}
\si@siu@newunit{joule}{tesla}
\si@siu@newunit{henry}{metre}
\si@siu@newunit{kilogram}{second}
\si@siu@newunit[square]{kilogram}{metre}
\si@siu@newunit{kilogram}{metre}
\si@siu@newunit[square]{newton}{metre}
\si@siu@newunit{watt}{kilogram}
\si@siu@newunit[cubic]{watt}{metre}
\si@siu@newunit{squaremetre}{kilogram}
\si@siu@newunit{cubicmetre}{kilogram}
\si@siu@newunit{newton}{metre}
\si@siu@newunit[cubic]{squaremetre}{second}
\si@siu@newunit{metre}{second}
\si@siu@newunit[cubic]{joule}{metre}
\si@siu@newunit{cubicmetre}{second}
```

`\si@siu@newunitx` `\si@siu@newunitxhook` For the more complex units, a slightly different approach is used; four arguments are required, and have to cover everything.

```
\newcommand*{\si@siu@newunitx}[4]{%
  \expandafter\newunit\expandafter{\csname #1per#2\endcsname}
    {#3\per#4}%
  \expandafter\newunit\expandafter{\csname #1per#2np\endcsname}
    {#3\reciprocal#4}
  \si@siu@newunitxhook{#1}{#2}{#3}{#4}}
\providecommand*{\si@siu@newunitxhook}[4]{}
```

The units are defined.

```
\si@siu@newunitx{kilogramsquaremetre}{second}
  {\kilogram\squaremetre}{\second}
\si@siu@newunitx{squaremetre}{newtonsecond}{\squaremetre}
  {\newton\second}
\si@siu@newunitx{kilogrammetre}{second}{\kilogram\metre}
  {\second}
\si@siu@newunitx{kilogram}{squaremetresecond}{\kilogram}
  {\squaremetre\second}
\si@siu@newunitx{joule}{molekelvin}{\joule}{\mole\kelvin}
\si@siu@newunitx{kilogram}{kilomole}{\kilogram}{\kilo\mole}
\si@siu@newunitx{kilogrammetre}{squaresecond}{\kilogram\metre}
  {\second\squared}
\si@siu@newunitx{watt}{squaremetresteradian}{\watt}
  {\squaremetre\steradian}
\si@siu@newunitx{joule}{kilogramkelvin}{\joule}
  {\kilogram\kelvin}
\si@siu@newunitx{watt}{metrekelvin}{\watt}{\metre\kelvin}
\si@siu@newunitx{kilogram}{cubicmetrecoulomb}{\kilogram}
  {\cubic\metre\coulomb}
\si@siu@newunitx{kilogram}{secondcubicmetre}{\kilogram}
  {\second\cubicmetre}
```

`\si@siu@unity` A bit of cleverness to get the "1" correct; to avoid any clash, the unit is given an internal name.

```
\newunit{\si@siu@unity}{1}
\si@siu@newunitx{}{squaremetresecond}{\si@siu@unity}
```

```
  {\squaremetre\second}
```

A few compound units that are best defined directly.

```
\newunit{\pascalsecond}{\pascal\second}
\newunit{\amperemetresecond}{\ampere\metre\second}
\newunit{\ohmmetre}{\ohm\metre}
\newunit{\newtonmetre}{\newton\metre}
\let\newtonmetrenp\newtonmetre
\newunit{\kilogramsquaremetre}{\kilogram\squaremetre}
\let\kilogramsquaremetrenp\kilogramsquaremetre
```

`\si@siu@newprefix` `#1` : prefix

To generate the prefixes correctly, a small saving in repetition.

```
\newcommand*{\si@siu@newprefix}[1]{%
  \edef\si@tempa{\expandafter\noexpand\csname #1d\endcsname}%
  \edef\si@tempb{\expandafter\noexpand\csname #1\endcsname}%
  \expandafter\expandafter\expandafter\newcommand\expandafter
    \expandafter\expandafter*\expandafter\expandafter
    \expandafter{\expandafter\si@tempa\expandafter}\expandafter
      {\expandafter\si@prefixsymbolicfalse\si@tempb}}
```

This is now implemented.

```
\si@siu@newprefix{yocto}
\si@siu@newprefix{zepto}
\si@siu@newprefix{atto}
\si@siu@newprefix{femto}
\si@siu@newprefix{pico}
\si@siu@newprefix{nano}
\si@siu@newprefix{micro}
\si@siu@newprefix{milli}
\si@siu@newprefix{centi}
\si@siu@newprefix{deca}
\si@siu@newprefix{deka}
\si@siu@newprefix{hecto}
\si@siu@newprefix{kilo}
\si@siu@newprefix{mega}
\si@siu@newprefix{giga}
\si@siu@newprefix{tera}
\si@siu@newprefix{peta}
\si@siu@newprefix{exa}
\si@siu@newprefix{zetta}
\si@siu@newprefix{yotta}
\ifsi@old@binary
  \si@siu@newprefix{kibi}
  \si@siu@newprefix{mebi}
  \si@siu@newprefix{gibi}
  \si@siu@newprefix{tebi}
  \si@siu@newprefix{pebi}
  \si@siu@newprefix{exbi}
\fi
```

`\derradian` `\dersteradian` `\derhertz` `\dernewton` `\derpascal` `\derjoule` `\derwatt` `\dercoulomb` `\dervolt` `\derfarad` `\derohm`

The derived units may need to be defined.

```
\ifsi@old@derived
  \newunit{\derradian}{\metre\reciprocal\metre}
```

```
\newunit{\dersteradian}{\squaremetre\rpsquare\metre}
\newunit{\derhertz}{\reciprocal\second}
\newunit{\dernewton}{\metre\kilogram\second\rpsquared}
\newunit{\derpascal}{\newton\rpsquare\metre}
\newunit{\derjoule}{\newton\metre}
\newunit{\derwatt}{\joule\reciprocal\second}
\newunit{\dercoulomb}{\ampere\second}
\newunit{\dervolt}{\watt\reciprocal\ampere}
\newunit{\derfarad}{\coulomb\reciprocal\volt}
\newunit{\derohm}{\volt\reciprocal\ampere}
```

\dersiemens In two blocks!

```
\newunit{\dersiemens}{\ampere\reciprocal\volt}
\newunit{\derweber}
  {\squaremetre\kilogram\second\rpsquared\reciprocal\ampere}
\newunit{\dertesla}{\weber\rpsquare\metre}
\newunit{\derhenry}{\weber\reciprocal\ampere}
\newunit{\dercelsius}{\kelvin}
\newunit{\derlumen}{\candela\steradian}
\newunit{\derlux}{\lumen\rpsquare\metre}
\newunit{\derbecquerel}{\derhertz}
\newunit{\dergray}{\joule\reciprocal\kilogram}
\newunit{\dersievert}{\dergray}
\newunit{\derkatal}{\rp\second\usk\mole}
\fi
```

\radianbase Also the “derived-in-base”.

```
\ifsi@old@derivedinbase
\newunit{\radianbase}{\metre\reciprocal\metre}
\newunit{\steradianbase}{\squaremetre\rpsquare\metre}
\newunit{\hertzbase}{\reciprocal\second}
\newunit{\newtonbase}{\metre\kilogram\second\rpsquared}
\newunit{\pascalbase}{\reciprocal\metre\kilogram\second%
  \rpsquared}
\newunit{\joulebase}{\squaremetre\kilogram\second\rpsquared}
\newunit{\wattbase}{\squaremetre\kilogram\rpcubic\second}
\newunit{\coulombbase}{\ampere\second}
\newunit{\voltbase}
  {\squaremetre\kilogram\rpcubic\second\reciprocal\ampere}
\newunit{\faradbase}
  {\rpsquare\metre\reciprocal\kilogram\fourth\second\ampere%
    \squared}
\newunit{\ohmbase}
  {\squaremetre\kilogram\rpcubic\second\rpsquare\ampere}
```

\siemensbase Also in two blocks.

```
\newunit{\siemensbase}
  {\rpsquare\metre\reciprocal\kilogram\cubic\second\ampere%
     \squared}
\newunit{\weberbase}
  {\squaremetre\kilogram\second\rpsquared\reciprocal\ampere}
\newunit{\teslabase}{\kilogram\second\rpsquared\reciprocal%
   \ampere}
\newunit{\henrybase}
```

\weberbase \teslabase \henrybase \celsiusbase \lumenbase \luxbase \becquerelbase \graybase \sievertbase \katalbase

```
    {\squaremetre\kilogram\second\rpsquared\rpsquare\ampere}
  \newunit{\celsiusbase}{\kelvin}
  \newunit{\lumenbase}{\candela\squaremetre\rpsquare\metre}
  \newunit{\luxbase}{\candela\squaremetre\rpfourth\metre}
  \newunit{\becquerelbase}{\hertzbase}
  \newunit{\graybase}{\squaremetre\second\rpsquared}
  \newunit{\sievertbase}{\graybase}
  \newunit{\katalbase}{\rp\second\mole}
\fi
```

Any configuration file is used if found.

```
\InputIfFileExists{SIunits.cfg}
  {\si@log@inf{SIunits config file loaded}}
  {\si@log@inf{SIunits config file not found}}
```

25.5 hepunits

The hepunits package provides some rather odd unit names, which are not really to be encouraged.

```
\ProvidesFile{si-hepunits.cfg}
  [2008/09/03 v1.01 siunitx: Emulation of hepunits]
\si@emulating{hepunits}{2007/09/27}
\requiresiconfigs{SIunits,accepted,prefix,hep}
```

`\invbarn` Inverses barn units.

`\invnanobarn` `\invpicobarn` `\invfemtobarn` `\invattobarn` `\invzeptobarn` `\invyoctobarn`

```
\ifsi@old@noprefixcmds\else
  \newunit{\invbarn}{\per\barn}
  \newunit{\invnanobarn}{\per\nano\barn}
  \newunit{\invpicobarn}{\per\pico\barn}
  \newunit{\invfemtobarn}{\per\femto\barn}
  \newunit{\invattobarn}{\per\atto\barn}
  \newunit{\invzeptobarn}{\per\zepto\barn}
  \newunit{\invyoctobarn}{\per\yocto\barn}
```

`\invnb` Also available abbreviated.

`\invpb` `\invfb` `\invab` `\invzb` `\invyb`

```
  \newunit{\invnb}{\per\nano\barn}
  \newunit{\invpb}{\per\pico\barn}
  \newunit{\invfb}{\per\femto\barn}
  \newunit{\invab}{\per\atto\barn}
  \newunit{\invzb}{\per\zepto\barn}
  \newunit{\invyb}{\per\yocto\barn}
\fi
```

`\invcmsqpersecond` Luminosity.

`\invcmsqpersec` `\lumiunits`

```
\newunit{\invcmsqpersecond}{\per\Square\centi\metre\per\second}
\newunit{\invcmsqpersec}{\per\Square\centi\metre\per\second}
\newunit{\lumiunits}{\per\Square\centi\metre\per\second}
```

`\inveV` The inverse of an electron-volt, plus prefixes.

`\minveV` `\minveV` `\kinveV` `\MinveV` `\GinveV` `\TinveV`

```
\newunit{\inveV}{\per\electronvolt}
\newunit{\minveV}{\per\milli\electronvolt}
\newunit{\kinveV}{\per\kilo\electronvolt}
\newunit{\MinveV}{\per\mega\electronvolt}
```

```
\newunit{\GinveV}{\per\giga\electronvolt}
\newunit{\TinveV}{\per\tera\electronvolt}
```

\eVoverc
\eVovercsq

Some combinations of electron-volts and the speed of light. As these are called `over`, they are set with a slash. The `eVcorrb` values have been set for Computer Modern.

```
\newunit[per=slash,eVcorrb=0.6ex]{\eVoverc}
  {\electronvolt\per\clight}
\newunit[per=slash,eVcorrb=0.6ex]{\eVovercsq}
  {\electronvolt\per\Square\clight}
```

\meVoverc

Prefixed combinations, first of the speed of light.

\keVoverc
\MeVoverc
\GeVoverc
\TeVoverc

```
\newunit[per=slash,eVcorrb=0.6ex]{\meVoverc}
  {\milli\electronvolt\per\clight}
\newunit[per=slash,eVcorrb=0.6ex]{\keVoverc}
  {\kilo\electronvolt\per\clight}
\newunit[per=slash,eVcorrb=0.6ex]{\MeVoverc}
  {\mega\electronvolt\per\clight}
\newunit[per=slash,eVcorrb=0.6ex]{\GeVoverc}
  {\giga\electronvolt\per\clight}
\newunit[per=slash,eVcorrb=0.6ex]{\TeVoverc}
  {\tera\electronvolt\per\clight}
```

\meVovercsq

Then of the square.

\keVovercsq
\MeVovercsq
\GeVovercsq
\TeVovercsq

```
\newunit[per=slash,eVcorrb=0.6ex]{\meVovercsq}
  {\milli\electronvolt\per\Square\clight}
\newunit[per=slash,eVcorrb=0.6ex]{\keVovercsq}
  {\kilo\electronvolt\per\Square\clight}
\newunit[per=slash,eVcorrb=0.6ex]{\MeVovercsq}
  {\mega\electronvolt\per\Square\clight}
\newunit[per=slash,eVcorrb=0.6ex]{\GeVovercsq}
  {\giga\electronvolt\per\Square\clight}
\newunit[per=slash,eVcorrb=0.6ex]{\TeVovercsq}
  {\tera\electronvolt\per\Square\clight}
```

25.6 fancynum

fancynum only does things with numbers, so there is only a little emulation and a few macros needed.

```
\ProvidesFile{si-fancynum.cfg}
  [2008/09/03 v1.0l siunitx: Emulation of fancynum]
\si@emulating{fancynum}{2000/08/08 0.92}
\sisetup{decimalsymbol=cdot,digitsep=comma}
```

\fnum

The `\fnum` macro is rather restricted, but this is not reproduced. Instead, it is `\let` as an alias to the `\num` macro.

```
\let\fnum\num
```

\setfnumdsym

The control macros are defined.

\setfnumgsym
\setfnummsym

```
\newcommand*{\setfnumdsym}[1]{\sisetup{decimalsymbol={#1}}}
\newcommand*{\setfnumgsym}[1]{\sisetup{digitsep={#1}}}
\newcommand*{\setfnummsym}[1]{\sisetup{expproduct={#1}}}
```

The various package options are now processed if necessary.

```
\ifsi@old@english
  \sisetup{decimalsymbol=cdot,digitsep=comma}
\fi
\ifsi@old@french
  \sisetup{decimalsymbol=comma,digitsep=fullstop}
\fi
\ifsi@old@tight
  \sisetup{expproduct=tighttimes}
\fi
\ifsi@old@loose
  \sisetup{expproduct=times}
\fi
\ifsi@old@thinspaces
  \sisetup{digitsep=thin}
\fi
\ifsi@old@commas
  \sisetup{digitsep=comma}
\fi
\ifsi@old@plain
  \sisetup{digitsep=none}
\fi
```

25.7 fancyunits

The fancyunits package is not available on CTAN, but is available from its authors homepage [7]. It is similar to SIunits, and so most of the code is shared here. However, a few bits of set up occur first, and an emulation-clash test is needed.

```
\ProvidesFile{si-fancyunits.cfg}
  [2008/09/03 v1.01 siunitx: Emulation of fancyunits]
\si@emulating{fancyunits}{2007/02/01 v1.0.1}
\si@ifloaded{SIunits}
  {\si@log@err{SIunits emulation loaded\MessageBreak before
    fancyunits emulation}{You need to load the fancyunits
    emualtion\MessageBreak code before that for
    SIunits.\MessageBreak Try emulate=fancyunits as the first
    option when\MessageBreak loading siunitx}}{}
```

`\si@siu@newunithook` `\si@siu@newunitxhook` To create the extra macros provided by fancyunits, the SIunits emulation code is changed to add the "`uf`" variants.

```
\newcommand*{\si@siu@newunithook}[3][]{%
  \edef\si@tempa{%
    \expandafter\noexpand\csname #2per#1#3uf\endcsname}%
  \renewcommand*{\si@tempb}{stickyper,per=fraction,
    fraction=nice}%
  \edef\si@tempc{%
    \noexpand\sisetup{\si@tempb}%
    \expandafter\noexpand\csname #2\endcsname\noexpand\si@per%
    \expandafter\noexpand\si@siu@power%
    \expandafter\noexpand\csname #3\endcsname}%
  \expandafter\expandafter\expandafter\newunit\expandafter
    \expandafter\expandafter{\expandafter\si@tempa\expandafter}%
      \expandafter{\si@tempc}%
```

```
  \edef\si@tempa{%
    \expandafter\noexpand\csname #2per#1#3Uf\endcsname}%
  \renewcommand*{\si@tempb}{stickyper,per=fraction,
    fraction=frac}%
  \edef\si@tempc{%
    \noexpand\sisetup{\si@tempb}%
    \noexpand\def\noexpand\si@frc@hook{\noexpand\textstyle}%
    \expandafter\noexpand\csname #2\endcsname\noexpand\si@per%
    \expandafter\noexpand\si@siu@power%
    \expandafter\noexpand\csname #3\endcsname}%
  \expandafter\expandafter\expandafter\newunit\expandafter
    \expandafter\expandafter{\expandafter\si@tempa\expandafter}%
      \expandafter{\si@tempc}%
  \edef\si@tempa{%
    \expandafter\noexpand\csname #2per#1#3UF\endcsname}%
  \edef\si@tempc{%
    \noexpand\sisetup{\si@tempb}%
    \noexpand\def\noexpand\si@frc@hook{\noexpand\displaystyle}%
    \expandafter\noexpand\csname #2\endcsname\noexpand\si@per%
    \expandafter\noexpand\si@siu@power%
    \expandafter\noexpand\csname #3\endcsname}%
  \expandafter\expandafter\expandafter\newunit\expandafter
    \expandafter\expandafter{\expandafter\si@tempa\expandafter}%
      \expandafter{\si@tempc}}
\newcommand*{\si@siu@newunitxhook}[4]{%
  \expandafter\newunit\expandafter{\csname #1per#2uf\endcsname}
    {\sisetup{stickyper,per=fraction,fraction=nice}%
     #3\si@per#4}%
  \expandafter\newunit\expandafter{\csname #1per#2Uf\endcsname}
    {\sisetup{stickyper,per=fraction,fraction=frac}%
     \renewcommand*{\si@frc@hook}{\textstyle}%
     #3\si@per#4}%
  \expandafter\newunit\expandafter{\csname #1per#2UF\endcsname}
    {\sisetup{stickyper,per=fraction,fraction=frac}%
     \renewcommand*{\si@frc@hook}{\displaystyle}%
     #3\si@per#4}}
```

With that done, the emulation modules can be loaded.

```
\requiresiconfigs{SIunits,addn,astro}
\sisetup{obeyall}
```

There is one fancyunits-specific option to handle. The other options all get sent through to the SIunits system.[56]

```
\ifsi@old@spaceqspace
  \sisetup{valuesep=space}
\fi
```

`\paminute` `\parsecond` `\AstroE` `\oersted` `\ton`

fancyunits provides some extra units, plus tonne spelled incorrectly (again).

```
\newunit{\paminute}{'}
\newunit{\parsecond}{''}
\newunit{\AstroE}{AE}
\newunit{\oersted}{OE}
```

[56] As SIunits is rather more widely known than fancyunits, any other options which could be for either are assumed to be for SIunits.

```
\provideunit{\ton}{t}
```

`\decaD` An additional prefix.

```
\let\decaD\decad
```

`\ufrac` The fractional unit macros need to be reproduced.

`\Ufrac` `\UFrac`

```
\newcommand*{\ufrac}[2]{%
  \si[stickyper,per=fraction,fraction=nice]{#1\si@per#2}}
\newcommand*{\Ufrac}[2]{%
  \ensuremath{\textstyle{%
    \si[stickyper,per=fraction,fraction=frac]{#1\si@per#2}}}}
\newcommand*{\UFrac}[2]{%
  \ensuremath{\displaystyle{%
    \si[stickyper,per=fraction,fraction=frac]{#1\si@per#2}}}}
```

`\pow` A slightly-shorted named `\power`.

```
\let\pow\power
```

`\Squaremetre` An alias for `\squaremetre`.

```
\let\Squaremetre\squaremetre
```

As with Slunits, there is now a list of compound units to add. Only a few are not covered by the Slunits emulation.

```
\si@siu@newunit{Gray}{second}
\si@siu@newunit[square]{Squaremetre}{metre}
\si@siu@newunitx{Squaremetre}{newtonsecond}{\Square\metre}
  {\newton\second}
\si@siu@newunit{Squaremetre}{second}
\si@siu@newunit[square]{Squaremetre}{squaresecond}
\si@siu@newunit{Squaremetre}{kilogram}
\si@siu@newunit[cubic]{Squaremetre}{second}
```

Part V
Notes

26 Change History

27 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z

28 References

[1] The IUPAC Green Book, 1993. http://old.iupac.org/publications/books/gbook/green_book_2ed.pdf.

[2] Victor Eijkhout. TEX by Topic, 2007. http://www.eijkhout.net/tbt/.

[3] http://physics.nist.gov/cuu/Units/index.html.

[4] http://www.bipm.org/fr/si/.

[5] http://www.bipm.org/en/si/.

[6] http://www.bipm.org/en/si/si_brochure/.

[7] Heiko Bauke. fancyunits, 2007. http://www.mpi-hd.mpg.de/personalhomes/bauke/LaTeX/Tips_und_Tricks/fancyunits/index.php.