si — A comprehensive (SI) units package*

Joseph Wright†

Released 2008/02/20

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). A number of LaTeX packages have been developed to provide consistent application of the various rules: Slunits, sistyle, unitsdef and units are the leading examples. The numprint package provides a large number of number-related functions, while dcolumn and rccol provide tools for typesetting tabular numbers.

The si package takes the best from the existing packages, and adds new features and a consistent interface. A number of new ideas have been incorporated, to fill gaps in the existing provision. The package also provides backward-compatibility with Slunits, sistyle, unitsdef and units. The aim is to have one package to handle all of the possible unit-related needs of LaTeX users.

Contents

*This file describes version v.06a, last revised 2008/02/20.

†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.

LaTeX support typesetting numbers and units is currently provided by a number of packages: SIunits, sistyle, units, unitsdef and numprint. Each package has advantages, and no single package has so far displaced use of the others. The aim of the si package is to learn from the existing implementations to provide a coherent and extendable approach to the problem. The original aim of developing si was to produce a "version 2" successor to SIunits or sistyle. However, as the package has been developed a number of inconsistencies in the interfaces of the existing packages have been noted. Thus by default si does not follow any one of the existing packages; the interface is intended to be self-consistent and logical. As a result, si is now intended as a new package. The author hopes that by providing a comprehensive package here, the other "unit" packages will be superseded.

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. In emulation mode, si tries hard to work in exactly the same manner as the emulated package. However, in certain places inconsistencies exist due to changes in the underlying mechanisms used. These are noted where they are known to the package author.

1 Existing packages

Both the SIunits an sistyle package are designed to allow typesetting of SI units, with consistent typography and following the rules laid down by NIST [1].[3] The sistyle package concentrates on typography, whereas the SIunits package is focussed on careful application of SI units in place of other systems. The key strengths of the two packages can be summarised as follows:[4]

- sistyle
  - Easy input for numbers, for example typing `\num{5.8e-7}` and getting "5.8×10^{-7}" as output.
  - Input of numbers can be with comma or dot as decimal sign and is independent of output.
  - Output style can follow particular regional conventions (*via* `\sistyle`) or even be dependent on the document language (implemented by `\sistyleToLang`).
- SIunits
  - The look of units can be easily be changed in the whole document by redefining the commands.

[1] General Conference on Weights and Measures.
[2] International System of Units.
[3] sistyle also allows the use of German and South African rules "out of the box".
[4] Thanks to Stefan Pinnow for the excellent summary on `comp.text.tex`, on which this is based.

- New units can be added on a document-specific basis (`\addunit`), for example to match journal requirements (*e.g.* "wt.-%" *versus* "wt%").
- Package aims to enforce use of SI units as far as possible.

At the same time, the unitsdef package allows "trailing" units, so for example `10\metre` to yield "10 m" with a non-breaking and definable space. However, this does not allow control of the format of the number. The unitsdef package is built on top of units, which is an even more general. The numprint package provides fine control for printing numbers, with features beyond those in sistyle. Finally, the hepunits package adds various physics units to SIunits.

2 The wish-list

The wish-list for the new package has developed as ideas have suggested themselves. This has been both from the package author and various contributors on `comp.text.tex`. Anything on the list is likely to be looked at: nothing is ruled out. Items marked **To do** are definitely going to be looked at, those marked **Ongoing** have at least some code written. Items marked **Completed** seem to work properly, and at this stage seem to be finished (in the sense that changes are not planned). However, that does not mean they are finalised or bug free!

- keyval package interface, with modification of settings in document using this system (like hyperref). **Completed**
- Remove need for `\usk` separator between unit names when using SIunits-style setup. **Completed**
- "Prefix" units, such as currency, possibly as an optional argument to `\SI`: `\SI[per=slash]{10}[\pounds]{\per\kilo\gram}` ⇒ "£10/kg" (suggested by Allan Ristow). **Completed**
- Stand-alone setting of units, for example `\unitsym{kg.m/s^2}` to give "kg m/s^2", for use in table headings, *etc.* (suggested by Allan Ristow). **Completed**
- numprint-like handling of numbers (suggested by Allan Ristow). **Completed**
- Ability to handle crystallography-style estimated standard deviations, *e.g.* `\SI{1.550(2)}{\angstrom}` ⇒ "1.550(2) Å".[5] **Completed**
- Ability to understand and alter negative powers/fractions, and typeset these flexibly; thus `\unitsym{\metre\per\second}` could give "m s^{-1}", "m/s", "$\frac{\mathrm{m}}{\mathrm{s}}$" or "$^{\mathrm{m}}/_{\mathrm{s}}$" depending on a package setting (suggested by Stefan Pinnow). **Completed**
- More logical handling of powers; for example `\deci\cubic\metre` or `\deci\metre\cubed` give "dm^3", but `\deci\cubed\metre` does not even though `\deci` *cannot* be cubed (adapted suggestion from Stefan Pinnow). **Completed**

[5] $(1.550 \pm 0.002) \times 10^{-10}$ m.

- Use of trailing units (as in unitsdef package, so for example `10\metre` to give "10 m" (suggestion from Lan Thuy Pham). **Completed**
- Support use of non-Latin characters where appropriate, for example µ in units as in `\unitsym{µm}` to give "µm" (suggested by Martin Heller). **Completed**
- Integrate the core functionality of the SIunits and sistyle packages (suggested by Danie Els,[6] as well as a key point of the review). **Completed**
- Modular design, with loadable definitions for different areas and typographic conventions. **Completed**
- Emulation of existing packages [units (**Completed**), unitsdef (**Completed**), sistyle (**Completed**), SIunits (**Completed**)] to allow easy upgrading. **Completed**
- Typesetting angles in "astronomy" style, for example `\ang[astroang]{30;5;3.2}` to give 30°5′3.″2 (suggested by Alok Singhal). **Completed**

3 Road map

The existing units packages provide valuable information on the problems and pitfalls of designing a package in this area. They have also shown how to solve many of the issues arising. However, in writing a new package, consistent interface design has been important. This is logical to the package author, but may not be to anyone else. The functionality provided also aims to cover everything from the existing packages and the suggestions contributed at `comp.text.tex`, but omissions are likely to exist. The current release of si is therefore regarded as a development version, to gain feedback from users and to find errors. The current "road map" for future releases is (broadly) given here.

v0.6 Seek feedback on implementation to date (current release);

v0.7 Add or modify functionality based on feedback, implement new suggestions for wish list;

v0.8 Fix bugs from 0.7 release, interface freeze;

v0.9 Release-candidate: fix remaining bugs from v0.8, complete remaining documentation;

v1.0 First release of completed package.

Depending on user feedback, the gap between these releases will vary. However, to finalise all of the potential issues will take some months (to allow time for proper testing).

[6] Current maintainer of sistyle.

Part II
Using the si package

4 Requirements

si requires a reasonably up to date TEX system. The package requires ε-TEX-extensions, which should be available on most systems.[7] 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, si does not load xfrac. If you want to use `fraction=sfrac`, *you* need to load xfrac in your preamble. 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.[8]

5 Loading the package

si is loaded by the usual LATEX method.

```
\usepackage{si}\\
\usepackage[key=option]{si}
```

As is shown in the example, the package can be loaded with one or more options, using the keyval system. The full range of package options are described in Section 11; some options are described in the along with the appropriate user macros. Most of the user macros accept the same keyval settings as an optional argument.

6 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.

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

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

1 123 12 345 123 456
0.1 0.123 0.1234 0.123 45
1×10^{10} 3.45×10^{-4} -10^{10}

```
\num{1}     \num{123}      \num{12345}  \num{123456}\\
\num{0.1}   \num{0.123}    \num{0,1234} \num{.12345}\\
\num{1e10}  \num{3.45d-4}  \num{-e10}\\
```

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.[9]

$1.234(5) = 1.234 \pm 0.005$

```
$\num{1.234(5)} = \num{1.234} \pm \num{0.005}$
```

A number of effects are available as options. These are fully explained in Section 11. 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.[10]

1 234 567 890 1 234 567 890

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

si 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} 1×5^{10}

```
\num{1e10}  \num[expproduct=cdot]{1e10}\\
\num{2e20}  \num[exppower=5]{1e10}
```

7 Tabular material

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.[11]

By default, the decimal marker of the contents is placed at the centre of the column (Table 1). This is achieved by having a negative value for the key `tabformat`. The second method for centring content is to specify a number of digits before and after the decimal sign to be reserved by the package. Thus in the example, `tabformat=2.4` provides space for two digits before the decimal

[9] This is common in chemical crystallography.
[10] This document is typeset using lowercase numbers in text mode, which emphasises the effect here.
[11] 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

marker and four after.[12] If an integer is given as the argument of `tabformat`, equal space is reserved before and after the decimal marker for numerals, and the column is typeset flush right. As is shown in the fourth column, any other options may also be set on a column-by-column basis.

```
\begin{table}
  \caption{Behaviour of \texttt{s} column type
    \label{tbl:default}}
  \centering
  \begin{tabular}{ss[tabformat=2.4]s[tabformat=4]%
    s[tabformat=2.4,decimalsign=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}
```

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). The contents of non-numeric cells are centred; this can be altered by using the standard `\multicolumn` macro. The use of digit separators in table columns is accounted for: extra space is reserved if digit separators will be added.

The use of exponents in the body of tables is not recommended; unlike numprint, si does not provide additional alignment of exponents. Certain strange effects can also result from the exponent marker letters (by default `dDeE`) being picked up by the package in text in columns. For example, using `<{after}` after a `s` column will add "aftr" at the end of each value. To avoid this, wrap any problematic text in *two* sets of braces (so for example, put `<{{after}}`). Alternatively, as part of the beginning of the table issue the command

```
\sisetup{numexp={}}
```

after `\begin{table}`.

[12] The separator for the number of digits before and after the decimal mark may be essentially any non-numeric character. Thus `tabformat=2.4`, `tabformat=2,4` and `tabformat=2a4` all give the same result.

8 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°
3.6789° 3.6789°
9°8′7″ 9° 8′ 7″

```
\ang{1.05}   \ang[decimalsign=comma]{1.05}\\
\ang{3.6789} \ang[digitsep=comma]{3.6789}\\
\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.[13]

1°2′3.4″
1°2′3.″4

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

9 Units and values

`\SI` The core aim of si is correctly typesetting values which have units. The main output macro here is `\SI`, which has the same syntax as the equivalent macro in sistyle and unitsdef. 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.

[13] The exact positioning of the symbols over the decimal marker is currently something of a guess. Some feedback on the "correct" result would be very welcome.

$1.23\,\mathrm{J\,mol^{-1}\,K^{-1}}$
$0.23 \times 10^{7}\,\mathrm{cd}$
£1.99/kg
$70\,\mathrm{m\,s^{-1}}$
$1.345\,\mathrm{A/mol}$

```
\SI[mode=text]{1.23}{J.mol^{-1}.K^{-1}}\\
\SI{.23e7}{\candela}\\
\SI[per=slash]{1.99}[\pounds]{\per\kilogram}\\
\SI{70}{\metre\per\second}\\
\SI[per=frac,fraction=nice]{1,345}{\ampere\per\mole}\\
```

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

9.1 Literal units

Units can be input in two ways, inspired by sistyle and Slunits. 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 fullstop (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.

$10\,\mathrm{kg\,m\,s^{-2}}$
$1.453\,\mathrm{g/cm^{3}}$
$33.562\,\mathrm{cd\,s}$
$100\,\mathrm{m\cdot s^{-2}}$

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

9.2 The unit interpreter

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

$10\,\mathrm{kg\,m\,s^{-2}}$
$1.453\,\mathrm{g\,cm^{-3}}$
$33.562\,\mathrm{cd\,s}$
$100\,\mathrm{m\cdot s^{-2}}$
$4.56 \times 10^{3}\,\mathrm{m\,s^{-1}}$

```
\SI{10}{\kilo\gram\metre\per\second\squared}\\
\SI{1.453}{\gram\per\cubic\centi\metre}\\
\SI{33.562}{\candela\second}\\
\SI[unitsep=cdot]{100}{\metre\per\Square\second}\\
\SI[prefix=power]{4.56}{\kilo\metre\per\second}
```

On its own, this is very similar to Slunits, and is less convenient than the direct input method.[14] 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.

$10\,\frac{\mathrm{g\,m}}{\mathrm{s^2}}$
$1.453\,\mathrm{g/cm^{3}}$
$33.562\,\mathrm{cd\,s}$
$100\,\mathrm{m/s^{2}}$

```
\SI[per=frac,fraction=frac]{10}{\gram\metre\per\second\squared}\\
\SI[per=slash]{1.453}{\gram\per\cubic\centi\metre}\\
\SI{33.562}{\candela\second}\\
\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.

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

9.3 Powers of units

\Square
\squared
\cubic
\cubed

Including powers in units is handled using a "natural language" method. Thus preceding a unit by `\Square` or `\cubic` with raise the unit to the appropriate power, while `\squared` or `\cubed` follow the unit they apply to.[15]

$10\,\mathrm{m}^2$	`\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}\\`
$30\,{}^{1}/_{\mathrm{s}^3}$	`\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}`

\tothe

For powers not defined above or with `\newpower`, the `\tothe` macro can be used "in line" to produce a power. As follows from standard English usage, this comes after the unit.[16]

$16.86\,\mathrm{m}^4$	`\SI{16.86}{\metre\tothe{4}}\\`
$7.895\,\mathrm{cd}^{0.5}$	`\SI{7.895}{\candela\tothe{0.5}}\\`
$7.895\,\mathrm{N}^{-6}$	`\SI{7.895}{\newton\tothe{-6}}\\`
$1.34\,\mathrm{K}^{-7}$	`\SI{1.34}{\per\kelvin\tothe{7}}`

9.4 Units with no values

\unitsym

For typesetting the symbol for a unit on its own, with the full font control and without extra spaces, the `\unitsym` macro is provided.[17] The macro name avoids a clash with the functionality of the earlier packages, but is similar to `\ilu` from the unitsdef package.

$\mathrm{kg\,m/s^2}$	`\SI{}{kg.m/s^2}\\`
$\mathrm{kg\,m/s^2}$	`\unitsym{kg.m/s^2}\\`
$\mathrm{mol{\cdot}dm^{-3}}$	`\unitsym[mode=text,unitsep=cdot]{\mole\per\cubic\deci\metre}`

9.5 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. si supports both operation modes, with the limitation that units trailing values loose font control of the value.

[15] The `\Square` macro is capitalised to avoid a name clash with pstricks.
[16] Suggestions for a macro name for before the unit for the same job are welcome!
[17] The same effect can be achieved using the `\SI` macro with an empty numerical argument.

Table 2: 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 3: The SI prefixes (`load=prefix`)

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

123 m
123 K
234 A
6 s

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

When used in this way, the units *do not* take an optional keyval argument.

9.6 Pre-defined units, prefixes and powers

The package always defines the seven base SI units, irrespective of any package options given (Table 2). 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*.

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`, si also defines the standard prefixes for powers of ten (Table 3). 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 which have assigned names and symbols (Table 4). Note that `\Gray` is capitalised to avoid a name clash with the pstricks package.[18] In addition to these units, there

[18]The macros `\ohm` and `\celsius` are not defined by si if the gensymb package is loaded.

Table 4: The derived SI units with defined names (`load=derived`)

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

Table 5: Units derived from experiments (`load=physical`)

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

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 5), those considered "accepted" (Table 6), and those accepted temporarily (Table 7).[19]

9.7 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`). These are available by loading the `si-abbr.cfg` file (*i.e.* `load=abbr`). In common with the units discussed above, the prefixed and abbreviated unit definitions are loaded by default.

Table 8: 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`

Continued on next page

[19] These are supposed to be replaced over time by SI units.

Unit	Macro	Symbol	Abbreviation
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
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			
squaremetre	\squaremetre	m^2	
squarecentimetre	\squarecentimetre	cm^2	
squarekilometre	\squarekilometre	km^2	
Volumes			
millilitre	\millilitre	ml	\ml
microlitre	\microlitre	µl	\micl

Continued on next page

Unit	Macro	Symbol	Abbreviation
centimetrecubed	`\centimetrecubed`	cm^3	`\cmc`
	`\centimetrecubed`	cm^3	`\cmc`
cubicdecimetre	`\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`	nV	`\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`
Powers			
milliwatt	`\milliwatt`	mW	
kilowatt	`\kilowatt`	kW	
megawatt	`\megawatt`	MW	
Capacitance			
femtofarad	`\femtofarad`	fF	
picofarad	`\picofarad`	pF	
nanofarad	`\nanofarad`	nF	
microfarad	`\microfarad`	µF	
millifarad	`\millifarad`	mF	
Resistance			
kiloohm	`\kiloohm`	kΩ	
megaohm	`\megaohm`	MΩ	
gigaohm	`\gigaohm`	GΩ	
millisiemens	`\millisiemens`	mS	
Forces			
millinewton	`\millinewton`	mN	

Continued on next page

Unit	Macro	Symbol	Abbreviation
kilonewton	\kilonewton	kN	
Other units			
hectopascal	\hectopascal	hPa	
megabecquerel	\megabecquerel	MBq	
millisievert	\millisievert	mSv	

9.8 Specialist units

\mmHg
\molar
\Molar

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. si comes with the predefined files `alsoload=chemistry` and `alsoload=hep`. The later defines the units from the hepunits package not provided elsewhere here. The former adds the common chemistry units \mmHg, \molar and \Molar. The \Molar macro is somewhat awkward, as it can be given as either "M" or "M". The later is obviously easily confused with the sign for the prefix mega.

\bit
\byte

The package also comes with equipped for `alsoload=binary`. This provides the binary units and prefixes. The extra units are \bit and \byte, with the new prefixes listed in Table 9.

9.9 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 to this unit only The most obvious example for using this macro is the \degree unit.[20] The (first) optional argument to \SI and \unitsym can be used to override the settings for the unit.

3.1415°

12 345XXX 67 890 XXX

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

Output that is only safe in maths mode should be protected with \ensuremath; text-only input requires \text. In the example below, \mathnormal is used to force the font choice only for the single character.[21]

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

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

\newpower
\renewpower
\providepower

Powers are defined: \newpower[⟨*post*⟩]{⟨*power*⟩}{⟨*num*⟩}. ⟨*power*⟩ is the name of the power macro, an ⟨*num*⟩ is the (positive) number it represents. The later argument is always processed internally by \num, but *must* be a number.

[20] Although the \ang macro is preferred for this job.
[21] The \mathrm font used for this document has an "ß" at the π position.

Table 6: Units accepted for use with SI (`load=accepted`)

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

Table 7: Additional (temporary) SI units (`load=addn`)

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

Table 9: Binary prefixes (`alsoload=binary`)

Prefix	Macro	Power
kibi	`\kibi`	2^{10}
mebi	`\mebi`	2^{20}
gibi	`\gibi`	2^{30}
tebi	`\tebi`	2^{40}
pebi	`\pebi`	2^{50}
exbi	`\exbi`	2^{60}

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}\\
\unitsym{\kilogram\totheforth}\\
\unitsym{\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{`⟨*prefix*⟩`}{`⟨*symbol*⟩`}{`⟨*powers-ten*⟩`}`, where ⟨*powers-ten*⟩ is the number of powers of ten the prefix represents. For example, `\kilo` is defined:

```
\newprefix{\kilo}{k}{3}
```

10 Font control

Following the lead of sistyle, si 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
30 kg

50

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

By giving the `obeyfamily` option, the surrounding font family (serif, sans serif, fixed width) is used for inline materials. Inside the display maths environments, the currently active maths font is used. The `obeybold` option causes the bold setting to be obeyed in the same way.

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,obeybold,obeyitalic} \\
\ang{1;1;1} $\ang{1;1;1}$ \\
\sffamily \ang{2;2;2} $\ang{2;2;2}$ \\
\textbf{\ang{3;3;3}} \boldmath $\ang{3;3;3}$ \\
\sisetup{mode=text} \emph{\ang{4;4;4}} $\ang{4;4;4}$ \\
\[ \ang{5;5;5} \]
```

11 Package options

`\sisetup`

The "native" options for the package are all given using the keyval methods. 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 `true`, so giving the option name alone sets the flag to true.

- `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. The default value is given in bold.
- `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. As with simple choice options, the default is given in bold.
- `option=⟨literal⟩` The given value is used as a literal by the package.
- `option=⟨macro⟩` These options expect a macro name as a value; the macro name is then used by the package. Note that the name does *not* include the leading `\`.
- `option=⟨length⟩` Requires a TEX lengths, for example `0.5ex`.
- `option=⟨list⟩` Takes a list of one or more items, which are not determined in advance.

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.

11.1 Font family and style

The font used when typesetting material can be tightly controlled using si. A number of options affect how the package matches the surrounding font, and the font families used to achieve this.

- `obeyfamily=⟨bool⟩` By default, the font family used for typesetting does *not* match the surroundings. This is altered using the `obeyfamily` switch; when active, serif (Roman), sans serif and typewriter fonts are detected.
- `mode=⟨choice⟩` The output of si can be typeset using either text or maths fonts. By default, maths mode is used, but this can be altered setting the `mode` option to `text`.
- `textmode=⟨bool⟩` A shortcut for `[`mode=text].
- `obeymode=⟨bool⟩` The package can detect and use the surrounding maths or text mode, if requested. Default is `false`.
- `obeybold=⟨bool⟩` If the typeset text should obey the local value of the bold setting, then this option should be set: the default is false.
- `inlinebold=⟨choice⟩` For inline maths, the package can check either the surrounding maths or the surrounding text. The options here are **`text`** and`maths` (or `math`).
- `obeyitalic=⟨bool⟩` Italic *versus* upright shape is handled slightly differently to bold. The option works in text mode, but has no effect in maths mode. This is because font changes plus italic is not possible in maths mode (for example, see the result of `$\mathit{\mathrr{10}}$`.

- `mathdefault=`⟨*macro*⟩ The default shape used for text printed in maths mode. The default is the value stored in `mathrm`.
- `textdefault=`⟨*macro*⟩ The default shape used for text printed in text mode. The default is the value stored in `mathrm`.
- `mathnumdefault=`⟨*macro*⟩ The default shape used for numbers printed in maths mode. The default is the value stored in `mathrm`.
- `textnumdefault=`⟨*macro*⟩ The default shape used for numbers printed in text mode. The default is the value stored in `mathrm`.
- `mathrm=`⟨*macro*⟩ The font command used in maths mode when the surrounding text is serif. The default is `mathrm`; the other maths font defaults follow the same pattern.
- `mathsf=`⟨*macro*⟩ The font command used in maths mode when the surrounding text is sans serif.
- `mathrm=`⟨*macro*⟩ The font command used in maths mode when the surrounding text is fixed width.
- `textrm=`⟨*macro*⟩ The font command used in text mode when the surrounding text is serif. The default is `rmfamily`; the other text font defaults follow the same pattern.
- `textsf=`⟨*macro*⟩ The font command used in text mode when the surrounding text is sans serif.
- `texttt=`⟨*macro*⟩ The font command used in text mode when the surrounding text is fixed width.

11.2 Spacing and separators

The spacings used between items are all user-definable. This is also true for the separators used for decimals, *etc.*.

- `unitsep=`⟨*choice*`,`*literal*⟩ This defines the separation of different unit symbols. The ⟨*list*⟩ takes values **`thin`**, `medium` (alias `med`), `thick` (all maths spacings), `space` (a full space), `cdot` (a centred dot) and `times`. ⟨*literal*⟩ values are typeset in maths mode.
- `unitspace=`⟨*choice*`,`*literal*⟩ The spacing represented by an explicit hard space (`~`) inside a unit macro. Takes the same list as `valuesep`.
- `valuesep=`⟨*choice*`,`*literal*⟩ Defines the separation between a value and the associated unit. Valid ⟨*list*⟩ values are **`thin`**, `medium` (alias `med`), `thick`, `space` and `none`.
- `digitsep=`⟨*choice*`,`*literal*⟩ The separation (if any) between groups of digits in large numbers. Valid ⟨*list*⟩ values are **`thin`**, `medium` (alias `med`), `thick`, `space`, `comma`, `fullstop` (aliases `stop` and `period`) and `none`.
- `decimalsign=`⟨*choice*`,`*literal*⟩ The decimal sign, either `comma` or **`fullstop`** (also aliased as `stop` and `period`).

- `anglesep=⟨choice,literal⟩` The separator between degrees, minutes and seconds in an angle. The options are `thin`, `medium` (alias `med`), `thick` and **`none`**.

11.3 Number formatting

There are two types of option for numbers. The first set are concerned with parsing numbers, and are very similar to the settings in numprint. These all begin `num`, and take literal values. Notice that the literals are *not* separated in any way in the input.

- `numlist=⟨literal⟩` The characters which are numbers: **`01234567890`**.
- `numdecimal=⟨literal⟩` Decimal markers: **`.,`**
- `numexp=⟨literal⟩` Exponent markers: **`edED`**
- `numgobble=⟨literal⟩` Characters to be gobbled when processing numbers: no default
- `numsign=⟨literal⟩` Signs (which must be at the start of a number): **`+-\pm\mp`**
- `numextra=⟨literal⟩` "Extra" characters, to be carried through directly to the output: **`()`**

The second type of option for numbers controls the output.

- `addsign=⟨choice⟩` Sets whether a sign is added to numbers without an explicit sign given. Valid choices are `mantissa` (or `mant`), `exponent` (or `exp`), `both` (or `all`) and **`none`**.The option will also act as a Boolean, taking `true` and optfalse, with `addsign` alone equal to giving the `true` (= `all`) value.
- `sign=⟨choice,literal⟩` The sign used by the above. Choices are **`plus`**, `minus`, `pm` and `mp` ($\pm$ and $\mp$, respectively). The sign will always be typeset in maths mode.
- `sepfour=⟨bool⟩` When separating out numbers (using `digitsep`), four-digit numbers can be skipped. This is the default.
- `expproduct=⟨choice,literal⟩` The symbol used to indicate a product for exponents (*i.e.* the $\times$ in 2×10^2). The choices are **`times`** and `cdot`.
- `exppower=⟨choice,literal⟩` Slightly esoterically, the power used for exponents can be altered. The "choice" list here only recognises `ten`; anything else is used literally.
- `padnumber=⟨choice⟩` This sets where zeros are added. The choices are **`leading`** (a leading zero added to `.1`), `trailing` (converts `1.` to 1.0), `all` (`leading` and `trailing`, also available as `both`) and `none` (no zeros added). The option will also act as a Boolean, taking `true` and optfalse, with `padnumber` alone equal to giving the `true` (= `all`) value.

11.4 Angle formatting

The angle formatter uses `\num` to format numbers; any options for numbers are therefore applicable here. When typesetting an angle using `\ang`, the following extra option is also relevant.

- `padangle=`⟨*choice*⟩ Determines whether small and large angles are padded. The choices are `none` (no additional zeros are added), **`small`** (angles with no degrees have 0° added), `large` (angles with no seconds have 0″ added) and `all` (`small` and `large` combined). The option also recognises `true` and `false` as choices, which are equal to `all` and `none`, respectively. If no value is given, `padangle` acts a Boolean choice.

- `astroang=`⟨*bool*⟩ Astronomers place the signs for angles over decimal signs; this is handled here.

11.5 Tabular material

The formatting of data in `s` columns is controlled by a single package option.

- `tabformat=`⟨*number*⟩ The number here determines how to centre decimal numbers in a column. If `number` is zero or negative, then the decimal marker is placed at the centre of the column with the number symmetrically placed around it. If `number` is positive, it is interpreted `\meta{pre}.\meta{post}`, where ⟨*pre*⟩ is the number of digits before the decimal marker and ⟨*post*⟩ is the number after. Appropriate space is reserved to centre a number of total length ⟨*pre*⟩ + ⟨*post*⟩ (plus the decimal marker). If the digits supplied are too long, overfull boxes will result. If only ⟨*pre*⟩ is given, an equal amount of space is reserved before and after the decimal marker, and the number is typeset flush right.

11.6 Units

The output of units (as opposed to the numerical argument of the unit) takes only a few options.

- `xspace=`⟨*bool*⟩ Determines whether to use xspace at the end of unit macros when not given inside `\SI`, for example `10\metre away` will give "10 m away" with `xspace` turned on, but "10 maway" with it turned off.

- `per=`⟨*choice*⟩ Affects how `\per` is interpreted in units. The options available are **`reciprocal`** (also available as `rp` and `power`), `slash` and `fraction` (or `frac`).

- `fraction=`⟨*choice*⟩ When using `per=frac`, further control of the appearance of the fraction is provided. The options available are `frac` (uses LaTeX `\frac` operation), `nice` (also available as `nicefrac`; uses a nicefrac-like system), `ugly` (also `uglyfrac`; the same as loading nicefrac with the `ugly` option: uses `\frac` for material in maths mode and a slash for

material in text mode) and `sfrac` (uses the `\sfrac` macro from the xfrac package.[22]

- `denlbrac=`⟨*literal*⟩ and `denrbrac=`⟨*literal*⟩ When using `per=slash`, using two or more units in the denominator gives an ambiguous fraction. The package therefore adds `denlbrac` and `denrbrac` in such cases.
- `prefix=`⟨*choice*⟩ Controls how prefixes to units are handled, with options **`symbol`** (or `letter`) and `number` (or `power`).
- `prefixpower=`⟨*choice*, *literal*⟩ and `prefixproduct=`⟨*choice*⟩ Works in the same way as the general exponent equivalents, but only for prefix modifiers.

11.7 Symbols

User access to control the symbols used for µ, Ω, Å, ° and °C 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.

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

When si is loaded, it can check for the presence of the textcomp and upgreek packages, to provide better symbols for certain items. To prevent this, us the `redefsymbols=false` option.

The eV symbol requires some fine-tuning, and so has two options of its own.

[22] xfrac is part of the experimental system for LaTeX3. As it requires a number of additional packages to work, si 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`.

- `eVcorra=⟨length⟩` The correction applied to the gap between "e" and "V" of the unit. The default is `0.3ex`.
- `eVcorrb=⟨length⟩` The correction applied to the gap between "V" of the unit and whatever follows. The default is `0ex`; a change is needed for example in `\unitsym[per=slash]{\electronvolt\per\metre}`, which gives eV/m by default, but eV⁄m by setting `eVcorrb=0.7ex`. The value needed will depend on the use of the unit and the font metrics used.

11.8 Package control

These macros alter the overall behaviour of the package.

- `load=⟨list⟩` Sets which additional configuration files are loaded. These all have names of the form `si-⟨option⟩.cfg`, where ⟨*option*⟩ should be given in the `load` list. The package recognises the `load=default` option, which is expanded to the standard list of loaded files. This is to allow easy addition of one or more files without needing to know the default list.
- `noload=⟨list⟩` Excludes files from the above from being loaded, so that a single file can be omitted without needing to type a long list of those to be used.
- `alsoload=⟨list⟩` Adds an item to the list to be loaded, without needing to specify all of the existing list.
- `log=⟨choice⟩` Sets the amount of information written to the log by si. The ⟨*list*⟩ is `none`, `errors`, **`normal`** and `debug`. The last option is also available as a Boolean, and gives *lots* of information in the log.
- `emulate=⟨choice⟩` Causes si to emulate the given package. The ⟨*list*⟩ takes values `SIunits`, `sistyle`, `units` and `unitsdef`. This option can only be used when loading the package.

11.9 Back-compatibility options

As well as the options outlined above, at load time a number of options are available to allow si to be used as a direct replacement for other unit-management packages. These are the same options as are available in SIunits, sistyle, units, and unitsdef. Using a legacy option will cause the package to load the appropriate emulation code

12 Emulation of other packages

si has been designed as a replacement for SIunits, sistyle, units and unitsdef. It therefore provides options a hooks to reproduce the functions of all of these packages. In this way, si should be usable as a straight replacement for the older packages. All of the user macros of ⟨*package*⟩ are (hopefully) available when using the `emulate=⟨package⟩`.[23] This means for example that the `\num` macro

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

takes an optional star when emulating sistyle. However, there are some points that should be remembered. In particular, si validates numerical input, meaning that places where a number is expected in the older packages *require* a number when emulated by si.

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 si. 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 si, the csnumprint macro could be provided using the following code.

$-123\,456$ $-123\,456\,\mathrm{N/mm^2}$

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

si 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 an 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.

13 Tricks and known issues

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.

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.[24] The use of the `\mathnormal` font may get around this issue.

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

```
\num[numextra=\pi]{4\pi e-7}\\
\num[numextra=\pi,mathnumdefault=mathnormal]{4\pi e-7}\\
```

On the other hand, you may want to use text mode. There, `\ensuremath` is needed.

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

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

There are several potential pitfalls in this area; experimentation may well be needed.[25]

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

[25] Any suggestions for the code that runs this are welcome; the issue is how to deal with active characters in the input while not expanding macros.

14 Reporting a problem

si 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`, by running `si.dtx` through (pdf)LaTeX.

```
\listfiles
\documentclass{article}
% Add other packages here.
% Add options need for si package, retain the debug option.
\usepackage[debug]{si}
\begin{document}
This is the bug test-case document for the \textsf{si}
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}
```

15 Acknowledgements

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 (Slunits), Patrick Happel (unitsdef), Axel Reichert (units) and Harald Harders (numprint). Philip Lehmann, Will Robertson and Heiko Oberdiek deserve much credit for demonstrating LaTeX coding best practice. Thanks to the various contributors of ideas for the package: Donald Arseneau, Michele Dondi, Paul Gans, Ben Morrow, Lan Thuy Pham, Stefan Pinnow, Allan Ristow and Patrick Steegstra.

Part III
Correct application of (SI) units

TO DO!

Part IV
Implementation

16 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. However, internal package macros may behave differently; if the user has redefined internal macros, then compatibility may be impaired.

The code used here uses LaTeX rather than TeX commands where possible.[26] 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.

16.1 Setup code

As always, the package starts with identification. A warning is then printed about possible changes.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{si}%
  [2008/02/20 v.06a A comprehensive (SI) units package]
\PackageInfo{si}
  {This package is experimental. \MessageBreak The interface and
   functionality is subject to review \MessageBreak and may be changed
   in later releases}
```

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

```
\begingroup
  \@ifundefined{eTeXversion}
    {\PackageError{si}
       {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}
```

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. xkeyval has to be at least v2.5, as earlier versions do not have the correct macros available. As this will lead to serious errors later, si aborts if xkeyval is too old.

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

```
\RequirePackage{xkeyval}
\@ifpackagelater{xkeyval}{2005/05/07}
  {}
  {\PackageError{si}
     {xkeyval >= 2.5 required}
     {si requires the 'xkeyval' package, version 2.5 or
       later.\MessageBreak The version loaded is:
       '\@nameuse{ver@xkeyval.sty}'.\MessageBreak
        This is a fatal error; the package will abort.}%
   \endinput}
\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@packagecheck
\si@tempa

As si 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. Also notice that unitsdef must be tested before units, so that users of the former get an intelligible message.

```
\newcommand*{\si@packagecheck}{%
  \begingroup
  \@for\si@tempa:=SIunits,siunits,sistyle,SIstyle,unitsdef\do{%
    \@ifpackageloaded{\si@tempa}
      {\PackageError{si}
         {Package '\si@tempa' incompatible}
         {The '\si@tempa' package and 'si' are
          incompatible.\MessageBreak Use the
          'emulate=\si@tempa' package option when loading si.}}
      {}}
```

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

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

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

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

\si@ifdefinable 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.
\si@ifdefinable{⟨*macro*⟩}

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

\si@addtolist It is quite useful to be able to add to a comma-separated list of expandable items.
\si@addtolist{⟨*macro*⟩}{⟨*items*⟩}

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

\si@addtocsname A second item to add to a is macro.
\si@temptoks \si@addtomacro{⟨*macro*⟩}{⟨*tokens*⟩}
\si@addtocsname{⟨*csname*⟩}{⟨*tokens*⟩}

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

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

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

16.2 Logging

\ifsi@debug To control logging, some new switches are declared.
\ifsi@logmin

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

\ifsi@lognone

```
\newif\ifsi@lognone
```

\si@log@err Some handy re-usable macros are defined here. These all take names beginning
\si@log@warn These pop up in various places. First errors, warnings and information are
\si@log@inf handled. Package options are used to control how much output is given.
\si@log@err{⟨*error*⟩}{⟨*explanation*⟩}
\si@log@warn{⟨*warning*⟩}
\si@log@inf{⟨*information*⟩}

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

`\si@log@debug` The debug macro only gives output if the appropriate package option is set.
`\si@log@debug{`⟨*debug-information*⟩`}`

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

16.3 Option handling

`\sisetup` To allow modification of options at run time, a setup macro is provided. The run of strange tests are to prevent problems in arrays and the like.
`\sisetup{`⟨*keyval-options*⟩`}`

```
\newcommand*{\sisetup}[1]{%
  \iffalse{\fi\ifnum0='}\fi
  \setkeys[si]{opt}{#1}%
  \ifnum0='{\fi\iffalse}\fi}
```

`\si@opt@key` 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.
`\si@opt@key{`⟨*keyname*⟩`}{`⟨*code*⟩`}`

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

`\si@opt@cmdkey` The command versions of the above.
`\si@opt@cmdkeys` `\si@opt@cmdkey[`⟨*default*⟩`]{`⟨*keyname*⟩`}{`⟨*function*⟩`}`
`\si@opt@cmdkeys[`⟨*default*⟩`]{`⟨*keynames*⟩`}`

```
\newcommand*{\si@opt@cmdkey}[3][]{%
```

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

`\si@opt@boolkey` Keys which only take switch values; anything other than `true` or `false` will generate a warning from xkeyval. `\si@opt@boolkey[`⟨*optional-processing*⟩`]{`⟨*keyname*⟩`}`

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

`\si@opt@choicekey` 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.
`\si@opt@choicekey[`⟨*default*⟩`]{`⟨*keyname*⟩`}{`⟨*choices*⟩`}{`⟨*in-list*⟩`}{`⟨*not-in-list*⟩`}`

```
\newcommand*{\si@opt@choicekey}[5][]{%
  \define@choicekey*+[si]{opt}{#2}[\si@tempa]{#3}[#1]
    {#4\si@log@debug{Option #2 set to ##1}}
    {#5\si@log@debug{Option #2 set to ##1}}}
```

`\si@opt@xchoicekey` 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.
`\si@opt@xchoicekey{`⟨*keyname*⟩`}{`⟨*choices*⟩`}{`⟨*initial*⟩`}`

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

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.

```
    {\expandafter\renewcommand\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.

```
    {\expandafter\renewcommand\expandafter*%
      \csname si@#1\endcsname{##1}}
```

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

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

`\si@opt@compatkey` 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.
`\si@opt@compatkey{`⟨*package*⟩`}{`⟨*keyname*⟩`}`.

```
\newcommand*{\si@opt@compatkey}[2]{%
  \define@boolkey[si]{opt}[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 si package}}}
```

\si@opt@disablekey The ability to disable a key with a meaningful message is a must; the warning will come from si, and not from xkeyval

\si@opt@disablekey{⟨*keyname*⟩}{⟨*warning*⟩}

```
\newcommand*{\si@opt@disablekey}[2]{%
  \key@ifundefined[si]{opt}{#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}
```

\si@tempa \si@tempb 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}
```

The option has not been recognised: give a warning (if appropriate).

```
   {\si@log@warn{Unrecognised value `#1' for option log}}
```

A quick method to set `log=debug`.

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

\si@emulate The `emulate` option is used for back-compatibility mode; if the keyword is given with no value, emulation of SIunits is assumed.

```
\newcommand*{\si@emulate}{}
\si@opt@choicekey[SIunits]{emulate}
  {SIunits,sistyle,numprint,units,unitsdef}
  {\si@log@debug{Found emulation request for #1 package}%
   \si@addtolist{\si@emulate}{#1}}
  {\si@log@warn{Unknown value `#1' for option emulate
   \MessageBreak No emulation will occur}}
```

The `emulate` option is no longer valid once the package has been loaded.

```
\AtEndOfPackage{%
```

```
  \si@opt@disablekey{emulate}
    {emulate option only available when\MessageBreak
     loading package}}
```

\si@unitsep
\si@unitspace
\si@valuesep

The two `...space` 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`[27] and `none`;[28] other entries will be treated as custom spaces.

```
\si@opt@xchoicekey{unitsep}{thin,med,medium,thick,space,none,cdot,
  times}{thin}
\si@opt@xchoicekey{unitspace}{space,thin,med,medium,thick,none}
  {thin}
\si@opt@xchoicekey{valuesep}{space,thin,med,medium,thick,none}
  {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}{thin}
```

\si@decimalsign

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{decimalsign}{comma,stop,fullstop,period,cdot}
  {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}{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@textmode

The output of the package can be typeset using either text or maths mode fonts. This is controlled using the `mode` option and the `\ifsi@textmode` switch.

```
\newif\ifsi@textmode
\si@opt@boolkey{textmode}
\si@opt@choicekey{mode}{math,maths,text}
```

\si@tempa
\si@tempb

`\si@tempb` used to for the expansion tests. The default is `none`, as the choice key will not allow other values to get here.

```
  {\si@textmodefalse
   \renewcommand*{\si@tempb}{text}%
   \ifx\si@tempa\si@tempb
     \si@textmodetrue
   \fi}
  {\si@log@warn{Unknown value `#1' for option mode}}
```

[27] only valid for `unitsep`.
[28] Only valid for `valuesep`.

\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`).

\si@tempa
\si@tempb

```
\newif\ifsi@inlinebtext
\si@opt@choicekey{inlinebold}{text,maths,math}
  {\si@inlinebtextfalse
   \renewcommand*{\si@tempb}{text}%
   \ifx\si@tempa\si@tempb
     \si@inlinebtexttrue
   \fi}
  {\si@log@warn{Unknown value '#1' for option inlinebold}}
```

\ifsi@obeyitalic Italic is slightly different to bold, as there is no convenient switch for maths.

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

\si@mathsdefault 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.

\si@mathsrm
\si@mathssf
\si@mathstt

```
\si@opt@cmdkeys{mathsdefault,mathsrm,mathssf,mathstt}
```

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

```
\si@opt@key{mathdefault}{\sisetup{mathsdefault=#1}}
\si@opt@key{mathrm}{\sisetup{mathsrm=#1}}
\si@opt@key{mathsf}{\sisetup{mathssf=#1}}
\si@opt@key{mathtt}{\sisetup{mathstt=#1}}
```

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

\si@textrm
\si@textsf
\si@texttt

```
\si@opt@cmdkeys{textdefault,textrm,textsf,texttt}
```

\si@mathnumdefault To allow numbers to be set in a different font to text, additional options are set up.

\si@textnumdefault

```
\si@opt@cmdkeys{mathnumdefault,textnumdefault}
\si@opt@key{mathsnumdefault}{\sisetup{mathnumdefault=#1}}
```

\si@numlist 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`.

\si@numdecimal
\si@numexp
\si@numgobble
\si@numsign
\si@numextra

```
\si@opt@cmdkeys{numlist,numdecimal,numexp,numgobble,numsign,numextra}
```

\si@numvalid 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.

```
\newcommand*{\si@numvalid}{\si@numgobble\si@numexp\si@numsign%
  \si@numdecimal\si@numlist\si@numextra}
```

\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}
```

\si@expproduct The marker for multiplication in exponential numbers is set up.

```
\si@opt@xchoicekey{expproduct}{times,cdot}{times}
```

\si@exppower In the same area, the power for exponents is variable. Only one choice is given.

```
\si@opt@xchoicekey{exppower}{ten}{ten}
```

\si@prefixproduct The marker for multiplication in prefixes.

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

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

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

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

```
\newif\ifsi@prefixnum
\si@opt@choicekey{prefix}{symbol,letter,power,number}
```

\si@tempa \si@tempb used to for the expansion tests. The default is `none`, as the choice \si@tempb key will not allow other values to get here.

```
  {\si@prefixnumfalse
   \renewcommand*{\si@tempb}{power}%
   \ifx\si@tempa\si@tempb
     \si@prefixnumtrue
   \fi
   \renewcommand*{\si@tempb}{number}%
   \ifx\si@tempa\si@tempb
     \si@prefixnumtrue
   \fi}
  {\si@log@warn{Unknown value `#1' for option prefix}}
```

\ifsi@num@padlead A setting is needed to indicate when to add zeros to decimal numbers, either \ifsi@num@padtrail 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@tempa \si@tempb is used to for the expansion tests. The default is `none`, as the choice \si@tempb key will not allow other values to get here.

```
  {\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@log@warn{Unknown value `#1' for option padnumber}}
```

`\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}
```

`\si@tempa` The option is now processed.

`\si@tempb`

```
  {\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}
  {\si@log@warn{Unknown value `#1' for option addsign}}
```

`\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@tempa` `\si@tempb` `\si@tempb` is used to for the expansion tests. The default is `none`, as the choice key will not allow other values to get here.

```
  {\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}
  {\si@log@warn{Unknown value `#1' for option padangle}}
```

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

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

\si@tabformat The formatting of numbers in tables is handled by a dcolumn-like system. For that, a single option is needed to control the centring of data in the table.

```
\si@opt@cmdkey[-1]{tabformat}{}
```

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

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

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

\ifsi@slash

```
\newif\ifsi@slash
\newif\ifsi@frac
\si@opt@choicekey[reciprocal]{per}
  {reciprocal,rp,power,slash,frac,fraction}
```

\si@tempa The usual value testing, with a default to use reciprocal powers.

\si@tempb

```
  {\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@log@warn{Unknown value `#1' for option per}}
```

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

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

\si@denrbrac Macros for the right and left brackets added to potentially-ambiguous denominators.

\si@denlbrac

```
\si@opt@cmdkeys{denrbrac,denlbrac}
```

\si@tempa In the case of fractional handling of the \per operator, further refinement is available.

\si@tempb

```
\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@log@warn{Unknown value '#1' for option fraction}}
```

\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 si}
  \si@opt@disablekey{also}
    {Configuration files can only be used\MessageBreak
     when loading si}
  \si@opt@disablekey{noload}
    {Configuration files can only be used\MessageBreak
     when loading si}}
```

\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}
\DeclareFontFamily{OML}{eur}{\skewchar\font127}
\DeclareFontShape{OML}{eur}{m}{n}%
  {<5> <6> <7> <8> <9> gen * eurm %
```

```
    <10><10.95><12><14.4><17.28><20.74><24.88>eurm10}{}
\DeclareSymbolFont{greek}{OML}{eur}{m}{n}
\DeclareMathSymbol{\si@mathsmu}{\mathord}{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 C}
\newcommand*{\si@mathscelsius}{\si@mathsdegree\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 si}}
```

`\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@loadlocales` `\si@loctolang` Handling typographic conventions needs three keys. `locale` is used to set the locale, whereas `\loadlocales` reads in the definitions at package load time.

```
\si@opt@cmdkeys{loadlocales,loctolang}
\si@opt@cmdkey{locale}{%
  \sisetup{loadlocales={#1}}%
  \AtEndOfPackage{\si@loc@set{#1}}}
\AtBeginDocument{%
  \si@opt@disablekey{loadlocales}
    {Locale files can only be loaded\MessageBreak
     in the preamble}
  \si@opt@disablekey{loctolang}
```

```
    {Locale files can only be loaded\MessageBreak
     in the preamble}
  \si@opt@cmdkey{locale}{\si@loc@set{#1}}}
```

16.4 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.

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

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

The second set are more general functionality.

```
\si@opt@compatkey{unitsdef}{LITER}
\si@opt@compatkey{unitsdef}{liter}
\si@opt@compatkey{unitsdef}{noxspace}
\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!

```
\si@opt@compatkey{unitsdef}{noabbr}
\si@opt@compatkey{unitsdef}{noampereageabbr}
\si@opt@compatkey{unitsdef}{nofrequncyabbr}
\si@opt@compatkey{unitsdef}{nomolabbr}
\si@opt@compatkey{unitsdef}{novoltageabbr}
\si@opt@compatkey{unitsdef}{novolumeabbr}
\si@opt@compatkey{unitsdef}{noweightabbr}
\si@opt@compatkey{unitsdef}{noenergyabbr}
\si@opt@compatkey{unitsdef}{nolengthabbr}
\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.

```
\si@opt@compatkey{SIunits}{cdot}
\si@opt@compatkey{SIunits}{thickspace}
\si@opt@compatkey{SIunits}{mediumspace}
\si@opt@compatkey{SIunits}{thinspace}
\si@opt@compatkey{SIunits}{thickqspace}
\si@opt@compatkey{SIunits}{mediumqspace}
\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.

```
\si@opt@compatkey{SIunits}{amssymb}
```

```
\si@opt@compatkey{SIunits}{squaren}
\si@opt@compatkey{SIunits}{pstricks}
\si@opt@compatkey{SIunits}{Gray}
\si@opt@compatkey{SIunits}{italian}
```

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

The miscellaneous options.

```
\si@opt@compatkey{SIunits}{textstyle}
\si@opt@compatkey{SIunits}{binary}
\si@opt@compatkey{SIunits}{noams}
\si@opt@compatkey{SIunits}{derivedinbase}
\si@opt@compatkey{SIunits}{derived}
```

16.5 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

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}
```

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

Signs for numbers are needed.

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

\si@fix@two
\si@fix@ten

The literals “2” and “10” are needed for exponents.

```
\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}{}
```

16.6 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\DeclareRobustCommand\expandafter*\expandafter{%
    \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
\si@sym@ringA
\si@sym@mu
\si@sym@degree
\si@sym@minute
\si@sym@second
\si@sym@celsius
The various symbols are now declared.

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

si@tempa The issue of redefinition of symbols now arises. si 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.

```
       \long\def\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@mathsmu}{\upmu}%
       \renewcommand*{\si@mathsOmega}{\Upomega}}{}
  \fi}
```

16.7 Handling fractions

\si@frac
\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{\frac{\expandafter\si@unt@out\expandafter{#1}}%
    {\expandafter\si@unt@out\expandafter{#2}}}}
\let\si@frac\si@frc@frac
\newcommand*{\si@frc@slash}[2]{%
    \expandafter\si@unt@out\expandafter{#1}%
    \si@out@text{\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.[29]

```
\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`

The maths mode system.
`\si@frc@mathsnf{`⟨*numerator*⟩`}{`⟨*denominator*⟩`}`

```
\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}}}}%
```

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

```
      {\raisebox{\si@frc@suplen}%
        {\ensuremath{\scriptscriptstyle{#1}}}}%
      {\raisebox{\si@frc@ssuplen}%
        {\ensuremath{\scriptscriptstyle{#1}}}}%
    \mkern-2mu/\mkern-1mu%
    \bgroup
      \mathchoice
        {\scriptstyle}%
        {\scriptstyle}%
        {\scriptscriptstyle}%
        {\scriptscriptstyle}%
      {#2}%
    \egroup
  \endgroup}
```

`\si@frc@textnf` A stripped down version of the nicefrac system for text mode.
`\si@frc@textnf{`⟨*numerator*⟩`}{`⟨*denominator*⟩`}`

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

`\si@frc@ugly` `\si@tempa` `\si@tempb` The `\si@frc@ugly` macro is needed to emulate the `ugly` option in units, where output depends on the current mode.
`\si@frc@ugly{`⟨*numerator*⟩`}`

```
\newcommand*{\si@frc@ugly}[1]{%
  \def\si@tempa{#1}%
  \ifmmode
    \expandafter\si@frc@frac%
  \else
    \def\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@tempbox=\hbox{\ensuremath{\si@valuesep}}%
      \hskip-\wd\si@tempbox%
      \def\si@tempa{}%
    \fi
    \expandafter\si@frc@slash%
  \fi
  {\si@tempa}}
```

16.8 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 si interface.

\si@fam@getmfam The font families in use in the document are needed.

\si@fam@sf
\si@fam@tt

```
\newcommand*{\si@fam@getmfam}{%
  \sbox{0}{$%
    \@ifundefined{mathsf}
      {\si@log@debug{No mathsf family found}%
       \global\chardef\si@fam@sf=99}%
      {\mathsf{\global\chardef\si@fam@sf=\fam}}%
    \@ifundefined{mathtt}
      {\si@log@debug{No mathtt family found}%
       \global\chardef\si@fam@tt=99}%
      {\mathtt{\global\chardef\si@fam@tt=\fam}}%
    $}}
\AtBeginDocument{\si@fam@getmfam}
```

\si@fam@ifbtext
\si@fam@ifbmaths
\si@tempa

These tests check for bold in text and maths mode, respectively.
\si@fam@ifbtext{⟨*code*⟩}
\si@fam@ifbmaths{⟨*code*⟩}

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

This test check for italic or slanted text in text mode, by negation (upright text is n).
\si@fam@ifitext{⟨*code*⟩}

```
\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}
```

`\ifsi@fam@set` A marker is set up to check if font-matching has been taken place.

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

`\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.

```
\newcommand*{\si@fam@set}{%
  \si@fam@settrue
```

`\si@tempa` `\si@tempb` The temporary macros are needed for the `\ifx` tests.

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

The surrounding font family is only tested if matching is requested.

```
  \ifsi@obeyfamily
    \si@log@debug{Font detection: checking font}%
```

`\si@fam@maths` `\si@fam@text` Next, checks are needed for maths *versus* text mode, and if in maths mode, whether this is inline or display. Once that is done, font families can be tested.

```
    \ifmmode
      \ifdim\displaywidth>0pt\relax
        \si@log@debug{Font detection: display maths}%
        \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@mathsdefault\endcsname
            \expandafter\let\expandafter\si@fam@text
              \csname\si@textdefault\endcsname
          \fi
        \fi
```

Inline maths is now handled.

```
      \else
        \si@log@debug{Font detection: inline maths}%
        \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@mathsdefault\endcsname
            \expandafter\let\expandafter\si@fam@text
              \csname\si@textdefault\endcsname
          \fi
        \fi
      \fi
```

Not in maths mode, so the text mode checks are carried out.

```
    \else
      \si@log@debug{Font detection: text}%
      \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@mathsdefault\endcsname
          \expandafter\let\expandafter\si@fam@text
            \csname\si@textdefault\endcsname
        \fi
      \fi
    \fi
```

If the local font is not to be matched, setting the fonts is rather less complex.

```
  \else
    \si@log@debug{Font detection: inactive}%
    \expandafter\let\expandafter\si@fam@maths
      \csname\si@mathsdefault\endcsname
    \expandafter\let\expandafter\si@fam@text
      \csname\si@textdefault\endcsname
  \fi
```

`\si@fam@bold` `\si@fam@setbold` 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 commands are set up with `\def`, as nested calls can occur.

```
\def\si@fam@bold{\unboldmath\mdseries}%
\def\si@fam@setbold{\boldmath\bfseries}%
\ifsi@obeybold
  \si@log@debug{Weight detection: checking weight}%
  \ifmmode
```

Display maths.

```
    \ifdim\displaywidth>0pt\relax
      \si@fam@ifbmaths
        {\let\si@fam@bold\si@fam@setbold
         \si@log@debug{Weight detection: bold weight}}%
```

Inline maths.

```
    \else
      \si@fam@ifbinline
        {\let\si@fam@bold\si@fam@setbold
         \si@log@debug{Weight detection: bold weight}}%
    \fi
```

Text mode.

```
  \else
    \si@fam@ifbtext%
      {\let\si@fam@bold\si@fam@setbold
       \si@log@debug{Weight detection: bold weight}}%
  \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}}%
  \ifsi@textmode\else
    \si@log@inf{maths mode - obeyitalic inactive}%
  \fi
\fi}
```

16.9 Formatting numbers

`\num` 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.

`\num[`⟨*options*⟩`]{`⟨*num*⟩`}`

```
\DeclareRobustCommand*{\num}[2][]{%
```

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

`\si@num` 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.

`\si@num{`⟨*num*⟩`}`

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

`\si@tempa` 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. A bit of trickery is needed to avoid getting an extraneous space introduced here by `\scantokens`, hence the use of `\@empty`.

```
  \begingroup
    \makeatletter%
    \@makeother{\,}%
    \@makeother{\.}%
    \@makeother{\+}%
    \@makeother{\-}%
    \def~{}%
    \def\,{}%
    \catcode`\~=\active\relax
    \catcode`\^=\active\relax
    \scantokens{\si@num@xdef\si@tempa{#1}\@empty}%
  \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 no argument, nothing happens; everything else in the `\si@num` macro occurs only if the argument is filled.

```
  \si@ifnotmtarg{\si@tempa}{%
```

The input is now validated. Further processing takes place a little later.

```
    \expandafter\si@num@valid\si@tempa\@empty\@empty%
```

If the input is valid, the input is passed to the number formatter.

```
  \ifsi@switch
    \expandafter\si@num@format\expandafter\si@tempa%
  \else
```

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

```
    {#1}%
  \fi}}
```

`\si@num@xdef` When carrying out the `\edef` used to fully-expand a number, `\,` and `~` are deactivated, so that macros do not end up in the number. By using a separate

macro, \scantokens is easier. \protected@xdef is not used here, as the argument given should only contain single (processable) characters or macros that expand to the same, not other macros or characters.
\si@num@xdef⟨*macro*⟩⟨*num*⟩\@empty

```
\def\si@num@xdef#1#2\@empty{\xdef#1{#2}}
```

\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.
\si@num@valid⟨*char*⟩⟨*chars*⟩\@empty

```
\def\si@num@valid#1#2\@empty{%
  \si@num@ifchr{#1}{\si@numvalid}{%
    \ifx\@empty#2\@empty\else
      \si@num@valid#2\@empty\@empty\@empty%
    \fi
    \si@switchtrue}%
```

If an invalid character has been picked up, the whole parsing system has to bail out.

```
    {\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
          macro\MessageBreak (or derivative such as an `s' column)}%
     \si@switchfalse}}
```

\si@num@ifchr A test is needed to check one string only contains characters from a second. The main macro sets up for the recursion system below.
\si@tempa
\si@tempb \si@num@ifchr{⟨*test-chars*⟩}{⟨*valid-chars*⟩}

```
\newcommand*{\si@num@ifchr}[2]{%
  \begingroup
    \si@switchfalse
    \def\si@tempa{#1}%
    \edef\si@tempb{#2}%
```

Now the test can occur for the initial comparison string.

```
    \expandafter\si@num@chrstr\si@tempb\@empty\@empty%
```

By ending the group inside the \if, \global is avoided, and the switch can be used for other jobs.

```
  \ifsi@switch%
    \endgroup\expandafter\@firstoftwo%
  \else
    \endgroup\expandafter\@secondoftwo%
  \fi}
```

\si@num@chrstr The second part of the comparison macro does the actual work. This takes one character of the string of valid input at a time, and compares it to the single character in \si@tempa.
\si@num@chrstr⟨*char*⟩⟨*chars*⟩\@empty

```
\def\si@num@chrstr#1#2\@empty{%
```

\si@tempb \si@tempc is used to hold the single character to check against \si@tempa,
\si@tempc while \si@tempb stores the remaining characters to be compared.

```
  \def\si@tempc{#1}%
  \edef\si@tempb{#2}%
  \ifx\si@tempa\si@tempc
    \si@switchtrue
  \else
```

If `\si@tempb` is `\@empty`, then the recursion has bottomed-out, and all of the comparisons are done. If not, go round again.

```
    \ifx\@empty\si@tempb\@empty\else
      \si@num@chrstr#2\@empty%
    \fi
  \fi}
```

`\si@num@exp` `\si@num@mant` `\si@num@expout` `\si@num@mantout` `\si@num@out`

Various storage macros are needed.

```
\newcommand*{\si@num@exp}{}
\newcommand*{\si@num@mant}{}
\newcommand*{\si@num@expout}{}
\newcommand*{\si@num@mantout}{}
\newcommand*{\si@num@out}{}
```

`\si@num@format` `\si@num@arg`

The number processor starts by saving `#1` (odd things happen otherwise), and locally clearing the stacks.
`\si@num@format{`⟨*num*⟩`}`

```
\newcommand*{\si@num@format}[1]{%
  \protected@edef\si@num@arg{#1}%
  \def\si@num@exp{}%
  \def\si@num@mant{}%
  \def\si@num@expout{}%
  \def\si@num@mantout{}%
  \si@log@debug{Formatting number `\si@num@arg'}%
  \si@switchfalse
```

The input is split into an mantissa and an exponent.

```
  \expandafter\si@num@mantexp\si@num@arg\@empty\@empty%
```

The mantissa and exponent are now processed separately. Firstly, a sign is tested for. If there is one, it is added to `\si@num@out`, while the rest of the number is returned "as is" in `\si@num@...`.

```
  \si@num@sign{mant}%
```

`\si@num@out`

To allow for the case where the mantissa is only a sign, but the exponent contains a number, the output is initially defined to whatever is in `\si@num@mantout`. This will change if there is a number in `\si@num@mant`.

```
  \protected@edef\si@num@out{\si@num@mantout}%
  \ifx\@empty\si@num@mant\@empty\else
     \si@num@digits{mant}%
     \protected@edef\si@num@out{\si@num@mantout}%
  \fi
  \si@num@sign{exp}%
  \ifx\@empty\si@num@exp\@empty\else
    \si@num@digits{exp}%
```

Allowance is made for the possibility of negative exponential-only numbers. Precautions are taken for the multiply sign (which is always in maths mode), and superscripts. `\textsuperscript` is used here, as this will work in text or maths mode in the output routine.

```
    \ifx\@empty\si@num@mant\@empty\else
      \protected@edef\si@num@out%
        {\si@num@out\noexpand\ensuremath{{}\noexpand\si@expproduct{}}}%
    \fi
    \protected@edef\si@num@out%
      {\si@num@out\si@exppower%
        \noexpand\textsuperscript{\si@num@expout}}%
  \fi
```

If there is nothing in either number macro, then something is wrong.

```
  \ifx\@empty\si@num@mant\@empty
    \ifx\@empty\si@num@exp\@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@numlist')?}%
      \renewcommand*\si@num@out{}%
    \fi
  \fi%
```

With everything done, the result is output.

```
  \si@num@out}
```

`\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.

`\si@num@mantexp`⟨*char*⟩⟨*chars*⟩`\@empty`

```
\def\si@num@mantexp#1#2\@empty{%
  \si@num@ifchr{#1}{\si@numgobble}{}{%
    \si@num@ifchr{#1}{\si@numexp}%
      {\ifsi@switch
         \si@log@err{Duplicate exponent marker found}%
           {Only a single exponent character (from the list
             `\si@numexp')\MessageBreak may occur in a
             numerical argument}%
      \fi
      \si@switchtrue
      \si@log@debug{Exponent marker `#1' found in `\si@num@arg'}}%
```

When building up the mantissa and exponent, everything must be expandable, so `\edef` can be used rather than `\g@addto@macro`.

```
      {\ifsi@switch
         \si@log@debug{Adding `#1' to exponent for `\si@num@arg'}%
         \protected@edef\si@num@exp{\si@num@exp#1}%
       \else
         \si@log@debug{Adding `#1' to mantissa for `\si@num@arg'}%
         \protected@edef\si@num@mant{\si@num@mant#1}%
       \fi}}%
```

If the recursion has not bottomed out, another loop occurs.

```
  \ifx\@empty#2\@empty\else
    \si@num@mantexp#2\@empty\@empty\@empty%
  \fi}
```

`\si@num@sign` The digit processor does several things to convert the run of digits, plus potentially a sign and a decimal point into the correct format for output. `\si@num@sign{`⟨*mant/exp*⟩`}`

```
\newcommand*{\si@num@sign}[1]{%
  \expandafter\ifx\expandafter\@empty\csname si@num@#1\endcsname%
    \@empty\else
```

If the whole argument is not empty, then the sign-testing macro is run. This will return the sign in `\si@tempa` and the digits in `\si@tempb`.

```
    \expandafter\expandafter\expandafter\si@num@gensign%
      \csname si@num@#1\endcsname\@empty\@empty\@empty%
```

`\si@tempa` If a sign has to be added to unsigned numbers, this is done here.

`\si@tempc`

```
    \edef\si@tempc{#1}%
    \ifx\@empty\si@tempa\@empty
      \def\si@tempa{mant}%
      \ifx\si@tempa\si@tempc
        \ifsi@num@signmant
          \si@log@debug{Adding sign \si@sign\space to mantissa for
            `\si@num@arg'}%
          \protected@edef\si@tempa{\si@sign}%
        \else
          \def\si@tempa{}%
        \fi
      \else
        \ifsi@num@signexp
          \si@log@debug{Adding sign \si@sign\space to exponent for
            `\si@num@arg'}%
          \protected@edef\si@tempa{\si@sign}%
        \else
          \def\si@tempa{}%
        \fi
      \fi
    \fi
```

If there is no sign, then the original macro contains a pure number, and nothing happens (an empty number has already been tested for).

```
    \ifx\@empty\si@tempa\@empty
      \def\si@tempa{mant}%
      \ifx\si@tempa\si@tempc
        \si@log@debug{Unsigned mantissa for `\si@num@arg'}%
      \else
        \si@log@debug{Unsigned exponent for `\si@num@arg'}%
      \fi
    \else
```

There is a sign, so it is added to the output stack.

```
      \expandafter\protected@edef\csname si@num@#1out\endcsname%
        {\noexpand\ensuremath{\si@tempa}}%
```

A sign but no number can only be correct if the input is something like `-e10` to give -10^{10}.

```
      \ifx\@empty\si@tempb\@empty
        \expandafter\def\csname si@num@#1\endcsname{}%
```

`\si@tempa` `\si@tempa` is no longer needed, so can be reused.

```
        \def\si@tempa{mant}%
```

Checks to see if this is a mantissa, and that there is an exponent.

```
        \ifx\si@tempa\si@tempc
          \ifx\@empty\si@num@exp\@empty
            \si@log@warn{Sign but no number for `\si@num@arg'}%
          \fi
        \else
          \si@log@warn{Sign but no number for `\si@num@arg'}%
        \fi
      \else
        \expandafter\protected@edef\csname si@num@#1\endcsname%
          {\si@tempb}%
      \fi
    \fi
  \fi}
```

`\si@num@gensign` `\si@tempa` `\si@tempb` 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.

`\si@num@gensign`⟨*char*⟩⟨*char*⟩⟨*chars*⟩`\@empty`

```
\def\si@num@gensign#1#2#3\@empty{%
  \si@num@ifchr{#1}{\si@numsign}{%
    \si@num@ifchr{#2}{\si@numsign}{%
      \if +#1
        \if -#2
          \si@log@debug{Found sign combination +- for `\si@num@arg'}%
          \def\si@tempa{\pm}%
        \else
          \si@log@warn{Unknown sign combination `#1#2'}%
          \def\si@tempa{#1#2}%
        \fi
      \else
        \if -#1
          \if +#2
          \si@log@debug{Found sign combination -+ for `\si@num@arg'}%
            \def\si@tempa{\mp}%
          \else
            \si@log@warn{Unknown sign combination `#1#2'}%
            \def\si@tempa{#1#2}%
          \fi
        \else
          \si@log@warn{Unknown sign combination `#1#2'}%
          \def\si@tempa{#1#2}%
        \fi
```

```
      \fi
      \edef\si@tempb{#3}}%
```

Only one valid sign character.

```
      {\si@log@debug{Found single sign character `#1' for
         `\si@num@arg'}%
       \def\si@tempa{#1}%
       \edef\si@tempb{#2#3}}}%
```

No valid sign, so `\@empty` is returned for the sign .

```
    {\si@log@debug{No sign found for `\si@num@arg'}%
     \def\si@tempa{}%
     \edef\si@tempb{#1#2#3}}}%
```

`\si@num@digits` `\si@tempb` `\si@tempc`

The core digit processor divides the number into the parts before and after the decimal point marker.

`\si@num@digits{`⟨*mant/exp*⟩`}`

```
\newcommand*{\si@num@digits}[1]{%
  \def\si@tempa{}%
  \def\si@tempb{}%
```

The package switch is used to indicate finding a decimal marker.

```
  \si@switchfalse
  \expandafter\expandafter\expandafter\si@num@split%
    \csname si@num@#1\endcsname\@empty\@empty%
```

The pre-decimal part of the number is now in `\si@tempa`, and the post-decimal part in `\si@tempb`. A quick check is made on the pre-decimal part of the number.

```
  \ifx\@empty\si@tempa\@empty
    \ifsi@num@padlead
      \si@log@debug{Adding leading zero for `\si@num@arg'}%
      \def\si@tempa{0}%
    \fi
  \fi
```

A second test is needed, in case a zero should be added when a decimal marker is followed by nothing at all. Here, the fact that a decimal marker was found is needed; the test is done now so `\ifsi@switch` can be reused.

```
  \ifx\@empty\si@tempb\@empty
    \ifsi@num@padtrail
      \ifsi@switch
        \si@log@debug{Adding trailing zero for `\si@num@arg'}%
        \def\si@tempb{0}%
      \fi
    \fi
  \fi
```

The contents of `\si@tempa` and `\si@tempb` are now completed. Some error checking is done, in case an odd argument has been given.

```
  \ifx\@empty\si@tempa\@empty
    \ifx\@empty\si@tempb\@empty\else
      \si@num@sepdigits{#1}%
    \fi
  \else
    \si@num@sepdigits{#1}%
```

```
  \fi}
```

`\si@num@split` The `\si@num@split` macro compares each character in the input against the list of characters valid at this stage: numbers, decimal markers and "extra" characters. Before finding a decimal marker, numbers and extra characters are added to `\si@tempa`; after a decimal is found, characters are added to `\si@tempb`.
`\si@num@split`⟨*char*⟩⟨*chars*⟩`\@empty`

```
\def\si@num@split#1#2\@empty{%
  \si@num@ifchr{#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'}%
      \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@num@ifchr{#1}{\si@numsign}{%
      \si@log@err{Misplaced sign in `\si@num@arg'}
        {Sign characters `\si@numsign' can only occur\MessageBreak
           at the start of a number}}{%
```

The current character is added to the appropriate stack.

```
      \ifsi@switch
        \si@log@debug{Adding `#1' to decimal part for `\si@num@arg'}%
        \protected@edef\si@tempb{\si@tempb#1}%
      \else
        \si@log@debug{Adding `#1' to integer part for `\si@num@arg'}%
        \protected@edef\si@tempa{\si@tempa#1}%
      \fi}}%
```

Unless the recursion has bottomed, loop round again.

```
  \ifx\@empty#2\@empty\else
    \si@num@split#2\@empty\@empty%
  \fi}
```

`\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` The `\si@num@sepdigits` macro is only called if at least one of the mantissa and exponent contain something to output. The integer and decimal parts of the number are processed separately. First, a check is made to see if each part contains "extra" characters; if it does, no digit-separating is even attempted.
`\si@num@sepdigits{`⟨*mant/exp*⟩`}`

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

`\si@tempc` If the input is a pure number, then separation is attempted.

```
    {\expandafter\si@num@int\expandafter{\si@tempa}}%
```

```
  \def\si@tempc{}%
  \ifx\@empty\si@tempb\@empty\else
    \protected@edef\si@tempc{\noexpand\ensuremath{{\noexpand%
      \si@decimalsign}}}%
    \si@num@decimalhook%
    \si@num@ifextra{\si@tempb}{}%
      {\expandafter\si@num@dec\expandafter{\si@tempb}}%
  \fi
```

The construction is finalised by re-combining the number.

```
  \expandafter\protected@edef\csname si@num@#1out\endcsname%
    {\csname si@num@#1out\endcsname\si@tempa\si@tempc\si@tempb}}
```

`\si@num@ifextra` `\si@num@extra`

A relatively simple test for "extra" characters. Once again, a bit of group trickery is used.

`\si@num@ifextra{`⟨*integer*⟩`}`
`\si@num@extra`⟨*char*⟩⟨*chars*⟩`\@empty`

```
\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'}%
      \endgroup\expandafter\@firstoftwo%
    \else
      \endgroup\expandafter\@secondoftwo%
    \fi}
\def\si@num@extra#1#2\@empty{%
  \ifx\@empty#1\@empty\else
    \si@num@ifchr{#1}{\si@numextra}{\si@switchtrue}{}%
    \ifx\@empty#2\@empty\else
      \si@num@extra#2\@empty\@empty%
    \fi
  \fi}
```

`\si@num@int` `\si@tempa`

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.

`\si@num@int{`⟨*integer-part*⟩`}`

```
\newcommand*{\si@num@int}[1]{%
  \def\si@tempa{}%
  \ifsi@sepfour
    \si@num@intfmt{}#1\@empty\@empty\@empty%
  \else
    \si@num@iffive{#1}
      {\si@num@intfmt{}#1\@empty\@empty\@empty}
      {\def\si@tempa{#1}}%
  \fi}
```

`\si@num@iffive` `\si@num@five`

A test is needed for the presence of more than four characters.

`\si@num@iffive{`⟨*num*⟩`}`

\si@num@five⟨*char*⟩⟨*char*⟩⟨*char*⟩⟨*char*⟩⟨*chars*⟩\end

```
\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
\si@num@fiint

The business end of the integer formatter. \si@num@intfmt{⟨*char*⟩}{⟨*char*⟩}{⟨*char*⟩}{⟨*char*⟩}
\si@num@fiint⟨*chars*⟩\fi\fi\fi

```
\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
\si@tempa

For adding separation to integers, an extra function is needed.
\si@num@intsep{⟨*char*⟩}{⟨*char*⟩}{⟨*char*⟩}{⟨*char*⟩}

```
\newcommand*{\si@num@intsep}[4]{%
  \protected@edef\si@tempa{\si@tempa#1#2#3}%
  \if\relax#4\relax
  \else
    \protected@edef\si@tempa{\si@tempa\noexpand\ensuremath{\noexpand%
      \si@digitsep}}%
    \expandafter\si@num@intsep\expandafter#4%
  \fi}
```

\si@num@dec
\si@num@decfmt
\si@tempb

Formatting a decimal uses a similar mechanism, but with a few alterations needed.
\si@num@dec{⟨*decimal-part*⟩}
\si@num@decfmt{⟨*char*⟩}{⟨*char*⟩}{⟨*char*⟩}{⟨*char*⟩}

```
\newcommand*{\si@num@dec}[1]{%
  \def\si@tempb{}%
  \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@tempb{\si@tempb#1}}%
```

```
  \fi
}
\newcommand*{\si@num@decfmt}[4]{%
  \protected@edef\si@tempb{\si@tempb#1#2#3}%
  \ifx\@empty#4\@empty%
  \else
    \protected@edef\si@tempb{\si@tempb\noexpand\ensuremath{\noexpand%
      \si@digitsep}}%
    \expandafter\si@num@decfmt\expandafter#4%
  \fi}
```

16.10 Formatting angles

`\ang` The approach used here is similar to that in sistyle, but has been modified in a few ways.

`\ang[`⟨*options*⟩`]{`⟨*decimal-angle*⟩`}`

`\ang[`⟨*options*⟩`]{`⟨*deg*⟩`;`⟨*min*⟩`;`⟨*sec*⟩`}`

```
\DeclareRobustCommand*{\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 semicolons.

`\si@ang@parse`⟨*num*⟩`;`⟨*num*⟩`;`⟨*num*⟩`;`⟨*chars*⟩`\@nil`

```
\def\si@ang@parse#1;#2;#3;#4\@nil{\si@ang@set{#1}{#2}{#3}}
```

`\si@ang@killdegree` A mechanism is needed to handle moving the angle unit sings for the `astroang` option. This requires two steps, producing the sign over the decimal sign and preventing duplicate symbols appearing.

`\si@ang@killminute`

`\si@ang@killsecond`

`\si@ang@astrosign` `\si@ang@astrosign{`⟨*degree/minute/second*⟩`}`

`\si@ang@decimalsign`

```
\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}
\newcommand*{\si@ang@astrosign}[1]{%
  \renewcommand*{\si@decimalsign}{%
    \rlap{\si@ang@decimalsign}%
    \expandafter\csname si@sym@#1\endcsname}%
    \def\si@num@decimalhook{\expandafter\aftergroup%
      \csname si@ang@kill#1\endcsname}}%
```

`\si@ang@set` The `\si@ang@set` macro does the work of assigning the degrees, minutes and seconds, and actually typesetting the result.

`\si@ang@set{`⟨*num*⟩`}{`⟨*num*⟩`}{`⟨*num*⟩`}`

```
\newcommand*{\si@ang@set}[3]{%
```

`\si@ang@degs` First, the three macros that will contain the measures must exist.

`\si@ang@mins`

`\si@ang@secs`

```
  \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
  \protected@edef\si@ang@decimalsign{\si@decimalsign}%
```

`\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'}%
      \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'}%
      \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 Padding is only added if requested; the zero is a literal.
\si@ang@pad{⟨*num*⟩}

```
\newcommand*{\si@ang@pad}[1]{%
  \ifsi@ang@padsmall
    #1%
  \else
    \relax%
  \fi}
```

\si@ang@num Modified versions of \num, one to typeset angles without a leading sign and the
\si@ang@signlessnum other with.
\si@ang@num{⟨*degree/minute/second*⟩}{⟨*num*⟩}
\si@ang@signlessnum{⟨*degree/minute/second*⟩}{⟨*num*⟩}

```
\newcommand{\si@ang@num}[2]{%
  \begingroup%
    \si@ang@movesign{#1}%
    \si@num{#2}%
  \endgroup}
\newcommand{\si@ang@signlessnum}[2]{%
  \begingroup%
    \si@ang@movesign{#1}%
    \sisetup{addsign=none}%
    \si@num{#2}%
  \endgroup}
```

16.11 New column types

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 first part of the job is to make a new column type. The letters D, N and R are taken by other packages, so s (for si) is chosen. As in rccol and numprint, initially no definition is given as lots of code needs to be added.

```
\newcolumntype{s}{}
```

\NC@rewrite@s Following the numprint approach, the \NC@rewrite@S macro is now changed to provide a hook for the collection of the tabular material. 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. After resetting the storage token registers, the internal macro which handles optional arguments is called.

```
\renewcommand*{\NC@rewrite@s}{%
  \@ifnextchar[%]
    {\si@tab@rewrite}
    {\si@tab@rewrite[]}
}
```

\si@tab@rewrite An optional argument can now be picked up (this does not work using the
\si@tempa optional argument to \renewcommand for \NC@rewrite@S). 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.
\si@tab@rewrite[⟨*options*⟩]

```
\def\si@tab@rewrite[#1]{%
  \edef\si@tempa{\the\@temptokena
    >{\noexpand\si@tab@begin[#1]\noexpand\ignorespaces}c%
    <{\noexpand\si@tab@end}}%
  \@temptokena\expandafter{\si@tempa}%
```

With the assignment done, the normal action of the array package is continued.

```
  \NC@find}
```

`\si@tab@numtoks` `\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@numtoks
\newtoks\si@tab@pretoks
\newtoks\si@tab@posttoks
```

`\si@tab@begin` The lead-off macro starts by setting any local values for `\sisetup`. Although this is an internal macro, square brackets for the option list are retained to make the option that this argument may be empty.
`\si@tab@begin[`⟨*options*⟩`]`

```
\newcommand*{\si@tab@begin}[1][]{%
  \begingroup
    \sisetup{#1}%
    \si@tab@numtoks{}%
    \si@tab@pretoks{}%
    \si@tab@posttoks{}%
    \si@switchfalse
    \si@log@debug{Processing s column cell contents}%
    \si@tab@gettok}
```

`\si@tab@gettok` `\si@tab@next` The iteration macro for collecting up tokens in the input. This is a pretty much direct copy from numprint. First `#1` is checked against various special values.
`\si@tab@gettok{`⟨*cell-contents*⟩`}`

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

The current cell could be the end of a line.

```
  \ifx\tabularnewline#1
    \let\si@tab@next\tabularnewline
  \else
```

If the table is in the usual `\begin...\end` construct, the `\end` might appear.

```
    \ifx\end#1
      \let\si@tab@next\end
    \else
```

The check for `\si@tab@end` deals with the likely situation that the current cell is not the last of the line; the result will be that the end-of-cell macro will be present.

```
      \ifx\si@tab@end#1
        \let\si@tab@next\si@tab@end
      \else
```

If `\begin...\end` has not been used for the table, then `\endtabular` might crop up.

```
        \ifx\endtabular#1
```

```
          \let\si@tab@next\endtabular
        \else
```

Apparently, `tabularx` might have a `\csname` at the end of the cell.

```
          \ifx\csname#1
            \let\si@tab@next\csname
          \else
```

`\relax` is always a possibility.

```
            \ifx\relax#1\relax
              \let\si@tab@next\relax
            \else
```

If the macro gets here, then the input should be stored, either as part of a number or to be appended to the number. This is checked by using `\si@numvalid`; initially this will hold the valid characters defined before, but will be altered to `\@empty` if collection of items after a number is in operation. The system is set to recur, and the input is saved to the appropriate token register.

```
              \let\si@tab@next\si@tab@gettok
              \si@num@ifchr{#1}{\si@numvalid}
                {\si@switchtrue
                 \si@log@debug{Found valid cell contents '#1'}%
                 \si@tab@numtoks=\expandafter{\the\si@tab@numtoks#1}}
                {\si@log@debug{Found other cell contents \string#1}%
                 \si@tab@othertok{#1}}%
            \fi
          \fi
        \fi
      \fi
    \fi
  \fi
```

Finally, execute whatever should be the next step.

```
  \si@tab@next}
```

`\si@tab@othertok` Unrecognised input is added to a token register, either before or after the number.
`\si@tab@othertok{`⟨*chars*⟩`}`

```
\newcommand*{\si@tab@othertok}[1]{%
  \ifsi@switch
```

If working after a number has been found, it is necessary to prevent any more input being added to the number.

```
    \si@tab@posttoks=\expandafter{\the\si@tab@posttoks#1}%
  \else
    \si@tab@pretoks=\expandafter{\the\si@tab@pretoks#1}%
  \fi}
```

`\si@tab@end` At the end of the cell, the actual output has to occur.

```
\newcommand{\si@tab@end}{%
  \hfil%
  \the\si@tab@pretoks%
```

If no number was found, then output is skipped.

```
  \ifsi@switch
    \expandafter\si@tab@numout%
  \fi
```

```
  \the\si@tab@posttoks%
  \hfil%
  \endgroup}
```

`\si@tempcnta` `\si@tempcntb` 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}{%
  \let\si@num@format\si@tab@num@format
```

Using the modified form of `\si@num`, the input (in `\si@tab@numtoks` is parsed. This results in the mantissa before the decimal place in `\si@num@mantout` and the rest of the mantissa, plus any exponent part, in `\si@num@out`. The part of the mantissa after the decimal marker (if any) is stored in `\si@tab@mantout`, which can therefore be used as a flag for the inclusion of a decimal sign.

```
  \expandafter\si@num\expandafter{\the\si@tab@numtoks}%
  \afterassignment\si@tab@format\expandafter\si@tempcnta%
    \si@tabformat\relax}
```

`\si@tab@mantout` A storage macro is needed.

```
\newcommand*{\si@tab@mantout}{}
```

`\si@tab@num@format` A modified version of `\si@num@format` is needed, as the "decomposed" number is needed directly by the table formatting system.
`\si@tab@num@format{`⟨*num*⟩`}`

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

`\si@tab@org@sepdigits` The crucial sub-macro is redirected.

```
  \let\si@tab@org@sepdigits\si@num@sepdigits
  \let\si@num@sepdigits\si@tab@num@sepdigits
```

`\si@num@arg` With that done, things continue as in the original.

```
  \edef\si@num@arg{#1}%
  \si@switchfalse
  \expandafter\si@num@mantexp\si@num@arg\@empty\@empty%
  \si@num@sign{mant}%
  \ifx\@empty\si@num@mant\@empty\else
     \si@num@digits{mant}%
```

`\si@num@out` The pre-decimal part of the the number is in `\si@num@mantout`, with the post-decimal part in `\si@tab@mantout`. This ensures that there is no need to shuffle the location of any sign. The macro now continues to build up everything after the decimal sign in `\si@num@out`.

```
     \protected@edef\si@num@out{\si@tab@mantout}%
  \fi
```

For the exponent, processing is back to normal.

```
  \let\si@num@sepdigits\si@tab@org@sepdigits
  \si@num@sign{exp}%
  \ifx\@empty\si@num@exp\@empty\else
```

```
    \si@num@digits{exp}%
    \ifx\@empty\si@num@mant\@empty\else
      \protected@edef\si@num@out%
        {\si@num@out\noexpand\ensuremath{\noexpand\si@expproduct}}%
    \fi
    \protected@edef\si@num@out%
      {\si@num@out\si@exppower%
        \noexpand\textsuperscript{\si@num@expout}}%
  \fi
  \ifx\@empty\si@num@mant\@empty
    \ifx\@empty\si@num@exp\@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@numlist')?}%
```

`\si@tab@mantout` Need to clear both storage areas.

```
      \renewcommand*\si@num@out{}%
      \renewcommand*\si@tab@mantout{}%
    \fi
  \fi}
```

`\si@tab@num@sepdigits` An altered version of `\si@num@sepdigits` is needed, so that the division of the data is made before and after the decimal sign for the mantissa.
`\si@tab@num@sepdigits{`⟨*num*⟩`}`

```
\newcommand*{\si@tab@num@sepdigits}[1]{%
  \si@num@ifextra{\si@tempa}{}
    {\expandafter\si@num@int\expandafter{\si@tempa}}%
  \def\si@tempc{}%
  \ifx\@empty\si@tempb\@empty\else
    \si@num@ifextra{\si@tempb}{}%
      {\expandafter\si@num@dec\expandafter{\si@tempb}}%
  \fi
```

The storage of the results is different to the original version. The pre-decimal part (plus any sign) is stored in the `\si@num@#1out` macro, while the post-decimal part ends up in `\si@tab@#1out`. No decimal sign is added in at all.

```
  \expandafter\protected@edef\csname si@num@#1out\endcsname%
    {\csname si@num@#1out\endcsname\si@tempa}%
  \expandafter\protected@edef\csname si@tab@#1out\endcsname%
    {\si@tempb}}%
```

`\si@tab@prebox` `\si@tab@postbox` `\si@tempbox` The various boxes needed for the column centring are declared Unlike the dcolumn original, private boxes are used here. `\si@tempbox` 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@postbox
\newbox\si@tempbox
```

`\si@tab@format` The formatting set up is taken from dcolumn, with a few minor changes to fit the scheme used here. There is only one argument here, as the appearance of the decimal sign is handled by the keyval system. The numerical test here has been changed, compared to dcolumn, so that a value of zero gives a column centred on

the decimal marker.

`\si@tab@format{`⟨*num*⟩`}\relax`

```
\def\si@tab@format#1\relax{%
  \ifnum\z@<\si@tempcnta
    \expandafter\si@tab@right%
  \else
    \expandafter\si@tab@centre%
  \fi
  {#1}%
```

Output of the formatted data occurs here; both positioning macros produce formatted data in boxes zero and two.

```
  \box\si@tab@prebox\box\si@tab@postbox}
```

`\si@tab@centre` This macro is executed if the decimal marker is at the centre of the column. The argument is needed here to throw away anything left on the input stack by `\si@tab@format`. Unlike dcolumn, only a single macro is needed here, as the (divided) number is already available.

`\si@tab@centre{`⟨*gobble*⟩`}`

```
\newcommand*\si@tab@centre[1]{%
```

Box zero is used to hold the pre-decimal part, with box two holding the post-decimal part *if* it is needed.

```
  \setbox\si@tab@prebox=\hbox%
    {\expandafter\si@out@num\expandafter{\si@num@mantout}}%
  \ifx\@empty\si@tab@mantout\@empty
    \ifx\@empty\si@num@out\@empty
      \setbox\si@tab@postbox=\hbox%
        {\phantom{\ensuremath{{\si@decimalsign}}}}%
    \else
      \setbox\si@tab@postbox=\hbox%
        {\expandafter\si@out@num\expandafter{\si@num@out}}%
    \fi
  \else
    \setbox\si@tab@postbox=\hbox%
      {\ensuremath{{\si@decimalsign}}%
        \expandafter\si@out@num\expandafter{\si@num@out}}%
  \fi
```

Which of the two boxes is wider is now checked, and the smaller is padded out.

```
  \ifdim \wd\si@tab@prebox>\wd\si@tab@postbox
    \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}
```

`\si@tab@predim` Some storage dimensions are declared.

`\si@tab@postdim`
`\si@tempdima`
`\si@tempdimb`

```
\newdimen\si@tab@predim
\newdimen\si@tab@postdim
\newdimen\si@tempdima
\newdimen\si@tempdimb
```

\si@tab@right The column is not centred on the decimal marker; the user specifies how many characters on each side are allowed for.
\si@tab@right{⟨*num*⟩}

```
\newcommand*\si@tab@right[1]{%
```

The width of a character is measured, and stored.

```
  \setbox\si@tempbox=\hbox{\si@out@num{1}}
  \si@tempdima\wd\si@tempbox
```

\si@tab@prea
\si@tab@preb If #1 is empty, then no special processing is needed for box two. On the other hand, if there is something in #1 then a bit of re-arranging is done. In particular notice that \si@tempcnta is used with the pre-decimal value, before the post-decimal setting is saved.

```
  \ifx\relax#1\relax
    \hfill
    \let\si@tab@prea\relax
    \let\si@tab@preb\relax
  \else
    \si@tab@predim\the\si@tempcnta\si@tempdima
    \si@tab@sepcorr{pre}%
    \edef\si@tab@prea{to\si@tab@predim}%
    \edef\si@tab@preb{\hss\hfill}%
    \si@tempcnta\@gobble#1\relax
  \fi
```

The width of the box needed is calculated by multiplying the width of a character (in \si@tempdima by the number of characters requested (in \si@tempcnta. The width of the decimal sign is also allowed for and added on.

```
  \si@tab@postdim\si@tempcnta\si@tempdima
  \setbox\si@tempbox=\hbox{\ensuremath{{\si@decimalsign}}}%
  \advance\si@tab@postdim\wd\si@tempbox
  \si@tab@sepcorr{post}%
```

The pre-decimal part of the number is now added to box zero, with the post decimal part in box two if needed.

```
  \setbox\si@tab@prebox=\hbox\si@tab@prea{\si@tab@preb%
    \expandafter\si@out@num\expandafter{\si@num@mantout}}%
  \ifx\@empty\si@tab@mantout\@empty
    \setbox\si@tab@postbox=\hbox to\si@tab@postdim%
      {\expandafter\si@out@num\expandafter{\si@num@out}\hfil}%
  \else
    \setbox\si@tab@postbox=\hbox to\si@tab@postdim%
      {\ensuremath{{\si@decimalsign}}\expandafter\si@out@num%
        \expandafter{\si@num@out}\hfil}%
  \fi}
```

\si@tab@sepcorr 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.
\si@tab@sepcorr{⟨*num*⟩}

```
\newcommand*{\si@tab@sepcorr}[1]{%
  \si@tempcntb\the\si@tempcnta\relax
```

Calculate how many groups of three there are, then allow for not separating four characters if \ifsi@sepfour is false.

```
\divide\si@tempcntb\thr@@
\ifsi@sepfour\else
  \ifnum\the\si@tempcnta=4
    \si@tempcntb\z@
  \fi
\fi
```

The width of the separators is measured, and the correct number of separator widths are added to the box dimension.

```
\setbox\si@tempbox=\hbox{\ensuremath{\si@digitsep}}%
\expandafter\advance\csname si@tab@#1dim\endcsname%
  \si@tempcntb\wd\si@tempbox}%
```

16.12 Units

\SI
\unitsym

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 \unitsym, which work in very similar ways. \unitsym 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 argument to \SI
\SI[⟨*options*⟩]{⟨*num*⟩} \unitsym[⟨*options*⟩]{⟨*unit*⟩}

```
\DeclareRobustCommand*{\SI}[2][]{%
  \@ifnextchar[%]
    {\si@SI[#1]{#2}}
    {\si@SI[#1]{#2}[]}}
\DeclareRobustCommand*{\unitsym}[2][]{\si@SI[#1]{}[]{#2}}
```

\newunit
\renewunit
\provideunit

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.
\newunit[⟨*valuesep=none*⟩]{⟨*unit*⟩}{⟨*symbol*⟩}
\renewunit[⟨*valuesep=none*⟩]{⟨*unit*⟩}{⟨*symbol*⟩}
\provideunit[⟨*valuesep=none*⟩]{⟨*unit*⟩}{⟨*symbol*⟩}

```
\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@unt@defunit[#1]{#2}{#3}}}
\newcommand*{\provideunit}[3][]{%
  \si@ifdefinable{#2}
    {\si@unt@defunit[#1]{#2}{#3}}
    {}}
```

\newprefix
\renewprefix
\provideprefix

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.
\newprefix{⟨*multiple*⟩}{⟨*powers-ten*⟩}{⟨*symbol*⟩}
\renewprefix{⟨*multiple*⟩}{⟨*powers-ten*⟩}{⟨*symbol*⟩}
\provideprefix{⟨*multiple*⟩}{⟨*powers-ten*⟩}{⟨*symbol*⟩}

```
\newcommand*{\newprefix}[3]{%
  \si@ifdefinable{#1}
    {\si@unt@defprefix{#1}{#2}{#3}}
    {\si@log@err{Prefix \string#1 already defined!}\@eha}}
\newcommand*{\renewprefix}[3]{%
  \si@ifdefinable{#1}
    {\si@log@err{Prefix \string#1 undefined}\@ehc
     \si@unt@defprefix{#1}{#2}{#3}}
    {\si@unt@defprefix{#1}{#2}{#3}}}
\newcommand*{\provideprefix}[3]{%
  \si@ifdefinable{#1}
    {\si@unt@defprefix{#1}{#2}{#3}}
    {}}
```

\newpower
\renewpower
\providepower

Here power multiples for units are set up. As with units and multiples, a layered approach is used to keep things easy to maintain. The optional argument here is *not* a keyval one: only post is a valid value.
\newpower[⟨*post*⟩]{⟨*num*⟩}{⟨*power*⟩}
\renewpower[⟨*post*⟩]{⟨*num*⟩}{⟨*power*⟩}
\providepower[⟨*post*⟩]{⟨*num*⟩}{⟨*power*⟩}

```
\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@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. The flag is true outside of the processor

```
\newif\ifsi@unt@num\si@unt@numtrue
```

\si@SI
\si@unt@SIopts

The internal processing starts with \si@SI, which processes the second optional argument to \SI (which is empty for \unitsym). Everything is set up in a group, and processing begins by handling the options.
\si@SI[⟨*options*⟩]{⟨*unit*⟩}[⟨*preunit*⟩]{⟨*unit*⟩}

```
\def\si@SI[#1]#2[#3]#4{%
  \begingroup
    \si@ifnotmtarg{#1}
      {\sisetup{#1}%
       \edef\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{#2}
      {\si@log@debug{Number found in \string\SI\space argument}%
       \num{#2}%
       \si@unt@numtrue}%
    \si@ifnotmtarg{#4}
      {\si@unt@printunit{#4}}%
  \endgroup}
```

`\si@unt@ifliteral` `\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.

`\si@unt@ifliteral{`⟨*text*⟩`}`

```
\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@tempbox=\hbox{\si@unt@out{#1}}%
    \ifdim\wd\si@tempbox>\z@\relax
      \endgroup\expandafter\@firstoftwo%
    \else
      \endgroup\expandafter\@secondoftwo%
    \fi}
```

`\ifsi@unt@litout` `\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).

`\si@unt@printunit{`⟨*unit*⟩`}`

```
\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
\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@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` The internal macro for defining a unit does not check for redefinition; that is done by the user macros. `\si@unt@defunit[`⟨*valuesep=none*⟩`]{`⟨*unit*⟩`}{`⟨*symbol*⟩`}`

```
\newcommand*{\si@unt@defunit}[3][]{%
  \si@log@debug{Declaring unit \string#2 with \MessageBreak
    meaning \string#3}%
```

The optional argument can only have the value `valuesep=none`, which is used to prevent a space occurring between a number and the units (for example, with `\degree`). The optional argument needs to be saved; `\edef` is used so there is no issue with redefinition. The macro name is effectively "reversed" so that life is easier with the expansions here.

```
  \si@ifnotmtarg{#1}
    {\expandafter\expandafter\expandafter\def\expandafter%
      \csname\expandafter\@gobble\string#2@opt@unt@si\endcsname{#1}}%
```

The unit macro itself is now defined. The definition simply selects the correct path for the rest of the processing to go down. To avoid needing specialised gobbling macros, the optional nature of the first argument is dropped.

```
  \DeclareRobustCommand*{#2}[1][]{%
    \ifsi@unt@littest
      \expandafter\si@gobblethree
    \else
```

For literal output, the third argument is all that is needed.

```
      \ifsi@unt@litout
        \expandafter\expandafter\expandafter\@gobbletwo
      \else
        \expandafter\expandafter\expandafter\si@unt@unit%
      \fi
    \fi
    {##1}{#2}{#3}}}
```

`\si@gobblethree` LaTeX does not have a `\@gobblethree` macro, but one is needed.

```
\long\def \si@gobblethree #1#2#3{}
```

`\si@unt@defprefix` As with units, multiples are defined by an internal macro. `\si@unt@defprefix{`⟨*multiple*⟩`}{`⟨*powers-t*

```
\newcommand*{\si@unt@defprefix}[3]{%
  \si@log@debug{Declaring multiple \string#1 with \MessageBreak
```

```
    meaning \string#3}%
  \DeclareRobustCommand{#1}{%
    \ifsi@unt@littest
      \expandafter\si@gobblethree
    \else
      \ifsi@unt@litout
        \expandafter\expandafter\expandafter\@gobbletwo
      \else
        \expandafter\expandafter\expandafter\si@unt@prefix%
      \fi
    \fi
    {#1}{#2}{#3}}}
```

`\si@unt@defpower` 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.

`\si@unt@defpower[`⟨*post*⟩`]{`⟨*power*⟩`}{`⟨*num*⟩`}`

```
\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\edef\expandafter%
    \csname\expandafter\@gobble\string#2@opt@si\endcsname{#1}%
  \DeclareRobustCommand{#2}{%
    \ifsi@unt@littest
      \expandafter\@gobbletwo
    \else
```

The literal output here does not need to gobble anything, but uses `\textsuperscript` to get the correct effect. This will of course give very odd results for prefix powers.

```
      \ifsi@unt@litout
        \expandafter\expandafter\expandafter\si@unt@litpower%
      \else
        \expandafter\expandafter\expandafter\si@unt@power%
      \fi
    \fi
    {#2}{#3}}}
```

`\si@unt@unithook` `\si@unt@unit` 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.

`\si@unt@unit{`⟨*num*⟩`}{`⟨*unit*⟩`}{`⟨*symbol*⟩`}`

```
\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` At this stage, the flag will be set for the first item to be processed whichever route the unit has been called by.
`\si@unt@firstorsecond{`⟨*num*⟩`}{`⟨*macro*⟩`}`

```
\newcommand{\si@unt@firstorsecond}[2]{%
\ifsi@unt@first
  \expandafter\si@unt@first%
\else
  \expandafter\si@unt@second%
\fi
{#1}{#2}}%
```

`\si@unt@first` For the first unit in the input, some extra tasks are needed. First, the optional argument for the unit macro needs to be tested.
`\si@unt@first{`⟨*num*⟩`}{`⟨*unit*⟩`}`

```
\newcommand*{\si@unt@first}[2]{%
\si@ifnotmtarg{#1}
  {\num{#1}}%
\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` `\si@unt@setSIopts` A rather long set of `\expandafter` commands to get the options to set safely.
`\si@unt@setopts{`⟨*unit*⟩`}`

```
\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` `\si@unt@third` 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.
`\si@unt@second{`⟨*num*⟩`}{`⟨*unit*⟩`}`

```
\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
```

`\si@tempa` 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`.

```
  \def\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}{%
  \def\si@tempa{space}%
  \expandafter\ifx\csname si@unt@lastadd\si@unt@checkstack%
    \endcsname\si@tempa\else
    \expandafter\si@unt@reciptest%
  \fi}
```

`\si@unt@prefix` Actual output of the prefixes.
`\si@unt@prefix{`⟨*multiple*⟩`}{`⟨*powers-ten*⟩`}{`⟨*symbol*⟩`}`

```
\newcommand*{\si@unt@prefix}[3]{%
  \si@unt@firstorsecond{}{#1}%
  \ifsi@prefixnum
    \expandafter\si@unt@countprefix%
  \else
    \expandafter\si@unt@addprefix%
  \fi
  {#2}{#3}}
```

`\si@unt@addprefix` To add the prefix, a little translation is needed.
`\si@unt@countprefix{`⟨*gobble*⟩`}{`⟨*symbol*⟩`}`

```
\newcommand*{\si@unt@addprefix}[2]{\si@unt@addtostack{prefix}{#2}}
```

`\si@unt@prefixcnt` On the other hand, to count the prefix numeral, the symbol is thrown away.
`\si@unt@countprefix` `\si@unt@countprefix{`⟨*powers-ten*⟩`}{`⟨*gobble*⟩`}`
`\si@unt@invprefix`

```
\newcount\si@unt@prefixcnt
\newcommand*{\si@unt@countprefix}[2]{%
 \si@tempcnta#1\relax
 \ifsi@unt@per
   \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` For literal power output, the number can't simply be dumped, so a macro is needed.
`\si@unt@litpower{`⟨*gobble*⟩`}{`⟨*num*⟩`}`

```
\newcommand*{\si@unt@litpower}[2]{\textsuperscript{#2}}
```

`\ifsi@unt@prepower` The handling of powers starts by checking if the number needs to be reversed.
`\si@unt@power` `\si@unt@power{`⟨*power*⟩`}{`⟨*num*⟩`}`

```
\newif\ifsi@unt@prepower
\newcommand*{\si@unt@power}[2]{%
  \si@unt@powerdim #2 pt\relax
  \ifsi@frac\else
    \ifsi@unt@per
      \expandafter\expandafter\expandafter\si@unt@invpower%
    \fi
  \fi
  \def\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%
  \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%
  \si@unt@perfalse
  \si@unt@perseenfalse}
```

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
    \edef\si@tempa{unit}%
    \expandafter\ifx\csname si@unt@lastadd\si@unt@checkstack%
      \endcsname\si@tempa
      \si@unt@addtostack{power}{^{\num{\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

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
\DeclareRobustCommand*{\per}{\si@per}
\newcommand*{\si@per}{%
  \si@unt@firstorsecond{}{\per}%
  \ifsi@unt@per
    \expandafter\si@unt@perfalse
  \else
    \expandafter\si@unt@pertrue
  \fi}
```

\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
\si@unt@addtostack
\si@tempa
\si@tempb

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. By indicating the type of data to be added to the stack, problems can be avoided.
\si@unt@addtostack{⟨*type*⟩}{⟨*token*⟩}

```
\newcommand*{\si@unt@lastadda}{}
\newcommand*{\si@unt@lastaddb}{}
\newcommand*{\si@unt@addtostack}[2]{%
  \edef\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
\si@tempa
\si@tempb

The space added after a prefix needs to be ignored. \si@unt@prespace{⟨*type*⟩}{⟨*stack*⟩}{⟨*token*⟩}{⟨*g*

```
\newcommand*{\si@unt@preplussp}[2]{%
  \def\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

The macro for actually doing the stacking up.
\si@unt@stack{⟨*type*⟩}{⟨*tokens*⟩}

```
\newcommand*{\si@unt@stack}[2]{%
  \expandafter\edef\csname si@unt@lastadd\si@unt@checkstack%
    \endcsname{#1}%
```

\si@tempa
\si@tempb

A count is kept of the number of *units* added to each stack.

```
  \edef\si@tempa{#1}%
  \def\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.

```
  \def\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@holdspace` `\si@unt@addstack` `\si@unt@holdstacka` `\si@unt@holdstackb` Depending on the nature of the addition, it is either held or added to the stack.

`\si@unt@holdspace{`⟨*tokens*⟩`}`
`\si@unt@addstack{`⟨*tokens*⟩`}`

```
\newcommand*{\si@unt@holdstacka}{}
\newcommand*{\si@unt@holdstackb}{}
\newcommand*{\si@unt@holdspace}[1]{%
  \expandafter\protected@edef\csname si@unt@holdstack%
    \si@unt@checkstack\endcsname{#1}}
\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}{%
  \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@text{\ensuremath{{}\si@prefixproduct{}}}%
    \fi
    \let\si@exppower\si@prefixpower
    \num{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
      \expandafter\expandafter\expandafter\si@unt@notabg%
    \fi
  \fi}
\newcommand*{\si@unt@notabg}{%
  \protected@edef\si@unt@stackb{\si@denlbrac\si@unt@stackb%
    \si@denrbrac}}
```

\si@unt@out

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. As usual, \scantokens is used to make life easier.

\si@unt@out{⟨*unit*⟩}

```
\begingroup
  \catcode`\~=\active
  \catcode`\.=\active
  \gdef\si@unt@out#1{%
    \begingroup
      \si@unt@nonlatin%
      \makeatletter%
      \catcode`\~=\active
      \catcode`\.=\active
      \def~{\ensuremath{\si@unitspace}}%
      \def.{\ensuremath{\si@unitsep}}%
      \scantokens{\si@out@text{#1}\@empty}%
    \endgroup}
\endgroup
```

\si@unt@nonlatin

To handle non-Latin symbols in the input, a single macro is provided. Initially, it does nothing

```
\newcommand*{\si@unt@nonlatin}{\relax}
```

The meaning of different characters depends on the encoding used. Thus a test is made for the presence of a suitable package *and* the encoding in use. The various characters can then be handled.

\si@tempa

```
\AtBeginDocument{%
  \@ifpackageloaded{inputenc}
    {\def\si@tempa{latin1}%
     \ifx\inputencodingname\si@tempa
```

The degree symbol is character 176 and the micro symbol is character 181 in latin1.

```
      \si@unt@sym{176}{\si@sym@degree}%
      \si@unt@sym{181}{\si@sym@mu}%
      \si@unt@sym{229}{\si@sym@ringA}%
     \fi}
```

No inputenc available.

```
    {}}
```

\si@unt@sym A macro for declaring symbols: a copy of \DeclareInputMath from inputenc.
\si@unt@sym{⟨*charcode*⟩}

```
\newcommand*{\si@unt@sym}[1]{%
  \bgroup
    \uccode`\~#1%
    \uccode{%
      \egroup
      \def~}}
```

\kilogram With the system set up, the basic unit macros are implemented. The only units
\metre defined whatever options are given are the base SI units.
\mole
\kelvin
\candela
\second
\ampere

```
\newunit{\kilogram}{kg}
\newunit{\metre}{m}
\newunit{\mole}{mol}
\newunit{\second}{s}
\newunit{\ampere}{A}
\newunit{\kelvin}{K}
\newunit{\candela}{cd}
```

\Square Unlike multiples (which can be skipped if needed), the basic powers are also
\squared always defined.
\cubic
\cubed

```
\newpower{\Square}{2}
\newpower[post]{\squared}{2}
\newpower{\cubic}{3}
\newpower[post]{\cubed}{3}
```

\tothe A macro for arbitrary powers, which comes after the unit and so needs to be
\tothe@opt@si marked as such.
\tothe{⟨*num*⟩}

```
\newcommand*{\tothe}[1]{%
  \ifsi@unt@littest
    \expandafter\@gobbletwo
  \else
    \ifsi@unt@litout
      \expandafter\expandafter\expandafter\si@unt@litpower%
    \else
      \expandafter\expandafter\expandafter\si@unt@power%
    \fi
  \fi
  {\tothe}{#1}}
\newcommand*{\tothe@opt@si}{post}
```

16.13 Locales

\si@loc@load When loading a locale, the setup is saved rather than applied. Anything other
\si@loc@sisetup than simple settings should be inside \addtolocale, which is already defined.
\si@loc@load{⟨*locale*⟩}

```
\newcommand*{\si@loc@load}[1]{%
  \let\si@loc@sisetup\sisetup
  \renewcommand*{\sisetup}[1]{%
    \expandafter\def\csname si@loc@#1\endcsname{##1}}
  \si@loadfile{#1}%
  \let\sisetup\si@loc@sisetup}
```

\si@loc@set Setting the locale transfers the settings to \sisetup, and runs any extra macros. \si@loc@set{⟨*locale*⟩}

```
\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}
```

\addtolocale Arbitrary macros may need to be added to the locale. \addtolocale{⟨*locale*⟩}{⟨*commands*⟩}

```
\newcommand*{\addtolocale}[2]{\si@addtocsname{si@loc@#1@extra}{#2}}
```

16.14 Output routine

\si@out@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. \si@out@text{⟨*text*⟩}

```
\begingroup
  \catcode`\^=\active\relax
  \catcode`\-=\active\relax
  \gdef\si@out@text#1{%
    \begingroup
      \catcode`\^=\active\relax%
      \makeatletter%
```

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@fam@italic\si@fam@bold\si@fam@text%
```

In text mode, ^ will execute \textsuperscript, whereas in maths mode it will be converted to \sp, which the kernel defines as ^ with catcode 7. \scantokens is used to set the catcodes here, plus any others that have been set by other pars of the package.

```
      \ifsi@textmode
```

```
        \let^\textsuperscript
        \catcode`\-=\active\relax%
        \let-\si@out@minus
```

The `\@empty` is needed here to mop up any extra space.

```
        \scantokens{#1\@empty}%
      \else
        \let^\sp
        \let\textsuperscript\sp
        $\si@fam@maths{\scantokens{#1}}$%
      \fi}%
```

Everything is done; a bit of tidying up is needed.

```
    \endgroup
    \check@mathfonts}
\endgroup
```

`\si@out@minus` An active minus sign is needed.

```
\newcommand*{\si@out@minus}{\ensuremath{-}}
```

`\si@out@num` For numerical output, the default fonts are controlled slightly differently to text output.

`\si@out@num{`⟨*num*⟩`}`

```
\newcommand*{\si@out@num}[1]{%
  \begingroup
    \sisetup{%
      textdefault=\si@textnumdefault,%
      mathdefault=\si@mathnumdefault}%
    \si@out@text{#1}%
  \endgroup}
```

16.15 Finalisation

With the si kernel macros defined, the package can now run through finalisation. First, the default key values are set. The user options are then processed.

```
\sisetup{
  unitsep=thin,
  valuesep=thin,
  decimalsign=fullstop,
  digitsep=thin,
  obeybold=false,
  inlinebold=text,
  obeyitalic=false,
  sign=plus,
  addsign=none,
  obeymode=false,
  mode=maths,
  padangle=small,
  padnumber=lead,
  sepfour=false,
  tabformat=-1,
  xspace=false,
  per=reciprocal,
  slash=slash,
```

```
  mathsdefault=\si@mathsrm,
  mathsrm=mathrm,
  mathssf=mathsf,
  mathstt=mathtt,
  textdefault=\si@textrm,
  textrm=rmfamily,
  textsf=sffamily,
  texttt=ttfamily,
  mathsnumdefault=\si@mathsrm,
  textnumdefault=\si@textrm,
  numlist=0123456789,
  numdecimal={.,},
  numexp=eEdD,
  numgobble={},
  numsign=+-\pm\mp,
  numextra=(),
  redefsymbols=true,
  load=default,
  noload={},
  eVcorra={0.3ex},
  eVcorrb={0ex},
  denlbrac=(,
  denrbrac=),
  astroang=false,
  loadlocales={},
  loctolang={},
  prefix=symbol,
  prefixpower=ten,
  prefixproduct=times}
\ProcessOptionsX[si]<opt>
```

`\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@ifloaded` A bit of borrowing from the LaTeX kernel.
`\si@ifloaded{`⟨*package*⟩`}`

```
\newcommand*{\si@ifloaded}[1]{%
  \@ifl@aded\si@extension{\si@fileprefix#1}}
```

`\si@loadfile` Loading configuration files is handled here.
`\si@loadfile{`⟨*file*⟩`}`

```
\newcommand*{\si@loadfile}[1]{%
  \si@ifloaded{#1}{}
    {\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}}}}
```

`\si@requirecfgs` `\si@tempb` The configuration files depend on each other.
`\si@requirecfgs{`⟨*cfg-file*⟩`}`

```
\newcommand*{\si@requirecfgs}[1]{%
  \@for\si@tempb:=#1\do{\si@loadfile{\si@tempb}}}
```

`\si@loademfile` For emulation files, an additional check is made.
`\si@loademfile{`⟨*file*⟩`}`

```
\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 si a back-compatibility
       option)\MessageBreak but the package is already loaded!}}
    {\si@loadfile{#1}}}
```

`\si@emclash` 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@tempa` 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.

`\si@tempa` `\si@tempb` `\si@tempc`

```
\newcommand*{\si@expanddefault}[2]{%
\expandafter\ifx\expandafter\@empty\csname si@#1\endcsname\@empty
\else
  \def\si@tempb{default}
  \def\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}
  \def\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}
```

\si@tempa
\si@tempb

The locale files are loaded; here there is a need to check on both `loadlocales` and `loctolang`.

```
\ifx\@empty\si@loadlocales\@empty\else
  \@for\si@tempa:=\si@loadlocales\do{%
    \si@loc@load{\si@tempa}}
\fi
```

For `loctolang`.

```
\ifx\@empty\si@loctolang\@empty\else
  \def\si@tempa#1:#2\@nil{\si@loc@load{#1}}
  \@for\si@tempb:=\si@loctolang\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:=\si@loctolang\do{%
         \expandafter\si@tempa\si@tempb::\@nil}}
      {\si@log@warn{babel not loaded - option\MessageBreak
        loctolang ignored}}}
\fi
```

The very last job is to load a local configuration file, if one exists.

```
\IfFileExists{si.cfg}
  {\si@log@inf{Local configuration file found}%
   \InputIfFileExists{si.cfg}{}{}}
  {}
```

17 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).

17.1 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}
  [2008/02/20 v.06a 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 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 \kilogram As the base unit of mass is the kilogram, rather than the gram, a bit of extra work 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}
```

17.2 Derived units with specific names

\becquerel \coulomb \farad \Gray \hertz \henry \joule \katal \lumen \lux \newton Derived units with specific names and symbols are defined. Litre is an awkward one, but here the UK standard is used.

```
\ProvidesFile{si-named.cfg}
  [2008/02/20 v.06a SI Named units]
\newunit{\becquerel}{Bq}
\newunit{\coulomb}{C}
\newunit{\farad}{F}
\newunit{\Gray}{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.

```
\Ohm \ifsi@old@OHM
\pascal   \newunit{\Ohm}{\si@sym@Omega}
\siemens \else
\sievert   \ifsi@gensymb\else
```

\tesla To be on the safe side, use \provideunit.

```
\volt     \provideunit{\ohm}{\si@sym@Omega}
\watt   \fi
\weber \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.

```
\Celsius \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}
```

17.3 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}
  [2008/02/20 v.06a SI Prefixed units]
\si@requirecfgs{prefix,named,accepted,physical}
```

\picometre Extra distances.

```
\nanometre \newunit{\picometre}{\pico\metre}
\micrometre \newunit{\nanometre}{\nano\metre}
\millimetre \newunit{\micrometre}{\micro\metre}
\centimetre \newunit{\millimetre}{\milli\metre}
\decimetre \newunit{\centimetre}{\centi\metre}
\kilometre \newunit{\decimetre}{\deci\metre}
\newunit{\kilometre}{\kilo\metre}
```

\femtogram Extra masses.

```
\picogram \newunit{\femtogram}{\femto\gram}
\nanogram
\microgram
\milligram
```

```
\newunit{\picogram}{\pico\gram}
\newunit{\nanogram}{\nano\gram}
\newunit{\microgram}{\micro\gram}
\newunit{\milligram}{\milli\gram}
```

`\femtomole` Now some moles.

```
\newunit{\femtomole}{\femto\mole}
\newunit{\picomole}{\pico\mole}
\newunit{\nanomole}{\nano\mole}
\newunit{\micromole}{\micro\mole}
\newunit{\millimole}{\milli\mole}
```

`\attosecond` Prefixed seconds.

```
\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.

```
\newunit{\picoampere}{\pico\ampere}
\newunit{\nanoampere}{\nano\ampere}
\newunit{\microampere}{\micro\ampere}
\newunit{\milliampere}{\milli\ampere}
\newunit{\kiloampere}{\kilo\ampere}
```

`\millivolt` More electricity-related units.

```
\newunit{\millivolt}{\milli\volt}
\newunit{\kilovolt}{\nano\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}
```

`\kiloohm` 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
```

\microlitre
\millilitre
\cubicmetre

Volumes (unlike unitsdef, with litre and metre spelled correctly). Only \millilitre and \microlitre are defined as they are the two officially-sanctioned prefixes for the litre.

```
\cubiccentimetre
\centimetrecubed
\cubicmicrometre
\cubicmillimetre
\cubicdecimetre
2243 \newunit{\microlitre}{\micro\litre}
2244 \newunit{\millilitre}{\milli\litre}
2245 \newunit{\cubicmetre}{\metre\cubed}
2246 \newunit{\cubiccentimetre}{\centi\metre\cubed}
2247 \newunit{\centimetrecubed}{\centi\metre\cubed}
2248 \newunit{\cubicmicrometre}{\micro\metre\cubed}
2249 \newunit{\cubicmillimetre}{\milli\metre\cubed}
2250 \newunit{\cubicdecimetre}{\cubic\deci\metre}
```

\squaremetre
\squarecentimetre

Areas, with metre spelled corrected; \are and \hectare are in the "temporarily accepted" file.

```
\squarekilometre
2251 \newunit{\squaremetre}{\Square\metre}
2252 \newunit{\squarecentimetre}{\Square\centi\metre}
2253 \newunit{\squarekilometre}{\Square\kilo\metre}
```

\millijoule

Some energy is needed by now!

```
\kilojoule
\megajoule
\millielectronvolt
\kiloelectronvolt
\megaelectronvolt
\gigaelectronvolt
\teraelectronvolt
2254 \newunit{\millijoule}{\milli\joule}
2255 \newunit{\kilojoule}{\kilo\joule}
2256 \newunit{\megajoule}{\mega\joule}
2257 \newunit{\millielectronvolt}{\milli\electronvolt}
2258 \newunit{\kiloelectronvolt}{\kilo\electronvolt}
2259 \newunit{\megaelectronvolt}{\mega\electronvolt}
2260 \newunit{\gigaelectronvolt}{\giga\electronvolt}
2261 \newunit{\teraelectronvolt}{\tera\electronvolt}
```

\millihertz

Frequencies.

```
\kilohertz
\megahertz
\gigahertz
\terahertz
2262 \newunit{\millihertz}{\milli\hertz}
2263 \newunit{\kilohertz}{\kilo\hertz}
2264 \newunit{\megahertz}{\mega\hertz}
2265 \newunit{\gigahertz}{\giga\hertz}
2266 \newunit{\terahertz}{\tera\hertz}
```

\millinewton

A few more from various areas.

```
\kilonewton
\hectopascal
\megabecquerel
\millisievert
2267 \newunit{\millinewton}{\milli\newton}
2268 \newunit{\kilonewton}{\kilo\newton}
2269 \newunit{\hectopascal}{\hecto\pascal}
2270 \newunit{\megabecquerel}{\mega\becquerel}
2271 \newunit{\millisievert}{\milli\sievert}
```

17.4 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.

```
\micA
\mA
\kA
2272 \ProvidesFile{si-abbr.cfg}
2273   [2008/02/20 v.06a Abbreviated units]
2274 \si@requirecfgs{prefix,named,accepted,physical}
2275 \newunit{\pA}{\pico\ampere}
2276 \newunit{\nA}{\nano\ampere}
2277 \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.

```
\newunit{\ml}{\milli\litre}
\newunit{\micl}{\micro\litre}
\newunit{\cmc}{\centi\metre\cubed}
\newunit{\dmc}{\deci\metre\cubed}
```

`\kg` Masses.

```
\newunit{\kg}{\kilo\gram}
```

`\pg` `\nanog` There is a name clash with babel here in French; hopefully there will not be too many complaints.

`\micg` `\mg` `\amu`

```
\provideunit{\fg}{\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}
```

`\picom` Lengths.

```
\newunit{\picom}{\pico\metre}
\newunit{\nm}{\nano\metre}
\newunit{\micm}{\micro\metre}
```

`\cm`

`\dm`

`\km`

```
\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 The letter class (and others) define \ps for postscripts, so \provideunit is best
\mics here.

```
\provideunit{\ps}{\pico\second}
\newunit{\ns}{\nano\second}
\newunit{\mics}{\micro\second}
\newunit{\ms}{\milli\second}
```

17.5 Additional (temporary) SI units

\angstrom Some units are "temporarily" acceptable for use in the SI system. These are
\are defined here, although some are in very general use.

```
\ProvidesFile{si-addn.cfg}
  [2008/02/20 v.06a SI Additional units]
\newunit{\angstrom}{\si@sym@ringA}
\newunit{\are}{a}
\newunit{\hectare}{\hecto\are}
\newunit{\barn}{b}
\newunit{\BAR}{bar}
\newunit{\millibar}{\milli\BAR}
\newunit{\gal}{Gal}
\newunit{\curie}{Ci}
\newunit{\roentgen}{R}
\newunit{\rad}{rad}
\newunit{\rem}{rem}
```

17.6 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.

```
\ProvidesFile{si-accepted.cfg}
  [2008/02/20 v.06a SI accepted units]
\newunit{\minute}{min}
\newunit{\hour}{h}
\newunit{\Day}{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{\tonne}{t}
\newunit{\neper}{Np}
\newunit{\bel}{B}
\newunit{\percent}{\%}
```

17.7 Units based on physical measurements

\si@eVspacea
\si@eVspaceb
\electronvolt
\atomicmassunit
\atomicmass
\dalton

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}
  [2008/02/20 v.06a 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}
\newunit{\dalton}{Da}
```

18 Additional configurations

To provide flexibility for people in specific areas, specialised units can be set up. These are then stored separately to ease use.

18.1 Synthetic chemistry

\mmHg
\molar
\Molar
\torr

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}
  [2008/02/20 v.06a Units for synthetic chemists]
\si@requirecfgs{prefix}
\newunit{\mmHg}{mmHg}
\newunit{\molar}{\mole\per\cubic\deci\metre}
\newunit{\Molar}{\textsc{m}}
\newunit{\torr}{Torr}
```

18.2 High-energy physics

The units here basically add the units from the \hepunits package which are not defined elsewhere here. It is not entirely clear if \mrad refers to radians or rad: feedback would be welcome. This set of commands is not in the emulation block as it does *not* seek to emulate hepunits: that package is a blot-on to SIunits. The units here have the same name as those in hepunits but stick with the new package interface.

```
\ProvidesFile{si-hep.cfg}
  [2008/02/20 v.06a Units for high-energy physics]
\si@requirecfgs{prefix,named}
```

\micron The first units are not specific to high-energy physics, but are not defined elsewhere in si.
\mrad
\gauss
```
\newunit{\micron}{\micro\metre}
\newunit{\mrad}{\milli\radian}
\newunit{\gauss}{G}
```

\nanobarn Various prefixed barns
\picobarn \femtobarn \attobarn \zeptobarn \yoctobarn
```
\newunit{\nanobarn}{\nano\barn}
\newunit{\picobarn}{\pico\barn}
\newunit{\femtobarn}{\femto\barn}
\newunit{\attobarn}{\atto\barn}
\newunit{\zeptobarn}{\zepto\barn}
\newunit{\yoctobarn}{\yocto\barn}
```

\invbarn Inverses barn units.
\invnanobarn \invpicobarn \invfemtobarn \invattobarn \invzeptobarn \invyoctobarn
```
\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}
```

\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}
```

\clight The speed of light is used in units for the area, although of course it is not strictly a unit.
```
\newunit{\clight}{\ensuremath{\mathnormal{c}}}
```

\inveV The inverse of an electron-volt, plus prefixes.
\minveV \minveV \kinveV \MinveV \GinveV \TinveV
```
\newunit{\inveV}{\per\electronvolt}
\newunit{\minveV}{\milli\per\electronvolt}
\newunit{\kinveV}{\kilo\per\electronvolt}
\newunit{\MinveV}{\mega\per\electronvolt}
\newunit{\GinveV}{\giga\per\electronvolt}
\newunit{\TinveV}{\tera\per\electronvolt}
```

\eVoverc Some combinations of electron-volts and the speed of light. As these are called
\eVovercsq `over`, they are set with a slash. The `eVcorrb` values have been set for Computer Modern.
```
\newunit[per=slash,eVcorrb=0.6ex]{\eVoverc}
```

```
2408   {\electronvolt\per\clight}
2409 \newunit[per=slash,eVcorrb=0.6ex]{\eVovercsq}
2410   {\electronvolt\per\Square\clight}
```

\meVoverc Prefixed combinations, first of the speed of light.

```
\keVoverc2411 \newunit[per=slash,eVcorrb=0.6ex]{\meVoverc}
\MeVoverc2412   {\milli\electronvolt\per\clight}
\GeVoverc2413 \newunit[per=slash,eVcorrb=0.6ex]{\keVoverc}
\TeVoverc2414   {\kilo\electronvolt\per\clight}
          2415 \newunit[per=slash,eVcorrb=0.6ex]{\MeVoverc}
          2416   {\mega\electronvolt\per\clight}
          2417 \newunit[per=slash,eVcorrb=0.6ex]{\GeVoverc}
          2418   {\giga\electronvolt\per\clight}
          2419 \newunit[per=slash,eVcorrb=0.6ex]{\TeVoverc}
          2420   {\tera\electronvolt\per\clight}
```

\meVovercsq Then of the square.

```
\keVovercsq2421 \newunit[per=slash,eVcorrb=0.6ex]{\meVovercsq}
\MeVovercsq2422   {\milli\electronvolt\per\Square\clight}
\GeVovercsq2423 \newunit[per=slash,eVcorrb=0.6ex]{\keVovercsq}
\TeVovercsq2424   {\kilo\electronvolt\per\Square\clight}
          2425 \newunit[per=slash,eVcorrb=0.6ex]{\MeVovercsq}
          2426   {\mega\electronvolt\per\Square\clight}
          2427 \newunit[per=slash,eVcorrb=0.6ex]{\GeVovercsq}
          2428   {\giga\electronvolt\per\Square\clight}
          2429 \newunit[per=slash,eVcorrb=0.6ex]{\TeVovercsq}
          2430   {\tera\electronvolt\per\Square\clight}
```

18.3 Binary units

\kibi The binary units, as specified by the IEC and made available by Slunits. First, the \mebi binary prefixes.

```
\gibi2431 \ProvidesFile{si-binary.cfg}
\tebi2432   [2008/02/20 v.06a Binary units]
\pebi2433 \newprefix{\kibi}{10}{Ki}
\exbi2434 \newprefix{\mebi}{20}{Mi}
     2435 \newprefix{\gibi}{30}{Gi}
     2436 \newprefix{\tebi}{40}{Ti}
     2437 \newprefix{\pebi}{50}{Pi}
     2438 \newprefix{\exbi}{60}{Ei}
```

\bit Now the units.

```
\byte2439 \newunit{\bit}{bit}
     2440 \newunit{\byte}{B}
```

19 Loadable locales

Some short files to provide the correct settings for various places.

19.1 United Kingdom

This is also used for the USA, and is the default.

```
\ProvidesFile{si-UK.cfg}
  [2008/02/20 v.06a UK locale]
\sisetup{
  unitsep=thin,
  expproduct=times,
  valuesep=thin,
  decimalsign=fullstop,
  digitsep=thin,
  sepfour=false}
```

19.2 United States

The same as for the UK.

```
\ProvidesFile{si-USA.cfg}
  [2008/02/20 v.06a USA locale]
\sisetup{
  unitsep=thin,
  expproduct=times,
  valuesep=thin,
  decimalsign=fullstop,
  digitsep=thin,
  sepfour=false}
```

19.3 Germany

Germany, hopefully.

```
\ProvidesFile{si-germany.cfg}
  [2008/02/20 v.06a Germany locale]
\sisetup{
  unitsep=cdot,
  valuesep=thin,
  decimalsign=comma,
  expproduct=cdot,
  digitsep=thin,
  sepfour=false}
```

19.4 South Africa

Taken from sistyle.

```
\ProvidesFile{si-south-africa.loc}
  [2008/02/20 v.06a UK Locale]
\sisetup{
  unitsep=cdot,
  valuesep=thin,
  expproduct=times,
  decimalsign=comma,
  digitsep=thin,
  sepfour=false}
```

20 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.

20.1 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}
  [2008/02/20 v.06a Emulation of units]
\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` 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 si rather than units they should expect more checking of input. As the units package uses the current mode, this has to be detected.
`\unit[⟨num⟩]{⟨unit⟩}`

```
\DeclareRobustCommand*{\unit}[2][]{%
  \ifmmode
    \SI{#1}{#2}%
  \else
    \SI[obeyfamily,obeyitalic]{#1}{#2}%
  \fi}
```

`\unitfrac` `\unitfrac` is a bit more of a hack.
`\unitfrac[⟨num⟩]{⟨numerator⟩}{⟨denominator⟩}`

```
\DeclareRobustCommand*{\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}
```

20.2 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}
  [2008/02/20 v.06a Emulation of unitsdef]
\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.

```
\si@ifloaded{units}{}
  {\InputIfFileExists{\si@fileprefix units.\si@cfgextension}
     {}
     {\si@log@err{Could not load \si@fileprefix
       units.\si@cfgextension}
       {The file \si@fileprefix units.\si@cfgextension is
         required to emulate\MessageBreak
        unitsdef, but cannot be found\MessageBreak
        Is the si package properly installed?}
       \endinput}}
```

The unitsdef package loads some packages that si 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}
\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}}
\si@opt@choicekey{valuesep}{space,thin,med,medium,thick,none}
    {\renewcommand*\unitvaluesep\@nameuse{si@fix@##1}}
    {\renewcommand*\unitvaluesep{##1}}
```

`\unitsignonly` `\ilu` `\arc` Some rather straight-forward definitions, with just a bit of fun to get the spacing correct.

```
\DeclareRobustCommand*{\unitsignonly}{\unitsym}
\DeclareRobustCommand*{\ilu}[2][]{%
  \begingroup
    #1\unitvaluesep%
    \unit{#2}%
  \endgroup}
\DeclareRobustCommand*{\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 si uses `\csname` ... `\endcsname`.

```
\newcommand*{\unitSIdef}{\upshape}
\newcommand*{\si@unitSIdef}{\unitSIdef\selectfont}
\sisetup{textdefault=si@unitSIdef,textnumdefault=si@unitSIdef}
```

`\per` Rather awkwardly, unitsdef uses `\per` in a different way to si.

```
\DeclareRobustCommand*{\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 si 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
```

`\meter` `\liter` `\ton` `\days` For some reason, unitsdef spells metre and litre incorrectly (the names have an official spelling). Tonne is also spelled as "ton", which is wrong in the UK at least (1 ton = 40 cwt = 2240 lb!)

```
\newunit{\meter}{\metre}
\newunit{\liter}{L}
\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}
    \renewunit{\micl}{\micro\liter}
    \renewunit{\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.

```
\si@requirecfgs{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
```

20.3 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}
  [2008/02/20 v.06a Emulation of sistyle]
\sisetup{%
  sepfour=true,
  obeyfamily,
  obeyitalic=true,
  numsign=+-,
  numextra={},
  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 `*`.

```
\DeclareRobustCommand{\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@decimalsign}}
```

`\SIgroupfourtrue` `\SIgroupfourfalse` Switches for grouping four characters.

```
\newcommand*{\SIgroupfourtrue}{\sisetup{sepfour=true}}
\newcommand*{\SIgroupfourfalse}{\sisetup{sepfour=false}}
```

`\SIunitsep` `\SIunitspace` `\SIunitdot` Whatever is given here is passed through to `\sisetup`.

```
\newcommand*{\SIunitsep}[1]{\sisetup{valuesep={#1}}}
\newcommand*{\SIunitspace}[1]{\sisetup{unitspace={#1}}}
\newcommand*{\SIunitdot}[1]{\sisetup{unitsep={#1}}}
```

`\SIdecimalsign` `\SIthousandsep` `\SIproductsign` The same is true here, with the appropriate translation.

```
\newcommand*{\SIdecimalsign}[1]{\sisetup{decimalsign={#1}}}
\newcommand*{\SIthousandsep}[1]{\sisetup{digitsep={#1}}}
\newcommand*{\SIproductsign}[1]{\sisetup{exppproduct={#1}}}
```

`\si@sis@savefont` The font definitions need a bit of extra work doing. As both settings here have `@` as a letter, all should be fine.

`\si@sis@savefont{`⟨*setting*⟩`}{`⟨*argument*⟩`}`

```
\newcommand{\si@sis@savefont}[2]{%
  \@namedef{si@sis@#1}{#2}%
  \sisetup{#1=si@sis@#1}}
```

`\SImathrm` `\SImathsf` `\SImathtt` The font control macros have to ensure that a macro name is passed to `\sisetup`.

```
\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` `\SIdefaultNfam` `\SIdefaultTfam` The same for the default keys.

```
\newcommand*{\SIdefaultMfam}[1]{\si@sis@savefont{mathdefault}{#1}}
\newcommand*{\SIdefaultNfam}[1]{\si@sis@savefont{mathnumdefault}{#1}}
\newcommand*{\SIdefaultTfam}[1]{\si@sis@savefont{textdefault}{#1}}
```

`\ensureupmath` The `\ensureupmath` command guarantees processing by the font-matching system. The argument cannot be processed here, so care is needed.

```
\DeclareRobustCommand*{\ensureupmath}[1]{%
  \begingroup
    \sisetup{mode=maths,obeyitalic=false}%
    \si@out@text{#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
\SIstyleToLang
\si@sis@addtolocale
```
\newcommand*{\SIstyle}[1]{\sisetup{locale=#1}}
\newcommand*{\SIstyleToLang}[2]{\sisetup{loctolang=#1:#2}}
\newcommand*{\AddToSIstyle}{%
  \si@switchfalse
  \@ifstar
    {\si@switchtrue
     \si@sis@addtolocale}
    {\si@sis@addtolocale}}
\newcommand*{\si@sis@addtolocale}[2]{%
    \ifsi@switch
       \expandafter\let\csname si@loc@#1@extra\endcsname\relax
     \fi
  \addtolocale{#1}{#2}}
```

20.4 SIunits

SIunits emulation starts in much the same way.

```
\ProvidesFile{si-SIunits.cfg}
  [2008/02/20 v.06a Emulation of SIunits]
\sisetup{
  unitsep=thick,
  valuesep=thick,
  prefixproduct=\si@valuesep}
\si@requirecfgs{prefix,named,accepted,physical}
```

\reciprocal A few very simple translations, using the internal version of \per to allow changes of output style.
\rp
\per
\usk
\power
\rpsquare
\rpcubic
\fourth
\rpfourth
```
\newcommand*{\reciprocal}{\sisetup{per=reciprocal}\si@per}
\let\rp\reciprocal
\renewcommand*{\per}{\sisetup{per=slash}\si@per}
\newcommand*{\usk}{}
\newcommand*{\power}[1]{#1\tothe}
\newcommand*{\rpsquare}{\sisetup{per=reciprocal}\si@per\Square}
\newcommand*{\rpcubic}{\sisetup{per=reciprocal}\si@per\cubic}
\newpower{\fourth}{4}
\newcommand*{\rpfourth}{\sisetup{per=reciprocal}\si@per\fourth}
```

\rpsquared Here, some low-level switch changing is needed.
\rpcubed
```
\newcommand*{\rpsquared}{%
  \sisetup{per=reciprocal}\si@unt@pertrue\si@unt@perseentrue\squared}
\newcommand*{\rpcubed}{\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@tempa
\si@siu@setup
```
\newcommand*{\SIsetup}[1]{%
  \@for\si@tempa:=#1\do{%
    \@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 si 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` `\bbar` `\dday` `\liter` `\rperminute` `\ton` `\degreecelsius` The unit macros defined by Slunits that are not defined by si (by default).

```
\newunit[valuesep=none]{\arcminute}{\si@sym@minute}
\newunit[valuesep=none]{\arcsecond}{\si@sym@second}
\newunit{\bbar}{bar}
\newunit{\dday}{day}
\newunit{\liter}{L}
\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}}
```

`\graypersecond` `\graypersecondnp` `\metrepersquaresecond` `\metrepersquaresecondnp` `\joulepermole` `\joulepermolenp` `\molepercubicmetre` `\molepercubicmetrenp` `\radianpersquaresecond` `\radianpersquaresecondnp` `\ogramsquaremetrepersecond` Slunits provides lots of macros with rather long names, which are not really needed with si. However, they have to be defined somewhere. There are a lot of them, so a few are tackled at a time.

```
\addunit{\graypersecond}{\gray\per\second}
\addunit{\graypersecondnp}{\gray\reciprocal\second}
\addunit{\metrepersquaresecond}{\metre\per\second\squared}
\addunit{\metrepersquaresecondnp}{\metre\second\rpsquared}
\addunit{\joulepermole}{\joule\per\mole}
\addunit{\joulepermolenp}{\joule\reciprocal\mole}
\addunit{\molepercubicmetre}{\mole\per\cubic\metre}
\addunit{\molepercubicmetrenp}{\mole\rpcubic\metre}
\addunit{\radianpersquaresecond}{\radian\per\second\squared}
```

```
\addunit{\radianpersquaresecondnp}{\radian\second\rpsquared}
\addunit{\kilogramsquaremetrepersecond}
  {\kilogram\usk\squaremetre\per\second}
```

ramsquaremetrepersecondnp Some more.

```
         \radianpersecond \addunit{\kilogramsquaremetrepersecondnp}
       \radianpersecondnp   {\kilogram\usk\squaremetre\reciprocal\second}
\squaremetrepercubicmetre \addunit{\radianpersecond}{\radian\per\second}
quaremetrepercubicmetrenp \addunit{\radianpersecondnp}{\radian\reciprocal\second}
      \katalpercubicmetre \addunit{\squaremetrepercubicmetre}{\squaremetre\per\cubic\metre}
    \katalpercubicmetrenp \addunit{\squaremetrepercubicmetrenp}{\squaremetre\rpcubic\metre}
         \coulombpermol   \addunit{\katalpercubicmetre}{\katal\per\cubic\metre}
       \coulombpermolnp   \addunit{\katalpercubicmetrenp}{\katal\rpcubic\metre}
    \amperepersquaremetre \addunit{\coulombpermol}{\coulomb\per\mole}
  \amperepersquaremetrenp \addunit{\coulombpermolnp}{\coulomb\reciprocal\mole}
                          \addunit{\amperepersquaremetre}{\ampere\per\squaremetre}
                          \addunit{\amperepersquaremetrenp}{\ampere\rpsquare\metre}
```

\kilogrampercubicmetre Some more.

```
 \kilogrampercubicmetrenp \addunit{\kilogrampercubicmetre}{\kilogram\per\cubic\metre}
quaremetrepernewtonsecond \addunit{\kilogrampercubicmetrenp}{\kilogram\rpcubic\metre}
aremetrepernewtonsecondnp \addunit{\squaremetrepernewtonsecond}
            \pascalsecond   {\squaremetre\per\newton\second}
    \coulombpercubicmetre \addunit{\squaremetrepernewtonsecondnp}
  \coulombpercubicmetrenp   {\squaremetre\reciprocal\newton\reciprocal\second}
      \amperemetresecond \addunit{\pascalsecond}{\pascal\second}
           \voltpermetre \addunit{\coulombpercubicmetre}{\coulomb\per\cubic\metre}
         \voltpermetrenp \addunit{\coulombpercubicmetrenp}{\coulomb\rpcubic\metre}
   \coulombpersquaremetre \addunit{\amperemetresecond}{\ampere\metre\second}
                          \addunit{\voltpermetre}{\volt\per\metre}
                          \addunit{\voltpermetrenp}{\volt\reciprocal\metre}
                          \addunit{\coulombpersquaremetre}{\coulomb\per\squaremetre}
```

\coulombpersquaremetrenp Some more.

```
           \faradpermetre \addunit{\coulombpersquaremetrenp}{\coulomb\rpsquare\metre}
         \faradpermetrenp \addunit{\faradpermetre}{\farad\per\metre}
                \ohmmetre \addunit{\faradpermetrenp}{\farad\reciprocal\metre}
            \kilowatthour \addunit{\ohmmetre}{\ohm\metre}
     \wattpersquaremetre  \addunit{\kilowatthour}{\kilo\watt\hour}
   \wattpersquaremetrenp  \addunit{\wattpersquaremetre}{\watt\per\squaremetre}
    \joulepersquaremetre  \addunit{\wattpersquaremetrenp}{\watt\rpsquare\metre}
  \joulepersquaremetrenp  \addunit{\joulepersquaremetre}{\joule\per\squaremetre}
    \newtonpercubicmetre  \addunit{\joulepersquaremetrenp}{\joule\rpsquare\metre}
  \newtonpercubicmetrenp  \addunit{\newtonpercubicmetre}{\newton\per\cubic\metre}
                          \addunit{\newtonpercubicmetrenp}{\newton\rpcubic\metre}
```

\newtonperkilogram Some more.

```
    \newtonperkilogramnp \addunit{\newtonperkilogram}{\newton\per\kilogram}
        \jouleperkelvin  \addunit{\newtonperkilogramnp}{\newton\reciprocal\kilogram}
      \jouleperkelvinnp  \addunit{\jouleperkelvin}{\joule\per\kelvin}
      \jouleperkilogram  \addunit{\jouleperkelvinnp}{\joule\reciprocal\kelvin}
    \jouleperkilogramnp  \addunit{\jouleperkilogram}{\joule\per\kilogram}
    \coulombperkilogram  \addunit{\jouleperkilogramnp}{\joule\reciprocal\kilogram}
  \coulombperkilogramnp  \addunit{\coulombperkilogram}{\coulomb\per\kilogram}
  \squaremetrepersecond
\squaremetrepersecondnp
quaremetrepersquaresecond
```

```
\addunit{\coulombperkilogramnp}{\coulomb\reciprocal\kilogram}
\addunit{\squaremetrepersecond}{\squaremetre\per\second}
\addunit{\squaremetrepersecondnp}{\squaremetre\reciprocal\second}
\addunit{\squaremetrepersquaresecond}
  {\squaremetre\per\second\squared}
```

aremetrepersquaresecondnp Some more.

```
\addunit{\squaremetrepersquaresecondnp}
  {\squaremetre\second\rpsquared}
\addunit{\kilogrammetrepersecond}{\kilogram\metre\per\second}
\addunit{\kilogrammetrepersecondnp}
  {\kilogram\metre\reciprocal\second}
\addunit{\candelapersquaremetre}{\candela\per\squaremetre}
\addunit{\candelapersquaremetrenp}{\candela\rpsquare\metre}
\addunit{\amperepermetre}{\ampere\per\metre}
\addunit{\amperepermetrenp}{\ampere\reciprocal\metre}
\addunit{\joulepertesla}{\joule\per\tesla}
\addunit{\joulepertesla np}{\joule\reciprocal\tesla}
\addunit{\henrypermetre}{\henry\per\metre}
\addunit{\henrypermetrenp}{\henry\reciprocal\metre}
```

\kilogrampersecond Some more.

```
\addunit{\kilogrampersecond}{\kilogram\per\second}
\addunit{\kilogrampersecondnp}{\kilogram\reciprocal\second}
\addunit{\kilogrampersquaremetresecond}
  {\kilogram\per\squaremetre\second}
\addunit{\kilogrampersquaremetresecondnp}
  {\kilogram\rpsquare\metre\reciprocal\second}
\addunit{\kilogrampersquaremetre}{\kilogram\per\squaremetre}
\addunit{\kilogrampersquaremetrenp}{\kilogram\rpsquare\metre}
\addunit{\kilogrampermetre}{\kilogram\per\metre}
\addunit{\kilogrampermetrenp}{\kilogram\reciprocal\metre}
\addunit{\joulepermolekelvin}{\joule\per\mole\kelvin}
\addunit{\joulepermolekelvinnp}
  {\joule\reciprocal\mole\reciprocal\kelvin}
\addunit{\kilogramperkilomole}{\kilogram\per\kilo\mole}
```

\kilogramperkilomolenp Some more.

```
\addunit{\kilogramperkilomolenp}{\kilogram\kilo\reciprocal\mole}
\addunit{\kilogramsquaremetre}{\kilogram\squaremetre}
\addunit{\kilogramsquaremetrenp}{\kilogramsquaremetre}
\addunit{\kilogrammetrepersquaresecond}
  {\kilogram\metre\per\second\squared}
\addunit{\kilogrammetrepersquaresecondnp}
  {\kilogram\metre\second\rpsquared}
\addunit{\newtonpersquaremetre}{\newton\per\squaremetre}
\addunit{\newtonpersquaremetrenp}{\newton\rpsquare\metre}
\addunit{\persquaremetresecond}{1\per\squaremetre\second}
\addunit{\persquaremetresecondnp}{\rpsquare\metre\reciprocal\second}
\addunit{\wattperkilogram}{\watt\per\kilogram}
\addunit{\wattperkilogramnp}{\watt\reciprocal\kilogram}
```

\wattpercubicmetre Some more.

```
\addunit{\wattpercubicmetre}{\watt\per\cubic\metre}
ttpersquaremetresteradian
persquaremetresteradiannp
  \jouleperkilogramkelvin
\jouleperkilogramkelvinnp
  \squaremetreperkilogram
\rpsquaremetreperkilogram
   \cubicmetreperkilogram
 \rpcubicmetreperkilogram
          \newtonpermetre
```

```
\addunit{\wattpercubicmetrenp}{\watt\rpcubic\metre}
\addunit{\wattpersquaremetresteradian}{\watt\per\squaremetre\steradian}
\addunit{\wattpersquaremetresteradiannp}
  {\watt\rpsquare\metre\rp\steradian}
\addunit{\jouleperkilogramkelvin}{\joule\per\kilogram\kelvin}
\addunit{\jouleperkilogramkelvinnp}
  {\joule\reciprocal\kilogram\reciprocal\kelvin}
\addunit{\squaremetreperkilogram}{\squaremetre\per\kilogram}
\addunit{\rpsquaremetreperkilogram}
  {\squaremetre\reciprocal\kilogram}
\addunit{\cubicmetreperkilogram}{\cubic\metre\per\kilogram}
\addunit{\rpcubicmetreperkilogram}
  {\cubic\metre\reciprocal\kilogram}
\addunit{\newtonpermetre}{\newton\per\metre}
```

\newtonpermetrenp Some more.

```
\addunit{\newtonpermetrenp}{\newton\reciprocal\metre}
\addunit{\wattpermetrekelvin}{\watt\per\metre\kelvin}
\addunit{\wattpermetrekelvinnp}
  {\watt\reciprocal\metre\reciprocal\kelvin}
\addunit{\newtonmetre}{\newton\metre}
\addunit{\newtonmetrenp}{\newtonmetre}
\addunit{\squaremetrepercubicsecond}{\squaremetre\per\cubic\second}
\addunit{\squaremetrepercubicsecondnp}
  {\squaremetre\rpcubic\second}
\addunit{\metrepersecond}{\metre\per\second}
\addunit{\metrepersecondnp}{\metre\reciprocal\second}
\addunit{\joulepercubicmetre}{\joule\per\cubicmetre}
\addunit{\joulepercubicmetrenp}{\joule\rpcubic\metre}
```

Last block.

```
\addunit{\kilogrampercubicmetrecoulomb}
  {\kilogram\per\cubic\metre\coulomb}
\addunit{\kilogrampercubicmetrecoulombnp}
  {\kilogram\rpcubic\metre\reciprocal\coulomb}
\addunit{\cubicmetrepersecond}{\cubicmetre\per\second}
\addunit{\rpcubicmetrepersecond}{\cubicmetre\reciprocal\second}
\addunit{\kilogrampersecondcubicmetre}
  {\kilogram\per\second\cubicmetre}
\addunit{\kilogrampersecondcubicmetrenp}
  {\kilogram\reciprocal\second\rpcubic\metre}
```

\yoctod The prefixes giving numerical output need a trick. First the small values.

```
\newunit{\yoctod}{\si@prefixnumtrue\yocto}
\newunit{\zeptod}{\si@prefixnumtrue\zepto}
\newunit{\attod}{\si@prefixnumtrue\atto}
\newunit{\femtod}{\si@prefixnumtrue\femto}
\newunit{\picod}{\si@prefixnumtrue\pico}
\newunit{\nanod}{\si@prefixnumtrue\nano}
\newunit{\microd}{\si@prefixnumtrue\micro}
\newunit{\millid}{\si@prefixnumtrue\milli}
\newunit{\centid}{\si@prefixnumtrue\centi}
```

\decad The the larger ones.

\dekad
\hectod
\kilod
\megad
\gigad
\terad
\petad
\exad
\zettad
\yottad

```
\newunit{\decad}{\si@prefixnumtrue\deca}
\newunit{\dekad}{\si@prefixnumtrue\deka}
\newunit{\hectod}{\si@prefixnumtrue\hecto}
\newunit{\kilod}{\si@prefixnumtrue\kilo}
\newunit{\megad}{\si@prefixnumtrue\mega}
\newunit{\gigad}{\si@prefixnumtrue\giga}
\newunit{\terad}{\si@prefixnumtrue\tera}
\newunit{\petad}{\si@prefixnumtrue\peta}
\newunit{\exad}{\si@prefixnumtrue\exa}
\newunit{\zettad}{\si@prefixnumtrue\zetta}
\newunit{\yottad}{\si@prefixnumtrue\yotta}
```

\kibid The binary versions need a little more work.

\mebid \gibid \tebid \pebid \exbid

```
\newunit{\kibid}{%
  \si@prefixnumtrue\let\si@prefixpower\si@fix@two\kibi}
\newunit{\mebid}{%
  \si@prefixnumtrue\let\si@prefixpower\si@fix@two\mebi}
\newunit{\gibid}{%
  \si@prefixnumtrue\let\si@prefixpower\si@fix@two\gibi}
\newunit{\tebid}{%
  \si@prefixnumtrue\let\si@prefixpower\si@fix@two\tebi}
\newunit{\pebid}{%
  \si@prefixnumtrue\let\si@prefixpower\si@fix@two\pebi}
\newunit{\exbid}{%
  \si@prefixnumtrue\let\si@prefixpower\si@fix@two\exbi}
```

\derradian The derived units may need to be defined.

\dersteradian \derhertz \dernewton \derpascal \derjoule \derwatt \dercoulomb \dervolt \derfarad \derohm

```
\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!

\derweber \dertesla \derhenry \dercelsius \derlumen \derlux \derbecquerel \dergray \dersievert \derkatal

```
  \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}
    {\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}}
```

Part V
Notes

21 Change History

22 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.

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

23 References

[1] http://physics.nist.gov/cuu/Units/rules.html.