siunitx – A comprehensive (SI) units package*

Joseph Wright†

Released 2025-01-14

Contents

*This file describes v3.4.1, last revised 2025-01-14.

†E-mail: joseph@texdev.net

Abstract

Physical quantities have both numbers and units, and each physical quantity should be expressed as the product of a number and a unit. Typesetting physical quantities requires care to ensure that the combined mathematical meaning of the number–unit combination is clear. In particular, the SI units system lays down a consistent set of units with rules on how these are to be used. However, different countries and publishers have differing conventions on the exact appearance of numbers (and units). The siunitx package provides a set of tools for authors to typeset quantities in a consistent way. The package has an extended set of configuration options which make it possible to follow varying typographic conventions with the same input syntax. The package includes automated processing of numbers and units, and the ability to control tabular alignment of numbers.

1 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 Génrale des Poids et Mesures* (CGPM): this has resulted in the *Système International d'Unités* (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.

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

2 siunitx for the impatient

The package provides the user macros:

- `\ang[`⟨*options*⟩`]{`⟨*angle*⟩`}`
- `\num[`⟨*options*⟩`]{`⟨*number*⟩`}`
- `\unit[`⟨*options*⟩`]{`⟨*unit*⟩`}`
- `\qty[`⟨*options*⟩`]{`⟨*number*⟩`}{`⟨*unit*⟩`}`
- `\numlist[`⟨*options*⟩`]{`⟨*numbers*⟩`}`
- `\numproduct[`⟨*options*⟩`]{`⟨*numbers*⟩`}`
- `\numrange[`⟨*options*⟩`]{`⟨*numbers*⟩`}{`⟨*number2*⟩`}`
- `\qtylist[`⟨*options*⟩`]{`⟨*numbers*⟩`}{`⟨*unit*⟩`}`
- `\qtyproduct[`⟨*options*⟩`]{`⟨*numbers*⟩`}{`⟨*unit*⟩`}`
- `\qtyrange[`⟨*options*⟩`]{`⟨*number1*⟩`}{`⟨*number2*⟩`}{`⟨*unit*⟩`}`

- `\complexnum[`⟨*options*⟩`]{`⟨*number*⟩`}`
- `\complexqty[`⟨*options*⟩`]{`⟨*number*⟩`}{`⟨*unit*⟩`}`
- `\sisetup{`⟨*options*⟩`}`
- `\tablenum[`⟨*options*⟩`]{`⟨*number*⟩`}`

plus the `S` column type for decimal alignments and units in tabular environments. These user macros and column types are designed for typesetting numbers and units with control of appearance and with intelligent processing.

Numbers are processed with understanding of exponents, or using additional commands for products and complex numbers.

12 345.678 90
0.3×10^{45}
$1 \pm 2\mathrm{i}$
$1.654 \times 2.34 \times 3.430$

```
\num{12345,67890}  \\
\num{.3e45}        \\
\complexnum{1+-2i} \\
\numproduct{1.654 x 2.34 x 3.430}
```

The unit system can interpret units given as text to be used directly or as macro-based units. In the latter case, different formatting is possible.

```
\unit{kg.m.s^{-1}}                \\
\unit{\kilogram\metre\per\second} \\
\unit[per-mode = symbol]
  {\kilogram\metre\per\second}  \\
\unit[per-mode = symbol]
  {\kilogram\metre\per\ampere\per\second}
```

$\mathrm{kg\,m\,s^{-1}}$
$\mathrm{kg\,m\,s^{-1}}$
$\mathrm{kg\,m/s}$
$\mathrm{kg\,m/(A\,s)}$

Simple lists and ranges of numbers can be handled.

```
\numlist{10;20;30}                     \\
\qtylist{0.13;0.67;0.80}{\milli\metre} \\
\numrange{10}{20}                      \\
\qtyrange{0.13}{0.67}{\milli\metre}
```

10, 20 and 30
0.13 mm, 0.67 mm and 0.80 mm
10 to 20
0.13 mm to 0.67 mm

A wide range of options are available to control the behavior of the package. For example, with the standard settings all text is typeset in the current upright math font. This can be adjusted to use text mode, follow various aspects of the surrounding formatting, *etc*. Similarly, the standard settings are based around an English-speaking locale, but can be adjusted to follow the traditions of other areas.

3 Using the siunitx package

3.1 Numbers

\num `\num[⟨options⟩]{⟨number⟩}`

Numbers are automatically formatted by the `\num` macro. This takes one optional argument, ⟨*options*⟩, and one mandatory one, ⟨*number*⟩. The contents of ⟨*number*⟩ are automatically formatted. The formatter removes both 'soft' (␣) and 'hard' spaces (`\,` and `~`), automatically identifies exponents (as standard marked using `e`, `E`, `d` or `D`) and adds the appropriate spacing of large numbers. If required, a leading zero is added before a decimal marker: both `.` and `,` are recognised as decimal markers.

123	`\num{123}` \\
1234	`\num{1234}` \\
12 345	`\num{12345}` \\
0.123	`\num{0.123}` \\
0.1234	`\num{0,1234}` \\
0.123 45	`\num{.12345}` \\
3.45×10^{-4}	`\num{3.45d-4}` \\
-10^{10}	`\num{-e10}`

Note that numbers are parsed before typesetting, which does have a performance overhead (only obvious with very large amounts of numerical input). The parser understands a range of input syntaxes, as demonstrated above.

\numlist `\numlist[⟨options⟩]{⟨numbers⟩}`

Lists of numbers may be processed using the `\numlist` function. Each ⟨*number*⟩ is given within the list of ⟨*numbers*⟩ within a brace pair, as the list can have a flexible length.

10, 30, 50 and 70	`\numlist{10;30;50;70}`

\numproduct `\numproduct[⟨options⟩]{⟨numbers⟩}`

Runs of products of numbers may be inserted using the `\numproduct` function. This acts in the same way as `\num`, but inserts either a symbol or phrase between the entries. The latter should be separated by `x` tokens.

10×30	`\numproduct{10 x 30}`

\numrange `\numrange[⟨options⟩]{⟨number1⟩}{⟨number2⟩}`

Simple ranges of numbers can be handled using the `\numrange` function. This acts in the same way as `\num`, but inserts a phrase or other text between the two entries.

10 to 30	`\numrange{10}{30}`

3.2 Angles

\ang `\ang[⟨options⟩]{⟨angle⟩}`

Angles can be typeset using the `\ang` command. The ⟨*angle*⟩ can be given either as a decimal number or as a semi-colon separated list of degrees, minutes and seconds, which is called 'arc format' in this document. The numbers which make up an angle are processed using the same system as other numbers.

10°	`\ang{10}` \\
12.3°	`\ang{12.3}` \\
4.5°	`\ang{4,5}` \\
$1^\circ 2' 3''$	`\ang{1;2;3}` \\
$1''$	`\ang{;;1}` \\
10°	`\ang{+10;;}` \\
$-1'$	`\ang{-0;1;}`

3.3 Units

\unit `\unit[⟨options⟩]{⟨unit⟩}`

The symbol for a unit can be typeset using the `\unit` macro: this provides full control over output format for the unit. Like the `\num` macro, `\unit` takes one optional and one mandatory argument. The unit formatting system can accept two types of input. When the ⟨*unit*⟩ contains literal items (for example letters or numbers) then siunitx converts `.` and `~` into inter-unit product and correctly positions sub- and superscripts specified using `_` and `^`. The formatting methods will work with both math and text mode.

$\mathrm{kg\,m/s^2}$	`\unit{kg.m/s^2}` \\
$\mathrm{g_{polymer}\,mol_{cat}\,s^{-1}}$	`\unit{g_{polymer}~mol_{cat}.s^{-1}}`

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

```
\unit{\kilo\gram\metre\per\square\second} \\
\unit{\gram\per\cubic\centi\metre}        \\
\unit{\square\volt\cubic\lumen\per\farad} \\
\unit{\metre\squared\per\gray\cubic\lux}  \\
\unit{\henry\second}
```

$\mathrm{kg\,m\,s^{-2}}$
$\mathrm{g\,cm^{-3}}$
$\mathrm{V^2\,lm^3\,F^{-1}}$
$\mathrm{m^2\,Gy^{-1}\,lx^3}$
$\mathrm{H\,s}$

On its own, this is less convenient than the direct method, although it does use meaning rather than appearance for input. 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 input, new functionality is made

available. By altering the settings used by the package, the same input can yield a variety of different output formats. For example, the \per macro can give reciprocal powers, slashes or be used to construct units as fractions.

\qty `\qty[⟨options⟩]{⟨number⟩}{⟨unit⟩}`

Very often, numbers and units are given together. Formally, the value of a quantity is the product of the number and the unit, the space being regarded as a multiplication sign. The \qty macro combines the functionality of \num and \unit, and makes this both possible and easy. The ⟨*number*⟩ and ⟨*unit*⟩ arguments work exactly like those for the \num and \unit macros, respectively.

```
\qty[mode = text]{1.23}{J.mol^{-1}.K^{-1}}          \\
\qty{.23e7}{\candela}                               \\
\qty[per-mode = symbol]{1.99}{\per\kilogram}        \\
\qty[per-mode = fraction]{1,345}{\coulomb\per\mole}
```

$1.23\,\mathrm{J\,mol^{-1}\,K^{-1}}$
$0.23 \times 10^{7}\,\mathrm{cd}$
$1.99/\mathrm{kg}$
$1.345\,\frac{\mathrm{C}}{\mathrm{mol}}$

It is possible to set up the unit macros to be available outside of the \qty and \unit functions. This is not the standard behavior as there is the risk of name clashes (for example, \day is a TEX primitive and several packages define \degree). Full details of using 'stand alone' units are found in 4.9.

\qtylist `\qtylist[⟨options⟩]{⟨numbers⟩}{⟨unit⟩}`

Lists of numbers with units can be handled using the \qtylist function. The behavior of this function is similar to \numlist, but with the addition of the unit to each number.

$10\,\mathrm{m}$, $30\,\mathrm{m}$ and $45\,\mathrm{m}$ `\qtylist{10;30;45}{\metre}`

\qtyproduct `\qtyproduct[⟨options⟩]{⟨numbers⟩}{⟨unit⟩}`

Runs of products of numbers with units can be handled using the \qtyproduct function. The behavior of this function is similar to \numproduct, but with the addition of a unit to each number.

$10\,\mathrm{m} \times 30\,\mathrm{m} \times 45\,\mathrm{m}$ `\qtyproduct{10 x 30 x 45}{\metre}`

\qtyrange `\qtyrange[⟨options⟩]{⟨number1⟩}{⟨number2⟩}{⟨unit⟩}`

Ranges of numbers with units can be handled using the \qtyrange function. The behavior of this function is similar to \numrange, but with the addition of a unit to each number.

$10\,\mathrm{m}$ to $30\,\mathrm{m}$ `\qtyrange{10}{30}{\metre}`

The input of lists, products and ranges of quantities using a single command allows them to be adjusted together. These commands are intended to allow consistent formatting of related values: as such, they apply a single unit to all of the values. This is particularly notable when using adjustment of the numerical values.

Table 1: SI base units.

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

3.4 Complex numbers and quantities

`\complexnum` `\complexnum[⟨options⟩]{⟨number⟩}`

Typesets the complex number, which can be given in the Cartesian form $a + b\mathrm{i}$ or $a + \mathrm{i}b$, or in the polar form $r:\theta$. Processing of the numerical parts is otherwise identical to the standard `\num` command.

`\complexqty` `\complexqty[⟨options⟩]{⟨number⟩}{⟨unit⟩}`

Typesets the complex number, which can be given in the Cartesian form $a + b\mathrm{i}$ or $a + \mathrm{i}b$, or in the polar form $r:\theta$. Processing of the numerical parts is otherwise identical to the standard `\qty` command.

3.5 The unit macros

The package always defines the basic set of SI units with macro names. This includes the base SI units, the derived units with special names and the prefixes. A small number of powers are also given pre-defined names. Full details of units in the SI are available on-line [1].

The seven base SI units are always defined (Table 1). In addition, the macro `\meter` is available as an alias for `\metre`, for users of US spellings. The full details of the base units are given in the SI Brochure.

The SI also lists a number of units which have special names and symbols: these are listed in Table 2.

In addition to the official SI units, siunitx also provides macros for a number of units which are accepted for use in the SI although they are not SI units. Table 3 lists the 'accepted' units. The command `\percent` is also provided for use in units: this is accepted with the SI as detailed in Section 5.4.7 of the Brochure.

In addition to the units themselves, siunitx provides pre-defined macros for all of the SI prefixes (Table 4). The spelling '`\deka`' is provided for US users as an alternative to `\deca`.

A small number of pre-defined powers are provided as macros. `\square` and `\cubic` are intended for use before units, with `\squared` and `\cubed` going after the unit.

Bq^2	`\unit{\square\becquerel} \\`
$\mathrm{J}^2\,\mathrm{lm}^{-1}$	`\unit{\joule\squared\per\lumen} \\`
$\mathrm{lx}^3\,\mathrm{V}\,\mathrm{T}^3$	`\unit{\cubic\lux\volt\tesla\cubed}`

Table 2: Coherent derived units in the SI with special names and symbols.

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

Table 3: Non-SI units accepted for use with the International System of Units.

Unit	Command	Symbol
astronomical unit	`\astronomicalunit`	au
bel	`\bel`	B
dalton	`\dalton`	Da
day	`\day`	d
decibel	`\decibel`	dB
degree	`\degree`	°
electronvolt	`\electronvolt`	eV
hectare	`\hectare`	ha
hour	`\hour`	h
litre	`\litre`	L
	`\liter`	L
minute (plane angle)	`\arcminute`	′
minute (time)	`\minute`	min
second (plane angle)	`\arcsecond`	″
neper	`\neper`	Np
tonne	`\tonne`	t

Table 4: SI prefixes.

Prefix	Command	Symbol	Power	Prefix	Command	Symbol	Power
quecto	\quecto	q	−30	deca	\deca	da	1
ronto	\ronto	r	−27	hecto	\hecto	h	2
yocto	\yocto	y	−24	kilo	\kilo	k	3
zepto	\zepto	z	−21	mega	\mega	M	6
atto	\atto	a	−18	giga	\giga	G	9
femto	\femto	f	−15	tera	\tera	T	12
pico	\pico	p	−12	peta	\peta	P	15
nano	\nano	n	−9	exa	\exa	E	18
micro	\micro	µ	−6	zetta	\zetta	Z	21
milli	\milli	m	−3	yotta	\yotta	Y	24
centi	\centi	c	−2	ronna	\ronna	R	27
deci	\deci	d	−1	quetta	\quetta	Q	30

Generic powers can be inserted on a one-off basis using the \tothe and \raiseto macros. These are the only macros for units which take an argument:

H^{5}
$\mathrm{rad}^{4.5}$

```
\unit{\henry\tothe{5}} \\
\unit{\raiseto{4.5}\radian}
```

Reciprocal powers are indicated using the \per macro. This applies to the next unit only, unless the sticky-per option is turned on.

$\mathrm{J\,mol^{-1}\,K^{-1}}$
$\mathrm{J\,mol^{-1}\,K}$
$\mathrm{H^{-5}}$
$\mathrm{Bq^{-2}}$

```
\unit{\joule\per\mole\per\kelvin} \\
\unit{\joule\per\mole\kelvin} \\
\unit{\per\henry\tothe{5}} \\
\unit{\per\square\becquerel}
```

As for generic powers, generic qualifiers are also available using the \of function:

```
\unit{\kilogram\of{metal}} \\
\qty[qualifier-mode = bracket]
  {0.1}{\milli\mole\of{cat}\per\kilogram\of{prod}}
```

$\mathrm{kg_{metal}}$
$0.1\,\mathrm{mmol(cat)\,kg(prod)^{-1}}$

When the cancel package is loaded, it is possible to 'cancel out' units using the \cancel macro. This applies to the next unit, in a similar manner to a prefix. The \highlight macro is also available to selectively color units. Both \cancel and \highlight are outside of the normal semantic meaning of units, but are provided as they may be useful in some cases.

```
\unit[per-mode = fraction]
  {\cancel\kilogram\metre\per\cancel\kilogram\per\second} \\
\unit{\highlight{red}\kilogram\metre\per\second} \\
\unit[unit-color = purple]
  {\highlight{blue}\kilogram\metre\per\second}
```

$\frac{\cancel{\mathrm{kg}}\,\mathrm{m}}{\cancel{\mathrm{kg}}\,\mathrm{s}}$
$\mathrm{kg\,m\,s^{-1}}$
$\mathrm{kg\,m\,s^{-1}}$

3.6 Unit abbreviations

In addition to the 'full' names, siunitx loads a set of abbreviated versions of the SI units (Table 5). The standard siunitx settings only create these abbreviations within the scope of the `\unit` and `\qty` functions, meaning that no clashes should occur (for example with the standard `\pm` symbol).

Table 5: Unit abbreviations

Unit	Abbreviation	Symbol
femtogram	`\fg`	fg
picogram	`\pg`	pg
nanogram	`\ng`	ng
microgram	`\ug`	µg
milligram	`\mg`	mg
gram	`\g`	g
kilogram	`\kg`	kg
picometre	`\pm`	pm
nanometre	`\nm`	nm
micrometre	`\um`	µm
millimetre	`\mm`	mm
centimetre	`\cm`	cm
decimetre	`\dm`	dm
metre	`\m`	m
kilometre	`\km`	km
attosecond	`\as`	as
femtosecond	`\fs`	fs
picosecond	`\ps`	ps
nanosecond	`\ns`	ns
microsecond	`\us`	µs
millisecond	`\ms`	ms
second	`\s`	s
femtomole	`\fmol`	fmol
picomole	`\pmol`	pmol
nanomole	`\nmol`	nmol
micromole	`\umol`	µmol
millimole	`\mmol`	mmol
mole	`\mol`	mol

Continued on next page

Continued from previous page

Unit	Abbreviation	Symbol
kilomole	`\kmol`	kmol
picoampere	`\pA`	pA
nanoampere	`\nA`	nA
microampere	`\uA`	μA
milliampere	`\mA`	mA
ampere	`\A`	A
kiloampere	`\kA`	kA
microlitre	`\ul`	μL
millilitre	`\ml`	mL
litre	`\l`	L
hectolitre	`\hl`	hL
microliter	`\uL`	μL
milliliter	`\mL`	mL
liter	`\L`	L
hectoliter	`\hL`	hL
millihertz	`\mHz`	mHz
hertz	`\Hz`	Hz
kilohertz	`\kHz`	kHz
megahertz	`\MHz`	MHz
gigahertz	`\GHz`	GHz
terahertz	`\THz`	THz
millinewton	`\mN`	mN
newton	`\N`	N
kilonewton	`\kN`	kN
meganewton	`\MN`	MN
pascal	`\Pa`	Pa
kilopascal	`\kPa`	kPa
megapacal	`\MPa`	MPa
gigapascal	`\GPa`	GPa
milliohm	`\mohm`	mΩ
kilohm	`\kohm`	kΩ
megohm	`\Mohm`	MΩ
picovolt	`\pV`	pV
nanovolt	`\nV`	nV
microvolt	`\uV`	μV
millivolt	`\mV`	mV

Continued on next page

Continued from previous page

Unit	Abbreviation	Symbol
volt	`\V`	V
kilovolt	`\kV`	kV
watt	`\W`	W
nanowatt	`\nW`	nW
microwatt	`\uW`	μW
milliwatt	`\mW`	mW
kilowatt	`\kW`	kW
megawatt	`\MW`	MW
gigawatt	`\GW`	GW
joule	`\J`	J
microjoule	`\uJ`	μJ
millijoule	`\mJ`	mJ
kilojoule	`\kJ`	kJ
electronvolt	`\eV`	eV
millielectronvolt	`\meV`	meV
kiloelectronvolt	`\keV`	keV
megaelectronvolt	`\MeV`	MeV
gigaelectronvolt	`\GeV`	GeV
teraelectronvolt	`\TeV`	TeV
kilowatt hour	`\kWh`	kW h
farad	`\F`	F
femtofarad	`\fF`	fF
picofarad	`\pF`	pF
nanofarad	`\nF`	nF
microfarad	`\uF`	μF
millifarad	`\mF`	mF
henry	`\H`	H
femtohenry	`\fH`	fH
picohenry	`\pH`	pH
nanohenry	`\nH`	nH
millihenry	`\mH`	mH
microhenry	`\uH`	μH
coulomb	`\C`	C
nanocoulomb	`\nC`	nC
millicoulomb	`\mC`	mC
microcoulomb	`\uC`	μC

Continued on next page

Table 6: Binary prefixes.

Prefix	Command	Symbol	Power
kibi	\kibi	Ki	10
mebi	\mebi	Mi	20
gibi	\gibi	Gi	30
tebi	\tebi	Ti	40
pebi	\pebi	Pi	50
exbi	\exbi	Ei	60
zebi	\zebi	Zi	70
yobi	\yobi	Yi	80

Continued from previous page

Unit	Abbreviation	Symbol
tesla	\T	T
millitesla	\mT	mT
microtesla	\uT	μT
kelvin	\K	K
decibel	\dB	dB

bit
byte

Binary data is expressed in units of bits and bytes. These are normally given prefixes which use powers of two, rather than the powers of ten used by the SI prefixes. As these binary prefixes are closely related to the SI prefixes, they are defined by siunitx.

3.7 Creating new macros

The various macro components of a unit have to be defined before they can be used. The package supplies a number of common definitions, but new definitions are also possible. As the definition of a logical unit should remain the same in a single document, these creation functions are all preamble-only.

\DeclareSIUnit `\DeclareSIUnit[⟨options⟩]{⟨unit⟩}{⟨symbol⟩}`

New units are produced using the \DeclareSIUnit macro. The ⟨*symbol*⟩ can contain literal input, other units, multiple prefixes, powers and \per, although literal text should not be intermixed with unit macros. Units can be created with ⟨*options*⟩ from the usual list understood by siunitx, and apply the specific unit macro only. The (first) optional argument to \qty and \unit can be used to override the settings for the unit: an example is the \degree unit.

3.1415° `\qty{3.1415}{\degree}`

This is declared in the package (effectively) as

```
\DeclareSIUnit[quantity-product = {}]
  \degree{\text{\textdegree}}
```

The spacing can still be altered at point of use:

```
\qty{67890}{\degree} \\
\qty[quantity-product = \,]{67890}{\degree}
```

67 890°
67 890°

The meaning of a pre-defined unit can be altered by using `\DeclareSIUnit` after loading siunitx. This will overwrite the original definition with the newer version.

`\DeclareSIPrefix` `\DeclareSIPrefix{⟨prefix⟩}{⟨symbol⟩}{⟨powers-ten⟩}`

The standard SI powers of ten are defined by the package, and are described above. However, the user can define new prefixes with `\DeclareSIPrefix`. For example, `\kilo` is defined

```
\DeclareSIPrefix\kilo{k}{3}
```

`\DeclareSIPower` `\DeclareSIPower{⟨symbol-before⟩}{⟨symbol-after⟩}{⟨power⟩}`

This function creates two symbols, one for use before a unit, the second for use after a unit, both of which are equivalent to the ⟨*power*⟩. For example, one might use

```
\DeclareSIPower\quartic\tothefourth{4}
```

with the functions then used in the document as

kg^{4} `\unit{\kilogram\tothefourth}\\`
m^{4} `\unit{\quartic\metre}`

`\DeclareSIQualifier` `\DeclareSIQualifier{⟨qualifier⟩}{⟨symbol⟩}`

Following the syntax of the other macros, qualifiers may be created using the `\DeclareSIQualifier` command. In contrast to the other parts of a unit, there are no pre-defined qualifiers. It is therefore entirely up to the user to create these. For example, to identify the mass of a product created when using a particular catalyst, the preamble could contain:

```
\DeclareSIQualifier\polymer{pol}
\DeclareSIQualifier\catalyst{cat}
```

and then in the body the document could read

```
\qty{1.234}{\gram\polymer\per\mole\catalyst\per\hour}
```

$1.234\,\mathrm{g_{pol}\,mol_{cat}^{-1}\,h^{-1}}$

Table 7: Standard behavior of the S column type.

Some Values
2.3456
34.2345
−6.7835
90.473
5642.5
1.2×10^3
10^4

3.8 Tabular material

Aligning numbers in tabular content is handled by a new column type, the S column. This new column type can align material using a number of different strategies, with the aim of flexibility of output without needing to alter the input. The method used as standard is to place the decimal marker in the number at the center of the cell and to align the material appropriately (Table 7).

```
\begin{table}
  \caption{Standard behavior of the \texttt{S} column type.%
    \label{tab:S:standard}}
  \begin{tabular}{@{}S@{}}
  \toprule
    {Some Values} \\
  \midrule
       2.3456 \\
      34.2345 \\
      -6.7835 \\
      90.473  \\
    5642.5    \\
        1.2e3 \\
          e4  \\
  \bottomrule
  \end{tabular}
\end{table}
```

The S column will attempt to automatically detect material which should be placed before or after a number, and will maintain the alignment of the numerical data (Table 8). If the material could be mistaken for part of a number, it should be protected by braces. The use of \color in a table cell will also be detected and will override any general color applied by siunitx.

```
\begin{table}
  \caption{Detection of surrounding material in an \texttt{S} column.%
    \label{tab:S:extras}}
  \begin{tabular}{@{}S@{}}
  \toprule
    {Some Values} \\
  \midrule
    12.34 \\
    \color{purple} 975,31 \\
```

Table 8: Detection of surrounding material in an S column.

Some Values
12.34
975.31
44.268[a]

```
    44.268 \textsuperscript{\emph{a}} \\
  \bottomrule
  \end{tabular}
\end{table}
```

\tablenum `\tablenum[⟨options⟩]{⟨number⟩}`

Within more complex tables, aligned numbers may be desirable within the argument of `\multicolumn` or `\multirow`.[1] The `\tablenum` function is available to achieve alignment in these situations: this is, in effect, a macro version of the S column (Table 9).

```
\begin{table}
  \caption{Controlling complex alignment with the \cs{tablenum} macro.%
    \label{tab:tablenum}}
  \begin{tabular}{@{}lr@{}}
    \toprule
    Heading & Heading \\
    \midrule
    Info & More info \\
    Info & More info \\
    \multicolumn{2}{c}{\tablenum[table-format = 4.4]{12,34}}    \\
    \multicolumn{2}{c}{\tablenum[table-format = 4.4]{333.5567}} \\
    \multicolumn{2}{c}{\tablenum[table-format = 4.4]{4563.21}}  \\
    \bottomrule
  \end{tabular}
  \hspace{\fill}%
  \begin{tabular}{@{}lr@{}}
    \toprule
    Heading & Heading \\
    \midrule
    \multirow{2}*{\tablenum{88,999}} & aaa \\
                                     & bbb \\
    \multirow{2}*{\tablenum{33,435}} & ccc \\
                                     & ddd \\
    \bottomrule
  \end{tabular}
\end{table}
```

[1] Provided by the multirow package

Table 9: Controlling complex alignment with the `\tablenum` macro.

Heading	Heading
Info	More info
Info	More info
12.34	
333.5567	
4563.21	

Heading	Heading
88.999	aaa bbb
33.435	ccc ddd

4 Package control options

4.1 The key–value control system

The package uses a range of different key types:

`Choice` Takes a limited number of choices, which are described separately for each key.

`Integer` Requires a number as the argument.

`Length` Requires a length, either as a literal value such as `2.0cm`, or stored as a LaTeX length.

`Literal` A key which uses the value(s) given directly, either to check input or in output.

`Macro` Requires a macro, which may need a single argument.

`Math` Similar to a `literal` option, but the input is always used in math mode, irrespective of other siunitx settings. Thus to text-mode only input must be placed inside the argument of a `\text` macro.

`Meta` These are options which actually apply a number of other options.

`Switch` These are on–off switches, and recognise `true` and `false`. Giving just the key name also turns the key on.

The tables of option names use these descriptions to indicate how the keys should be used.

4.2 Printing

The siunitx package can control the font used to print output independently of the surrounding material. Which aspects of the font follow those of the surroundings is influenced by a range of setting as detailed in Table 10.

`mode` `number-mode` `unit-mode`

The `mode` option determines whether siunitx uses math or text mode when printing output. The choices are `match`, `math`, `text`. The `match` setting means that printing uses the prevailing mode unchanged whereas `math` and `text` select the appropriate TeX mode. It is possible to have different fonts in math and text modes, which will highlight the difference. The font settings which apply are also different depending on the mode. As well as the overall setting, it is possible to apply mode to numbers and units separately using the `number-mode` and `unit-mode` options.

`reset-text-family` `reset-text-series` `reset-text-shape`

When printing in text mode, the options `reset-text-family`, `reset-text-series`

Table 10: Print options.

Option name	Type	Default
color	Literal	⟨*none*⟩
mode	Choice	math
number-color	Literal	⟨*none*⟩
number-mode	Choice	math
propagate-math-font	Switch	false
reset-math-version	Switch	true
reset-text-family	Switch	true
reset-text-series	Switch	true
reset-text-shape	Switch	true
text-family-to-math	Switch	false
text-font-command	Literal	⟨*none*⟩
text-subscript-command	Literal	\textsubscript
text-superscript-command	Literal	\textsuperscript
text-series-to-math	Switch	false
unit-color	Literal	⟨*none*⟩
unit-mode	Choice	math

and reset-text-shape apply. When these are active, siunitx resets the relevant font selection axis property when printing: the standard font setting is upright mid-weight roman (\upshape \mdseries \rmfamily).

```
\sisetup{mode = text}
{\itshape  \num{1234}}\\
{\bfseries \num{1234}}\\
{\sffamily \num{1234}}\\
\sisetup{
  reset-text-family = false ,
  reset-text-series = false ,
  reset-text-shape  = false
}
{\itshape  \num{1234}}\\
{\bfseries \num{1234}}\\
{\sffamily \num{1234}}\\
```

1234
1234
1234
1234
1234
1234

propagate-math-font
reset-math-version

In math mode, the font used by LaTeX is 'invariant', and this is reflected in the options available. With the standard settings, in math mode printing uses the standard math font and version (weight). The option propagate-math-font may be used to apply the prevailing math font to the printed material. The setting reset-math-version controls whether the math version is reset or not. Note that math version is typically used to set 'bold math' but may also be used for other effects, for example all sanserif math.

```
{\boldmath \unit{\kilogram}}\\
{\sansmath $\unit{\kilogram}$}\\
{$\mathsf{\unit{\kilogram}}$}\\
\sisetup{
  propagate-math-font = true  ,
  reset-math-version  = false
}
{\boldmath \unit{\kilogram}}\\
{\sansmath $\unit{\kilogram}$}\\
{$\mathsf{\unit{\kilogram}}$}
```

kg
kg
kg
kg
kg
kg

`text-family-to-math` `text-series-to-math`

The options `text-family-to-math` and `text-series-to-math` can be used to match (as far as possible) math mode output to the surrounding text. These options work by detecting the current text settings and making the appropriate choice in math mode.

```
{\sffamily \unit{\kilogram}}\\
{\bfseries $\unit{\kilogram}$}\\
\sisetup{
  text-family-to-math = true  ,
  text-series-to-math = true
}
{\sffamily \unit{\kilogram}}\\
{\bfseries $\unit{\kilogram}$}
```

kg
kg
kg
kg

`text-font-command`

In some circumstances, it may be desirable to use a non-standard font command when printing in text mode. This might be used for example to switch from old-style to lining numbers whilst still using text mode. This may be achieved by setting `text-font-command`. For example, this document uses old-style numbers in text mode as-standard, which can be over-ridden by selecting the font variant which does not feature them.

```
\sisetup{number-mode = text}
\qty{123456789}{\kilo\volt\per\centi\metre} \\
\sisetup{text-font-command = \fontfamily{pplx}\selectfont}
\qty{123456789}{\kilo\volt\per\centi\metre}
```

123 456 789 kV cm^{-1}
123 456 789 kV cm^{-1}

`text-subscript-command` `text-superscript-command`

In most cases, the commands `\textsubscript` and `\textsuperscript` are appropriate for creating sub- and superscript material in text mode. However, when the realscripts package is loaded, it is possible that the output is not as desired. These two commands are therefore available to allow tuning of the results. They can also be used for non-standard effects, for example as here adding color to subscripts.

```
\sisetup{unit-mode = text}
\unit{\kg\of{polymer}} \\
\newcommand*\mysubscript[1]{%
  \textsubscript{\textcolor{blue}{#1}}%
}
\sisetup{text-subscript-command = \mysubscript}
\unit{\kg\of{polymer}}
```

kg$_{\text{polymer}}$
kg$_{\text{polymer}}$

Table 11: Options for number parsing.

Option name	Type	Default
evaluate-expression	Switch	false
expression	Literal	#1
input-close-uncertainty	Literal	)
input-comparators	Literal	<=>\approx\ge\geq \gg\le\leq\ll\sim
input-decimal-markers	Literal	.,
input-digits	Literal	0123456789
input-exponent-markers	Literal	dDeE
input-ignore	Literal	⟨*none*⟩
input-open-uncertainty	Literal	(
input-signs	Literal	+-\pm\mp
input-uncertainty-divider	Literal	:
input-uncertainty-signs	Literal	\pm
parse-numbers	Switch	true
retain-explicit-decimal-marker	Switch	false
retain-explicit-plus	Switch	false
retain-negative-zero	Switch	false
retain-zero-uncertainty	Switch	false

color
number-color
unit-color

The color of printed output can be set using the color option. When no color is given, printing follows the surrounding text. In contrast, when a specific color is given, it is used irrespective of the surroundings. As with mode, the color setting may also be applied to numbers and units independently.

Some text
4 kg
More text
4 kg
Still red here!

```
\color{red}%
Some text \\
\qty{4}{\kilogram} \\
More text \\
\qty[color = blue]{4}{\kilogram} \\
Still red here!
```

4.3 Parsing numbers

The package uses a sophisticated parsing system to understand numbers. This allows siunitx to carry out a range of formatting, as described later. All of the input options take lists of literal tokens, and are summarised in Table 11.

input-digits
input-decimal-markers
input-signs
input-exponent-markers

The basic parts of a number are the digits, any sign and a separator between the integer and decimal parts. These are stored in the input options input-digits, input-decimal-markers and input-signs, respectively. More than one input decimal marker can be used: it will be converted by the package to the appropriate output marker. Numbers which include an exponent part also require a marker for the exponent: this again is taken from the range of tokens in the input-exponent-markers option.

input-ignore

Tokens given in the input-ignore list are totally passed over by siunitx: they will be removed from the input with no further processing.

input-comparators

In addition to signs, siunitx can recognise comparators, such as <. The package will

automatically carry out conversions for `<<`, `>>`, `<=`, `>=` and `~` to `\ll`, `\gg`, `\le`, `\ge` and `\sim`, respectively.

<10	`\num{< 10} \\`
$\gg 5\,\mathrm{m}$	`\qty{>> 5}{\metre} \\`
≤ 0.12	`\num{\le 0.12}`

`input-open-uncertainty` `input-close-uncertainty` `input-uncertainty-signs`

In some fields, it is common to give the uncertainty in a number in brackets after the main part of the number, for example '1.234(5)'. The opening and closing symbols used for this type of input are set as `input-open-uncertainty` and `input-close-uncertainty`. Alternatively, the uncertainty may be given as a separate part following a sign. Which signs are valid for this operation is determined by the `input-uncertainty-signs` option. As with other signs, the combination `+-` will automatically be converted to `\pm` internally.

$9.99(9)$	`\num{9.99(9)}       \\`
$9.99(9)$	`\num{9.99 +- 0.09}  \\`
$9.99(9)$	`\num{9.99 \pm 0.09} \\`
$123.0(45)$	`\num{123 +- 4.5}    \\`
$12.3(60)$	`\num{12.3 +- 6}`

Uncertainties which cross the decimal marker may be given with or without a decimal marker in 'compact' form. These are treated as equivalent by the code.[2] Where multiple uncertainty values cross the decimal marker, they should all have the same number of digits.

$123.4(12)$	`\num{123.4(12)} \\`
$123.4(12)$	`\num{123.4(1.2)}`

Multiple symmetrical uncertainties may also be given: these must all either be in the short or the long form.

$123.4 \pm 1.2 \pm 4.5$	`\num{123.4(12)(45)} \\`
$123.4 \pm 1.2 \pm 4.5$	`\num{123.4 \pm 1.2 \pm 4.5}`

`input-uncertainty-divider`

In some areas, uncertainties (or tolerances) are given in an asymmetric format. This is supported in the 'compact' input form in siunitx, with the positive and negative parts of the uncertainty divided by a symbol listed in `input-uncertainty-divider`. When given in this divided form, the two parts are interpreted as first the positive and then the negative component. Asymmetric uncertainties which cross the decimal marker are supported provided both components have the same number of digits.

$10.56^{+0.12}_{-0.34}$	`\num{10.56(12:34)} \\`
$123.4^{+4.6}_{-7.8}$	`\num{123.4(4.6:7.8)}`

Symmetrical and asymmetrical uncertainties may be intermixed.

$10.56^{+0.01}_{-0.02} \pm 0.03$	`\num{10.56(1:2)(3)} \\`
$6.45 \pm 0.02^{+0.03}_{-0.04}$	`\num{6.45(2)(3:4)}`

parse-numbers

The `parse-numbers` option turns the entire parsing system on and off. The option is made available for two reasons. First, if all of the numbers in a document are to be reproduced 'as given', turning off the parser will represent a significant saving in processing required. Second, it allows the use of arbitrary TEX code in numbers. If the parser is turned off, the input will be printed in math mode (requiring `\text` to protect any text in the number).

```
\num[parse-numbers = false]{\sqrt{2}}        \\
\qty[parse-numbers = false]{\sqrt{3}}{\metre}
```

$\sqrt{2}$
$\sqrt{3}\,\mathrm{m}$

evaluate-expression
expression

With the standard settings, numerical input is parsed 'as is' with no attempt to interpret it mathematically. By enabling the `evaluate-expression` option, the input can be processed by the standard LaTeX3 FPU (see package xfp for more). The nature of the expression itself can be adjusted using the `expression` setting: as standard, the entire input is simply parsed with no change, but this setting may be used to add additional steps. The *input* in such an expression is represented by `#1`. Note that the FPU uses its own syntax for numbers, most notably in that a decimal marker must be `.`.

```
\sisetup{evaluate-expression}%
\qty{2 + 4 * 3}{\joule} \\
\qty[expression = 10 * (#1)]{2 + 4 * 3}{\joule}
```

$14\,\mathrm{J}$
$140\,\mathrm{J}$

retain-explicit-decimal-marker
retain-explicit-plus
retain-negative-zero
retain-zero-uncertainty

In some areas, a trailing decimal marker with no decimal part present is used to show that zeros in the integer part are significant. This can be enabled using the `retain-explicit-decimal-marker` option. The inclusion of a leading plus sign is usually unnecessary for positive numbers, and so they are not retained as-standard when parsing. The `retain-explicit-plus` option is available to control this behavior. Similarly, an uncertainty of zero is normally not meaningful, and so is ignored by the parser. This can be controlled using the `retain-zero-uncertainty` option. Finally, a negative sign for an entirely zero value may or may not have significance: this is controlled by the `retain-negative-zero` option.

```
\num{10.} \\
\num[retain-explicit-decimal-marker]{10.} \\
\num{+345} \\
\num[retain-explicit-plus]{+345} \\
\num{-0} \\
\num[retain-negative-zero]{-0} \\
\num{12.3(0)} \\
\num[retain-zero-uncertainty]{12.3(0)}
```

[2]The package author favors the form without a decimal marker, and formal guidance is ambiguous on which is correct. The form with a decimal marker is seen in for example some NIST publications.

Table 12: Number post-processing options.

Option name	Type	Default
drop-exponent	Switch	false
drop-uncertainty	Switch	false
drop-zero-decimal	Switch	false
exponent-mode	Switch	input
exponent-thresholds	Literal	-3:3
fixed-exponent	Integer	0
minimum-integer-digits	Integer	0
minimum-decimal-digits	Integer	0
round-direction	Choice	nearest
round-half	Choice	up
round-minimum	Literal	0
round-mode	Choice	none
round-pad	Switch	true
round-precision	Integer	2
round-zero-positive	Switch	true
uncertainty-round-direction	Choice	nearest

10
10.
345
+345
0
−0
12.3
12.3(0)

4.4 Post-processing numbers

Before typesetting numbers, various post-processing steps can be carried out. These involve adding or removing information from the number in a systematic way; the options are summarised in Table 12.

exponent-mode
fixed-exponent

Numbers can be converted to scientific notation by the package. This is controlled by the `exponent-mode` option, which takes choices `input`, `fixed`, `engineering`, `scientific` and `threshold`. The `fixed` setting will use the exponent value by the `fixed-exponent` option. When `engineering` is set, the exponent is always a power of three.

0.001
0.0100
1200
1×10^{-3}
1.00×10^{-2}
1.200×10^{3}
1×10^{-3}
10.0×10^{-3}
1.200×10^{3}
$0.000\,01 \times 10^{2}$
$0.000\,100 \times 10^{2}$
12.00×10^{2}

```
\num{0.001}  \\
\num{0.0100} \\
\num{1200}   \\
\sisetup{exponent-mode = scientific}%
\num{0.001}  \\
\num{0.0100} \\
\num{1200}   \\
\sisetup{exponent-mode = engineering}%
\num{0.001}  \\
\num{0.0100} \\
\num{1200}   \\
\sisetup{
  exponent-mode  = fixed,
  fixed-exponent = 2,
}%
\num{0.001}  \\
\num{0.0100} \\
\num{1200}
```

When used with a `fixed-exponent` of zero, this may be used to remove scientific notation from the input

```
\num{1.23e4} \\
\num[exponent-mode = fixed, fixed-exponent = 0]{1.23e4}
```

1.23×10^{4}
12 300

`exponent-thresholds` When the `exponent-mode` is set to `threshold`, values outside of a threshold range for the exponent are always printed in scientific form. The threshold range itself is controlled by `exponent-thresholds`, which is given as ⟨*min*⟩`:`⟨*max*⟩ (Table 13).

```
\begin{table}
  \caption{Thresholds for exponents.%
    \label{tab:threshold}}
  \begin{tabular}
    {
      @{}
      S
      S[exponent-mode = threshold]
      S[exponent-mode = threshold, exponent-thresholds = -2:2]
      @{}
    }
  \toprule
    {Input} & {Threshold $-3:3$} &  {Threshold $-2:2$} \\
  \midrule
    0.001 & 0.001 & 0.001 \\
    0.012 & 0.012 & 0.012 \\
    0.123 & 0.123 & 0.123 \\
    1     & 1     & 1     \\
    12    & 12    & 12    \\
    123   & 123   & 123   \\
    1234  & 1234  & 1234  \\
  \bottomrule
  \end{tabular}
\end{table}
```

Table 13: Thresholds for exponents.

Input	Threshold $-3:3$	Threshold $-2:2$
0.001	1×10^{-3}	1×10^{-3}
0.012	0.012	1.2×10^{-2}
0.123	0.123	0.123
1	1	1
12	12	12
123	123	1.23×10^{2}
1234	1.234×10^{3}	1.234×10^{3}

Exponent mode applies *after* rounding, such that the number of decimal places for rounding is those which appear in the output.

`drop-exponent` `drop-uncertainty` The use of an uncertainty can be suppressed entirely using the `drop-uncertainty` option: this applies *before* rounding is attempted. Similarly, exponents can be dropped using `drop-exponent` can be used to suppress the exponent part (*after* conversion to a fixed exponent).

0.01(2)
0.01
0.01×10^{3}
0.01

```
\num{0.01(2)} \\
\num[drop-uncertainty]{0.01(2)} \\
 \num{0.01e3} \\
\num[drop-exponent]{0.01e3}
```

`round-mode` `round-precision` `round-pad` The package can round numerical input to a fixed number of significant figures or decimal places. This is controlled by the `round-mode` option, which takes the choices `none`, `figures`, `places` and `uncertainty`. When rounding is turned on, the number of digits used (either decimal places or significant figures in the mantissa) is set using the `round-precision` option. Rounding numbers with uncertainties may be carried out using the `uncertainty` setting to `round-mode`. In this case the precision is used first to round the uncertainty itself (to a number of figures), before rounding the main value to follow.

```
\num{1.23456}     \\
\num{14.23}       \\
\num{0.12345(9)} \\
\sisetup{
  round-mode      = places,
  round-precision = 3
}%
\num{1.23456}     \\
\num{14.23}       \\
\num{0.12345(9)} \\
\sisetup{
  round-mode      = figures,
  round-precision = 3
}%
\num{1.23456}     \\
\num{14.23}       \\
\num{0.12345(9)} \\
\sisetup{
  round-mode      = uncertainty,
  round-precision = 1
}%
\num{0.12345(9)}   \\
\num{0.12345(23)}  \\
\num{0.12345(234)}
```

1.234 56
14.23
0.123 45(9)
1.235
14.230
0.123 45(9)
1.23
14.2
0.123 45(9)
0.123 45(9)
0.1235(2)
0.123(2)

Rounding may 'extend' a short number to more digits (or figures): this is controlled by the switch `round-pad`, which is `true` as standard.

```
\sisetup{round-mode = figures, round-precision = 4}%
\num{12.3} \\
\num[round-pad = false]{12.3}
```

12.30
12.3

`round-direction` `round-half`

The detail of which approach to use for rounding may be adjusted using the options `round-direction` and `round-half`. The option `round-direction` determines which direction a value is rounded toward: a choice of `nearest`, `up` or `down`. The standard setting, `nearest`, gives the common outcome that values round depending on whether the preceding digit is greater or less than 5. The options `up` and `down` mean that values are always rounded in one direction: the `down` option may be thought of as 'truncation'.

```
\sisetup{round-mode = places}
\num{0.054} \\
\num{0.046} \\
\sisetup{round-direction = down}%
\num{0.054} \\
\num{0.046} \\
\sisetup{round-direction = up}%
\num{0.054} \\
\num{0.046}
```

0.05
0.05
0.05
0.04
0.06
0.05

For rounding to `nearest`, where the rounded part of a number is exactly half, there are two common methods for 'breaking the tie'. The choice of method is determined by the option `round-half`, which recognises the choices `up` and `even`.

```
\sisetup{
  round-mode      = figures,
  round-precision = 1,
  round-half      = up
}%
\num{0.055} \\
\num{0.045} \\
\sisetup{round-half = even}%
\num{0.055} \\
\num{0.045}
```

0.06
0.05
0.06
0.04

`uncertainty-round-direction` When rounding with uncertainties, the direction to round the uncertainty part may be set independently of the main number: this is typically used to round the uncertainty strictly upward.

```
\sisetup{round-mode = uncertainty}
\num{0.123(41)} \\
\sisetup{uncertainty-round-direction = up}%
\num{0.123(41)}
```

0.123(41)
0.123(41)

`round-minimum` There are cases in which rounding will result in the number reaching zero. It may be desirable to show such results as below a threshold value. This can be achieved by setting `round-minimum` to the threshold value. There will be no effect when rounding to a number of significant figures as it is not possible to obtain the value zero in these cases.

```
\sisetup{round-mode = places}%
\num{0.0055} \\
\num{0.0045} \\
\sisetup{round-minimum = 0.01}%
\num{0.0055} \\
\num{0.0045}
```

0.01
0.00
0.01
<0.01

`round-zero-positive` When rounding negative numbers to a fixed number of places, a zero value may result. Usually this is expressed as an unsigned value, but in some cases retaining the negative sign may be desirable. This behavior can be controlled using the `round-zero-positive` switch.

```
\sisetup{round-mode = places}%
\num{-0.001} \\
\sisetup{round-zero-positive = false}%
\num{-0.001}
```

0.00
−0.00

`drop-zero-decimal` It may be desirable to convert decimals to integers when the decimal part is zero. This is set up using the `drop-zero-decimal` option, which applies after rounding but before setting minimum numbers of digits.

```
\num{2.0} \\
\num{2.1} \\
\sisetup{drop-zero-decimal}%
\num{2.0} \\
\num{2.1}
```

2.0
2.1
2
2.1

minimum-decimal-digits
minimum-integer-digits

The minimum-decimal-digits and minimum-integer-digits option may be used to pad numbers to a given size. This applies independent of any rounding.

```
\num{123} \\
\num[minimum-integer-digits = 2]{123} \\
\num[minimum-integer-digits = 4]{123} \\
\num{0.123} \\
\num[minimum-decimal-digits = 2]{0.123} \\
\num[minimum-decimal-digits = 4]{0.123} \\
```

123
123
0123
0.123
0.123
0.1230

4.5 Printing numbers

Actually printing numbers is controlled by a number of settings, which apply ideas such as differing decimal markers, digit grouping and so on. All of these options are concerned with the appearance of output, rather than the data it conveys. The options are summarised in Table 14.

group-digits
group-separator

Grouping digits into blocks of three is a common method to increase the ease of reading of numbers. The group-digits choice controls whether this behavior applies, and takes the values all, none, decimal and integer. Grouping can be activated separately for the integer and decimal parts of a number using the appropriately-named values.

```
\num{12345.67890} \\
\num[group-digits = none]{12345.67890}   \\
\num[group-digits = decimal]{12345.67890} \\
\num[group-digits = integer]{12345.67890}
```

12 345.678 90
12345.67890
12345.678 90
12 345.67890

The separator used between groups of digits is stored by the group-separator option. This takes literal input and may be used in math mode: for a text-mode full space use \␣.

12 345
12,345
12 345

```
\num{12345} \\
\num[group-separator = {,}]{12345} \\
\num[group-separator = \ ]{12345}
```

group-minimum-digits

Grouping is not always applied to smaller numbers; this can be controlled using the option group-minimum-digits, which specifies how many digits must be present before grouping is applied. The number of digits is considered separately for the integer and decimal parts of the number: grouping does not 'cross the boundary'.

Table 14: Output options for numbers.

Option name	Type	Default
bracket-negative-numbers	Switch	false
digit-group-size	Integer	3
digit-group-first-size	Integer	3
digit-group-other-size	Integer	3
exponent-base	Literal	10
exponent-product	Math	\times
group-digits	Choice	all
group-minimum-digits	Integer	5
group-separator	Literal	\,
negative-color	Literal	⟨*none*⟩
output-close-uncertainty	Literal	)
output-decimal-marker	Literal	.
output-exponent-marker	Literal	⟨*none*⟩
output-open-uncertainty	Literal	(
print-exponent-implicit-plus	Switch	false
print-implicit-plus	Switch	false
print-mantissa-implicit-plus	Switch	false
print-unity-mantissa	Switch	true
print-zero-exponent	Switch	false
print-zero-integer	Switch	true
simplify-uncertainty	Switch	false
tight-spacing	Switch	false
uncertainty-descriptor-mode	Choice	bracket-separator
uncertainty-descriptor-separator	Literal	\
uncertainty-descriptors	Literal	⟨*none*⟩
uncertainty-mode	Choice	compact
uncertainty-separator	Literal	⟨*none*⟩
zero-decimal-as-symbol	Switch	false
zero-symbol	Literal	\mbox{---}

```
\num{1234} \\
\num{12345} \\
\num[group-minimum-digits = 5]{1234} \\
\num[group-minimum-digits = 5]{12345} \\
\num{1234.5678} \\
\num{12345.67890} \\
\num[group-minimum-digits = 5]{1234.5678} \\
\num[group-minimum-digits = 5]{12345.67890}
```

1234
12 345
1234
12 345
1234.5678
12 345.678 90
1234.5678
12 345.678 90

`digit-group-size` `digit-group-first-size` `digit-group-other-size`

The number of digits in each group can be controlled by the setting `digit-group-size`, which has standard value 3. Finer control can be achieved using `digit-group-first-size` and `digit-group-other-size`: the first group is that immediately by the decimal point, the other value applies to the second and subsequent groupings. These can be used for example to achieve the grouping typically used in India (lakh).

```
\num{1234567890} \\
\num[digit-group-size = 5]{1234567890} \\
\num[digit-group-other-size = 2]{1234567890}
```

1 234 567 890
12345 67890
1 23 45 67 890

`output-decimal-marker`

The decimal marker used in output is set using the `output-decimal-marker` option; this can differ from the input marker.

```
\num{1.23} \\
\num[output-decimal-marker = {,}]{1.23} \\
```

1.23
1,23

`exponent-base` `exponent-product`

When exponents are present in the input, the `exponent-base` and `exponent-product` options set the obvious parts of the output.

1×10^{2}
$1 \cdot 10^{2}$
1×2^{2}

```
\num[exponent-product = \times]{1e2} \\
\num[exponent-product = \cdot]{1e2} \\
\num[exponent-base = 2]{1e2}
```

`output-exponent-marker`

Alternatively, if the `output-exponent-marker` option is set then the value stored will be used in place of the normal product and base combination.

```
\num[output-exponent-marker = e]{1e2} \\
\num[output-exponent-marker = \mathrm{E}]{1e2}
```

1*e*2
1E2

uncertainty-mode
output-open-uncertainty
output-close-uncertainty
uncertainty-separator
allow-uncertainty-breaks

When input is given including a single uncertainty, it can be printed either with the uncertainty in brackets or as a separate number. This behavior is controlled by the uncertainty-mode choice. When this is set to separate, the uncertainty is printed as an entirely separate number preceded by \pm. Other settings all place the uncertainty in brackets directly attached to the main value. The standard setting of compact prints digits of uncertainty in the least-significant digits. It does *not* print a decimal marker if the uncertainty crosses the decimal. The setting full prints the full value of the uncertainty. Finally, compact-marker is available to print in the compact style except where the uncertainty crosses the decimal, in which case the full style is used. When the uncertainty is given in brackets, a space may be added between the main number and the uncertainty: this is stored using the uncertainty-separator option. The opening and closing brackets used are stored in output-open-uncertainty and output-close-uncertainty, respectively. Tokens may be inserted before the opening bracket using uncertainty-separator. As standard, a break is allowed before a separated uncertainty: this can be suppressed by setting allow-uncertainty-breaks to false.

```
\num{123.45(120)} \\
\num{0.035(14)}   \\
\sisetup{uncertainty-mode = full}
\num{123.45(120)} \\
\num{0.035(14)}   \\
\sisetup{uncertainty-mode = compact-marker}
\num{123.45(120)} \\
\num{0.035(14)}   \\
\sisetup{uncertainty-mode = separate}
\sisetup{
  output-open-uncertainty  = [,
  output-close-uncertainty = ],
  uncertainty-separator    = \,
}%
\num{1.234(5)}
```

123.45(120)
0.035(14)
123.45(1.20)
0.035(0.014)
123.45(1.20)
0.035(14)
1.234 ± 0.005

uncertainty-descriptors
uncertainty-descriptor-mode
uncertainty-descriptor-separator

Multiple uncertainties can be given for a number. These are always printed in a separated form. When there is more than one uncertainty part, it may be useful to describe the nature of this value. This can be achieved using the uncertainty-descriptors option, which take a comma-separated list of descriptions. The formatting of the descriptors can be adjusted using the settings uncertainty-descriptor-mode and uncertainty-descriptor-separator. The choices for the mode are bracket, bracket-separator, separator and subscript.

```
\num{1.2(3)(4)} \\
\sisetup{uncertainty-descriptors = {sys, stat}}
\num{1.2(3)(4)} \\
\num[uncertainty-descriptor-mode = subscript]{1.2(3)(4)}
```

$1.2 \pm 0.3 \pm 0.4$
$1.2 \pm 0.3\ (\mathrm{sys}) \pm 0.4\ (\mathrm{stat})$
$1.2 \pm 0.3_{\mathrm{sys}} \pm 0.4_{\mathrm{stat}}$

`simplify-uncertainty` Where the upper and lower parts of an asymmetrical uncertainty are identical, it may be desirable to print as in symmetrical format. This can be enabled using the `simplify-uncertainty` option.

$10.56^{+0.02}_{-0.02}$
10.56 ± 0.02

```
\num{10.56(2:2)} \\
\num[simplify-uncertainty]{10.56(2:2)}
```

`bracket-ambiguous-numbers` There are certain combinations of numerical input which can be ambiguous. This can be corrected by adding brackets in the appropriate place, and is controlled by the `bracket-ambiguous-numbers` switch. This option only applies to pure numbers: when formatting quantities, the need for brackets also depends on the placement of units, so is controlled by `separate-uncertainty-units`.

```
\sisetup{uncertainty-mode = separate}
\num{1.2(3)e4} \\
\num[bracket-ambiguous-numbers = false]{1.2(3)e4}
```

$(1.2 \pm 0.3) \times 10^{4}$
$1.2 \pm 0.3 \times 10^{4}$

`negative-color` The package can detect negative mantissa values and alter print color accordingly. This is disabled by setting the option to an empty value.

$-15\,673$
$-15\,673$

```
\num{-15673} \\
\num[negative-color = red]{-15673}
```

`bracket-negative-numbers` A common means to display negative numbers in financial situations is to place them in brackets. This can be carried out automatically using the `bracket-negative-numbers` option.

```
\num{-15673} \\
\num[bracket-negative-numbers]{-15673} \\
\qty{-10}{\metre} \\
\qty[bracket-negative-numbers]{-10}{\metre}
```

$-15\,673$
$(15\,673)$
$-10\ \mathrm{m}$
$(10)\ \mathrm{m}$

`tight-spacing` Under some circumstances is may be desirable to 'squeeze' the output spacing. This is turned on using the `tight-spacing` switch, which compresses spacing where possible.

2×10^{3}
$2{\times}10^{3}$

```
\num{2e3} \\
\num[tight-spacing = true]{2e3}
```

`print-implicit-plus` `print-mantissa-implicit-plus` `print-exponent-implicit-plus` It may be useful to force all numbers to have a sign. This behavior is controlled by the `print-implicit-plus` option: this is used if given and if no sign was present in the input. It is possible to set this behavior for the exponent and mantissa independently.

345
+345

```
\num{345} \\
\num[print-implicit-plus]{345}
```

print-unity-mantissa
print-zero-exponent
print-zero-integer

Printing of a mantissa of 1, an exponent of 0 and an integer component of 0 is controllable by the options `print-unity-mantissa`, `print-zero-exponent` and `print-zero-integer`. The standard settings print a mantissa of 1 and an integer part of 0, but omit an exponent of 0. Notice that where both `print-unity-mantissa` and `print-zero-exponent` are set to `false`, the value 1 will still be printed (*i.e.* `print-zero-exponent` has a higher priority).

```
\num{1e4} \\
\num[print-unity-mantissa = false]{1e4} \\
\num{444e0} \\
\num[print-zero-exponent = true]{444e0} \\
\num{0.123} \\
\num[print-zero-integer = false]{0.123}
```

1×10^{4}
10^{4}
444
444×10^{0}
0.123
.123

zero-decimal-as-symbol
zero-symbol

In some areas, particularly financial, entirely zero decimal parts are replaced by a dash. This is supported by option `zero-decimal-as-symbol`, which then uses the material stored using `zero-symbol` as the replacement.

```
\num{123.00} \\
\sisetup{zero-decimal-as-symbol}
\num{123.00} \\
\num[zero-symbol = \text{[{---}]}]{123.00}
```

123.00
123.—
123.[—]

4.6 Lists, products and ranges

Lists, products and ranges of numbers and quantities have a small number of specialised options, which apply to these more unusual input forms (Table 15).

list-final-separator
list-pair-separator
list-separator

Lists of numbers are printed with a separator between each item, which is stored using the `list-separator` option. The separator before the last item of a list may be different, and is therefore set using the `list-final-separator` option. The separator used for exactly two items is set using the `list-pair-separator` option. Any spaces needed should be included in the option settings: none are added within the code. The separators are always printed in text mode.

```
\numlist{0.1;0.2;0.3}                              \\
\numlist[list-separator = {; }]{0.1;0.2;0.3}       \\
\numlist[list-final-separator = {, }]{0.1;0.2;0.3} \\
\numlist[
  list-separator       = { and },
  list-final-separator = { and finally }
]{0.1;0.2;0.3} \\
\numlist{0.1;0.2} \\
\numlist[list-pair-separator = {, and }]{0.1;0.2}
```

Table 15: Output options for lists, products and ranges of numbers and quantities.

Option name	Type	Default*
`list-close-bracket`	Literal	`)`
`list-exponents`	Choice	`individual`
`list-final-separator`	Literal	`␣\text{and}␣`
`list-independent-prefix`	Switch	`false`
`list-open-bracket`	Literal	`(`
`list-pair-separator`	Literal	`␣\text{and}␣`
`list-separator`	Literal	`\text{,}␣`
`list-units`	Choice	`repeat`
`product-close-bracket`	Literal	`)`
`product-exponents`	Choice	`individual`
`product-independent-prefix`	Switch	`false`
`product-mode`	Choice	`symbol`
`product-open-bracket`	Literal	`(`
`product-phrase`	Literal	`␣\text{by}␣`
`product-symbol`	Literal	`\times`
`product-units`	Choice	`repeat`
`range-close-bracket`	Literal	`)`
`range-exponents`	Choice	`individual`
`range-independent-prefix`	Switch	`false`
`range-open-bracket`	Literal	`(`
`range-open-phrase`	Literal	⟨*empty*⟩
`range-phrase`	Literal	`␣\text{to}␣`
`range-units`	Choice	`repeat`

* The default values are actually more complex for two reasons: allowing spaces to work in both math and text modes, and localization of strings.

0.1, 0.2 and 0.3
0.1; 0.2 and 0.3
0.1, 0.2, 0.3
0.1 and 0.2 and finally 0.3
0.1 and 0.2
0.1, and 0.2

`product-mode` `product-phrase` `product-symbol`

Products of numbers can be output using either a product symbol or phrase: this is controlled by the `product-mode` setting. When `symbol` is set, the appropriate symbol is stored in `product-symbol`. When using `phrase-mode`, the information is stored in `product-phrase`. Phrases are always printed in text mode; symbols are printed using the same routine as for numbers.

```
\numproduct{5 x 100 x 2} \\
\numproduct[product-symbol = \ensuremath{\cdot}]{5 x 100 x 2} \\
\sisetup{product-mode = phrase}%
\numproduct{5 x 100 x 2}\\
\numproduct[product-phrase = { BY }]{5 x 100 x 2} \\
```

5 × 100 × 2
5 · 100 · 2
5 by 100 by 2
5 BY 100 BY 2

`range-open-phrase` `range-phrase`

Ranges of numbers can be given as input. These will have an appropriate word or symbol inserted between the two entries: this is stored using the `range-phrase` option. The phrase should include any necessary spaces: no extra space is added. The phrase is always printed in text mode.

5 to 100
5–100

```
\numrange{5}{100} \\
\numrange[range-phrase = --]{5}{100}
```

In some languages, an opening phrase is *required*; this is supported using the `range-open-phrase` key.

```
\numrange{10}{12} \\
\numrange[range-open-phrase = {\text{from} }]{5}{100}
```

10 to 12
from 5 to 100

`list-exponents` `product-exponents` `range-exponents`

Lists, products and ranges can be 'compressed' by combining the exponent parts. This is controlled by the options `list-exponents`, `product-exponents` and `range-exponents`, all of which take the choices `individual`, `combine-bracket` and `combine`. The standard setting, `individual`, leaves the exponent with the matching value. Both `combine` and `combine-bracket` take the exponent of the first entry and apply to all other entries, with the exponent itself places at the end.

```
\numlist{5e3;7e3;9e3;1e4} \\
\numproduct{5e3 x 7e3 x 9e3 x 1e4} \\
\numrange{5e3}{7e3} \\
\sisetup
  {
    list-exponents    = combine-bracket ,
    product-exponents = combine-bracket ,
    range-exponents   = combine-bracket
  }
\numlist{5e3;7e3;9e3;1e4} \\
\numproduct{5e3 x 7e3 x 9e3 x 1e4} \\
\numrange{5e3}{7e3} \\
\sisetup
  {
    list-exponents    = combine ,
    product-exponents = combine ,
    range-exponents   = combine
  }
\numlist{5e3;7e3;9e3;1e4} \\
\numproduct{5e3 x 7e3 x 9e3 x 1e4} \\
\numrange{5e3}{7e3}
```

5×10^3, 7×10^3, 9×10^3 and 1×10^4
$5 \times 10^3 \times 7 \times 10^3 \times 9 \times 10^3 \times 1 \times 10^4$
5×10^3 to 7×10^3
(5, 7, 9 and 10) $\times 10^3$
$(5 \times 7 \times 9 \times 10) \times 10^3$
(5 to 7) $\times 10^3$
5, 7, 9 and 10×10^3
$5 \times 7 \times 9 \times 10 \times 10^3$
5 to 7×10^3

`list-units` `product-units` `range-units`

The `list-units`, `product-units` and `range-units` options determine how `\qtylist`, `\qtyproduct` and `\qtyrange` command print units, respectively. The standard setting for these is `repeat`, where each number will be printed with a unit. Alternatives are `bracket` and `single`. If set to `single`, this will override collection of exponents.

```
\qtylist{2;4;6;8}{\tesla} \\
\qtylist[list-units = bracket]{2;4;6;8}{\tesla} \\
\qtylist[list-units = repeat]{2;4;6;8}{\tesla} \\
\qtylist[list-units = single]{2;4;6;8}{\tesla} \\
\qtyrange{2}{4}{\degreeCelsius} \\
\qtyrange[range-units = bracket]{2}{4}{\degreeCelsius} \\
\qtyrange[range-units = repeat]{2}{4}{\degreeCelsius} \\
\qtyrange[range-units = single]{2}{4}{\degreeCelsius}
```

2 T, 4 T, 6 T and 8 T
(2, 4, 6 and 8) T
2 T, 4 T, 6 T and 8 T
2, 4, 6 and 8 T
2 °C to 4 °C
(2 to 4) °C
2 °C to 4 °C
2 to 4 °C

The option `product-units` also offers the settings `bracket-power` and `power`.

```
\qtyproduct{2 x 4}{\metre} \\
\qtyproduct[product-units = bracket-power]{2 x 4}{\metre} \\
\qtyproduct[product-units = power]{2 x 4}{\metre}
```
$2\,\mathrm{m} \times 4\,\mathrm{m}$
$(2 \times 4)\,\mathrm{m}^2$
$2 \times 4\,\mathrm{m}^2$

`list-close-bracket` `list-open-bracket` `product-close-bracket` `product-open-bracket` `range-close-bracket` `range-open-bracket`

The brackets used can be customised using the relevant `…-open-bracket` and `…-close-bracket` options.

```
\sisetup{
  list-units         = bracket ,
  list-open-bracket  = [ ,
  list-close-bracket = ]
}
\qtylist{5e3;7e3;9e3;1e4}{\second} \\
\sisetup{
  product-units         = bracket ,
  product-open-bracket  = \{ ,
  product-close-bracket = \}
}
\qtyproduct{5e3 x 7e3 x 9e3 x 1e4}{\metre} \\
\sisetup{
  range-units         = bracket ,
  range-open-bracket  = \langle ,
  range-close-bracket = \rangle
}
\qtyrange{2}{4}{\degreeCelsius}
```
$[5 \times 10^3, 7 \times 10^3, 9 \times 10^3 \text{ and } 1 \times 10^4]\,\mathrm{s}$
$\{5 \times 10^3 \times 7 \times 10^3 \times 9 \times 10^3 \times 1 \times 10^4\}\,\mathrm{m}$
$\langle 2 \text{ to } 4 \rangle\,^\circ\mathrm{C}$

`list-independent-prefix` `product-independent-prefix` `range-independent-prefix`

When converting exponents to prefixes, there are two possible approaches to the case where units are repeated. They can all be converted to use the same (initial) prefix, or they can be processed separately. The `…-independent-prefix` options are used to control this outcome.

```
\sisetup{exponent-to-prefix = true}%
\qtyrange{1e3}{1e9}{\watt}\\
\sisetup{range-independent-prefix = true}%
\qtyrange{1e3}{1e9}{\watt}
```
1 kW to 1 000 000 kW
1 kW to 1 GW

4.7 Complex numbers

A small number of options apply specifically to the handling of complex numbers; these are summarised in Table 16.

`complex-mode`

The format in which complex values are printed can be set using the `complex-mode` option. With the standard setting (`input`), the complex value is printed as-given. By setting the option to `cartesian` or `polar`, the output format can be set to an Cartesian or polar form. Conversion uses the LaTeX3 floating-point unit, so is limited to 16 decimal places. When converting from Cartesian to polar form, the complex root symbol must come at the *end* of the imaginary part. It must also be specified using `i`.

Table 16: Options for complex numbers.

Option name	Type	Default
complex-angle-unit	Choice	degrees
complex-mode	Choice	input
complex-phase-command	Literal	\angle
complex-root-position	Choice	after-number
complex-symbol-degree	Literal	\degree
input-complex-root	Literal	ij
output-complex-root	Literal	\mathrm{i}
print-complex-unity	Switch	false

```
\complexnum{1 + i} \\
\complexnum{1:45}  \\
\complexnum[complex-mode = cartesian]{1 + i} \\
\complexnum[complex-mode = cartesian, round-mode = places]{1:45}  \\
\complexnum[complex-mode = polar]{1 + i} \\
\complexnum[complex-mode = polar]{1:45}
```

$1 + \mathrm{i}$
$1\angle 45^\circ$
$1 + \mathrm{i}$
$0.71 + 0.71\mathrm{i}$
$1.414\,213\,562\,373\,095\angle 45^\circ$
$1\angle 45^\circ$

`input-complex-root` When using complex numbers in input, the complex root ($\mathrm{i} = \sqrt{-1}$) is indicated by one of the tokens stored in `input-complex-roots`. The parser understands complex root symbols given either before or after the associated number (but will detect any invalid arrangement):

```
\complexnum{9.99 + 88.8i} \\
\complexnum{9.99 + i88.8}
```

$9.99 + 88.8\mathrm{i}$
$9.99 + 88.8\mathrm{i}$

`output-complex-root` The output complex root symbol is independent of the input and can be changed using the `output-complex-root` setting.

```
\complexnum[output-complex-root = i]{1+2i} \\
\complexnum[output-complex-root = j]{1+2i}
```

$1 + 2i$
$1 + 2j$

`complex-root-position` The position of the complex root can be adjusted to place it either before or after the associated numeral in a complex number using the `complex-root-position` option.

```
\complexnum{67-0.9i} \\
\complexnum[complex-root-position = before-number]{67-0.9i} \\
\complexnum[complex-root-position = after-number]{67-0.9i}
```

$67 - 0.9\mathrm{i}$
$67 - \mathrm{i}0.9$
$67 - 0.9\mathrm{i}$

Table 17: Angle options.

Option name	Type	Default
angle-mode	Choice	input
angle-symbol-degree	Literal	\degree
angle-symbol-minute	Literal	\arcminute
angle-symbol-over-decimal	Switch	false
angle-symbol-second	Literal	\arcsecond
angle-separator	Literal	⟨*empty*⟩
fill-angle-degrees	Switch	false
fill-angle-minutes	Switch	false
fill-angle-seconds	Switch	false

complex-angle-unit
complex-symbol-degree
complex-phase-command

When printing or converting to polar form, the angle may be interpreted in units set by complex-angle-unit: one of degrees or radians. The unit symbol used for degrees is controlled by complex-symbol-degree. To allow control of the surrounding of the entire phase part, the option complex-phase-command is available. This can be used for example to print using Steinmetz notation (which requires the steinmetz package).

```
\complexqty{1:1}{\ohm} \\
\complexqty[complex-angle-unit = radians]{1:1}{\ohm} \\
\complexqty[complex-symbol-degree = d]{1:1}{\ohm} \\
\complexqty[complex-phase-command = \phase]{1:1}{\ohm}
```

$1\angle 1^\circ\,\Omega$
$1\angle 1\,\Omega$
$1\angle 1\,\mathrm{d}\,\Omega$
$1\,\underline{/1^\circ}\,\Omega$

print-complex-unity

When the complex part of a number is exactly 1, it is possible to either print or suppress the value. This is controlled by the switch print-complex-unity.

```
\complexqty{1i}{\ohm} \\
\complexqty[print-complex-unity]{1i}{\ohm}
```

$\mathrm{i}\,\Omega$
$1\mathrm{i}\,\Omega$

4.8 Angles

Angle processing provided by the \ang function has a set of options which apply in addition to the general ones set up for number processing.

angle-mode

The format in which angles are printed can be set using the angle-mode option. With the standard setting (input), the angle is printed as-given. By setting the option to arc or decimal, the output format can be set to an arc (degrees/minutes/seconds) or decimal value. Conversion uses the LaTeX3 floating-point unit, so is limited to 16 decimal places.

2.67°
2°3′4″
2°40′12″
2°3′4″
2.67°
2.051 111 111 111 111°

```
\ang{2.67} \\
\ang{2;3;4} \\
\ang[angle-mode = arc]{2.67} \\
\ang[angle-mode = arc]{2;3;4} \\
\ang[angle-mode = decimal]{2.67} \\
\ang[angle-mode = decimal]{2;3;4} \\
```

When integer angles are converted to arc format, the presence of minute and second components is controlled by the options `fill-angle-minutes` and `fill-angle-seconds` (see below)

`angle-separator`

When angles are printed in arc format, the separation of the different parts is set up using the `arc-separator` option.

6°7′6.5″
6° 7′ 6.5″

```
\ang{6;7;6.5} \\
\ang[angle-separator = \,]{6;7;6.5}
```

`fill-angle-degrees` `fill-angle-minutes` `fill-angle-seconds`

Zero-filling for the degree, minute or second parts of an arc is controlled using the `fill-angle-degrees`, `fill-angle-minutes` and `fill-angle-seconds` options. All are off as standard.

−1°
−2′
−3″
−1°
−0°2′
−0°3″
−1°0′
−2′
−0′3″
−1°0″
−2′0″
−3″

```
\ang{-1;;} \\
\ang{;-2;} \\
\ang{;;-3} \\
{
  \sisetup{fill-angle-degrees}
  \ang{-1;;} \\
  \ang{;-2;} \\
  \ang{;;-3} \\
}
{
  \sisetup{fill-angle-minutes}
  \ang{-1;;} \\
  \ang{;-2;} \\
  \ang{;;-3} \\
}
{
  \sisetup{fill-angle-seconds}
  \ang{-1;;} \\
  \ang{;-2;} \\
  \ang{;;-3}
}
```

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

With the standard settings, the symbols used for arc angles are the unit commands `\degree`, `\arcminute` and `\arcsecond`. These can be altered using `angle-symbol-degree`, `angle-symbol-minute` and `angle-symbol-second`. This is most likely to be used when the definition of the unit macros is altered, for example to set `\arcsecond` as `as`.

6°7′6.5″
6 d7 m6.5 s

```
\ang{6;7;6.5} \\
\sisetup{
  angle-symbol-degree = d ,
  angle-symbol-minute = m ,
  angle-symbol-second = s
}
\ang{6;7;6.5}
```

Table 18: Unit creation options.

Option name	Type	Default
free-standing-units	Switch	false
overwrite-command	Switch	false
space-before-unit	Switch	false
unit-optional-argument	Switch	false
use-xspace	Switch	false

angle-symbol-over-decimal

In some subject areas, most notably astronomy, the angle symbols are given over the decimal marker, rather than at the end of the number. This behavior is available using the angle-symbol-over-decimal option.

```
\ang{45.697}  \\
\ang{6;7;6.5} \\
\ang[angle-symbol-over-decimal]{45.697} \\
\ang[angle-symbol-over-decimal]{6;7;6.5}
```

45.697°
6°7′6.5″
45°.697
6°7′6.″5

4.9 Creating units

The various macro units are created at the start of the document. siunitx can define these such that they are only available for use within the \unit and \qty functions, or can make the unit macros available throughout the document body. There are a number of settings which control this creation process (Table 18). As a result, these options all apply in the preamble only.

free-standing-units
overwrite-functions

The free-standing-units option controls whether the unit macros are usable outside of the \unit and \qty arguments. When this option is true, siunitx creates the macros for general use. The standard method to achieve this does not overwrite any existing macros: this behavior can be altered using the overwrite-commands switch. For technical reasons, when free-standing-units is set false, the names of unit macros still have to be defined: as such, they will be created if they do not exist, but will raise an error if used outside of the \unit and \qty arguments.

When using the approach of 'free-standing' unit commands, only macros created using \DeclareSIUnit are defined generally. Thus prefixes and powers should be combined with the desired unit into a single free-standing command, for example

```
\DeclareSIUnit\kilometre{\kilo\metre}
```

space-before-unit
unit-optional-argument
use-xspace

When 'free standing' unit macros are created, their behavior can be adjusted by a number of options. These are mainly intended for emulating the input syntax of older packages. The option unit-optional-argument gives the same behavior for the inputs

```
\qty{10}{\metre}
```

and

```
\metre[10].
```

Table 19: Unit output options.

Option name	Type	Default
bracket-unit-denominator	Switch	true
forbid-literal-units	Switch	false
fraction-command	Literal	\frac
inter-unit-product	Literal	\,
parse-units	Switch	true
per-mode	Choice	power
per-symbol-script-correction	Literal	\!
per-symbol	Literal	/
power-half-as-sqrt	Switch	false
qualifier-mode	Choice	subscript
qualifier-phrase	Literal	⟨*empty*⟩
sticky-per	Switch	false
unit-font-command	Literal	\mathrm

The space-before-unit and use-xspace options control the behavior at the 'ends' of the unit macros. Activating space-before-unit inserts the number–unit space before the unit is printed. This is suitable for the input syntax

```
30\metre
```

but does mean that the unit macros are incorrectly spaced in running text. On the other hand, the use-xspace option attempts to correctly space input such as

```
\metre is the symbol for metres.
```

4.10 Using units

Part of the power of siunitx is the ability to alter the output format for units without changing the input. The behavior of units is therefore controlled by a number of options which alter either the processing of units or the output directly (Table 19).

inter-unit-product

The separator between each unit is stored using the inter-unit-product option. The standard setting is a thin space: another common choice is a centered dot. To get the correct spacing it is necessary to use \ensuremath{{}\cdot{}} in the latter case.

```
\unit{\farad\squared\lumen\candela} \\
\unit[inter-unit-product = \ensuremath{{}\cdot{}}]
  {\farad\squared\lumen\candela}
```

$\mathrm{F}^2\,\mathrm{lm}\,\mathrm{cd}$
$\mathrm{F}^2 \cdot \mathrm{lm} \cdot \mathrm{cd}$

per-mode
display-per-mode
inline-per-mode
per-symbol
fraction-command
bracket-unit-denominator

The handling of \per is altered using the per-mode choice option. The standard setting is power, meaning that \per generates reciprocal powers for units. Setting the option to fraction uses the \frac function to typeset the positive and negative powers of a unit separately. The exact function can be adjusted using the fraction-command option.

```
\unit{\joule\per\mole\per\kelvin} \\
\unit{\metre\per\second\squared} \\
\unit[per-mode = fraction]{\joule\per\mole\per\kelvin} \\
\unit[per-mode = fraction]{\metre\per\second\squared}
```

$\mathrm{J\,mol^{-1}\,K^{-1}}$
$\mathrm{m\,s^{-2}}$
$\frac{\mathrm{J}}{\mathrm{mol\,K}}$
$\frac{\mathrm{m}}{\mathrm{s^2}}$

The closely-related `power-positive-first` setting acts in the same way but places all of the positive powers before any negative ones.

$\mathrm{A\,mol^{-1}\,s}$
$\mathrm{A\,s\,mol^{-1}}$

```
\unit{\ampere\per\mole\second} \\
\unit[per-mode = power-positive-first]
  {\ampere\per\mole\second}
```

It is possible to use a symbol (usually `/`) to separate the two parts of a unit by setting `per-mode` to `symbol`; the symbol used is stored using the setting `per-symbol`. This method for displaying units can be ambiguous, and so brackets are added unless `bracket-unit-denominator` is set to `false`. Notice that `bracket-unit-denominator` only applies when `per-mode` is set to `symbol`.

```
\sisetup{per-mode = symbol}%
\unit{\joule\per\mole\per\kelvin} \\
\unit{\metre\per\second\squared} \\
\unit[per-symbol = \ \text{div}\ ]{\joule\per\mole\per\kelvin} \\
\unit[bracket-unit-denominator = false]{\joule\per\mole\per\kelvin}
```

J/(mol K)
$\mathrm{m/s^2}$
J div (mol K)
J/mol K

The often-requested (but mathematically invalid) `repeated-symbol` option is also available to repeat the symbol for each `\per`.

```
\unit[per-mode = repeated-symbol]{\joule\per\mole\per\kelvin}
```

J/mol/K

The use of a symbol can be restricted to the case where exactly one is required: the setting `single-symbol` will use a symbol if and only if there are one or more positive powers and exactly one negative power. In other cases, powers are used.

$\mathrm{10\,m^{-1}}$
20 m/s
$\mathrm{30\,J\,mol^{-1}\,K^{-1}}$

```
\sisetup{per-mode = single-symbol}
\qty{10}{\per\metre} \\
\qty{20}{\metre\per\second} \\
\qty{30}{\joule\per\mole\per\kelvin}
```

It is possible for the behavior of the `\per` function to depend on the prevailing math style. Setting either `display-per-mode` or `inline-per-mode` independently can be used to achieve this. For example, the following example will use the `symbol` setting for in line math, and the `fraction` setting when used in display math.

$\mathrm{J/(mol\,K)}$

$\frac{\mathrm{J}}{\mathrm{mol\,K}}$

$\mathrm{J/(mol\,K)}$

$\frac{\mathrm{J}}{\mathrm{mol\,K}}$

$\mathrm{J/(mol\,K)}$

```
\sisetup{
  display-per-mode = fraction ,
  inline-per-mode = symbol
}%
$ \unit{\joule\per\mole\per\kelvin} $
\[ \unit{\joule\per\mole\per\kelvin} \]
   \unit{\joule\per\mole\per\kelvin} \\
$
  \displaystyle
  \unit{\joule\per\mole\per\kelvin}
$
\[
  \textstyle
  \unit{\joule\per\mole\per\kelvin}
\]
```

`per-symbol-script-correction`

When using the `symbol` setting for `per-mode`, there may the need to adjust spacing between a superscript power and the symbol. This is provided as a command to be inserted between the two items: the standard value is a thin negative space, `\!`.

```
\sisetup{per-mode = symbol}%
\unit{\cm\cubed\per\gram} \\
\unit[per-symbol-script-correction = ]{\cm\cubed\per\gram}
```

$\mathrm{cm^3/g}$
$\mathrm{cm^3/g}$

`sticky-per`

By default, `\per` applies only to the next unit given.[3] By setting the `sticky-per` flag, this behavior is changed so that `\per` applies to all subsequent units.

```
\unit{\pascal\per\gray\henry} \\
\unit[sticky-per]{\pascal\per\gray\henry}
```

$\mathrm{Pa\,Gy^{-1}\,H}$
$\mathrm{Pa\,Gy^{-1}\,H^{-1}}$

`qualifier-mode` `qualifier-phrase`

Unit qualifiers can be printed in three different formats, set by the `qualifier-mode` option. The standard setting is `subscript`, while the options `bracket`, `combine` and `phrase` are also possible. With the last settings, powers can lead to ambiguity and are automatically detected and brackets added as appropriate.

```
\unit{\kilogram\of{pol}\squared\per\mole\of{cat}\per\hour} \\
\unit[qualifier-mode = bracket]
  {\kilogram\of{pol}\squared\per\mole\of{cat}\per\hour} \\
\unit[qualifier-mode = combine]
  {\deci\bel\of{i}}
```

$\mathrm{kg^2_{pol}\,mol^{-1}_{cat}\,h^{-1}}$
$\mathrm{kg(pol)^2\,mol(cat)^{-1}\,h^{-1}}$
dBi

The `phrase` option is used with `qualifier-phrase`, which allows for example a space or other linking text to be inserted.

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

Table 20: Options for quantities.

Option name	Type	Default
allow-quantity-breaks	Switch	false
extract-mass-in-kilograms	Switch	true
prefix-mode	Choice	input
quantity-product	Literal	\,
separate-uncertainty-units	Choice	bracket

```
\sisetup{qualifier-mode = phrase, qualifier-phrase = \ }%
\unit{\kilogram\of{pol}\squared\per\mole\of{cat}\per\hour} \\
\sisetup{qualifier-phrase = \ \mbox{of}\ }%
\unit{\kilogram\of{pol}\squared\per\mole\of{cat}\per\hour}
```

$\mathrm{kg\ pol^{2}\ mol\ cat^{-1}\ h^{-1}}$
$\mathrm{kg\ of\ pol^{2}\ mol\ of\ cat^{-1}\ h^{-1}}$

power-half-as-sqrt

In some cases, the power of 0.5 is shown by giving the unit symbol as a square root. This can be enabled by setting power-half-as-sqrt to true

```
\unit{\Hz\tothe{0.5}} \\
\unit[power-half-as-sqrt]{\Hz\tothe{0.5}}
```

$\mathrm{Hz^{0.5}}$
$\sqrt{\mathrm{Hz}}$

parse-units

Normally, siunitx is used with the unit parse enabled, and only prints units directly if there is literal input. However, if the input is known to be essentially consistent and high performance is desired, then the parser can be turned off using the parse-units switch.

300 MHz
300 MHz

```
\qty{300}{\MHz} \\
\qty[parse-units = false]{300}{\MHz}
```

forbid-literal-units

Some users may prefer to completely disable the use of literal input in units, for example to enforce consistency. This can be accomplished by setting the forbid-literal-units switch. With this option enabled, only macro-based units can be used in a document. This only applies when parse-units is true.

unit-font-command

The command used to set unit themselves may be adjusted using the unit-font-command option. This is typically set to \mathrm.

```
\unit{\lumen} \\
\unit[unit-font-command = \mathit]{\lumen}
```

lm
lm

4.11 Quantities

Some options apply to quantities (the combination of a number and a unit), rather than to the numbers or units alone (Table 20).

allow-quantity-breaks

Usually, the combination of a number and unit is regarded as a single mathematical entity which should not be split across lines. However, there are cases (very long units, narrow columns, *etc.*) where breaks may be needed. This can be turned on using the allow-quantity-breaks option.

X
10 m
X 10
m

```
\begin{minipage}{3cm}
  % Gives an underfull hbox
  X\hspace{2.4cm}\qty{10}{\metre} \\
  \sisetup{allow-quantity-breaks}
  X\hspace{2.4cm}\qty{10}{\metre}
\end{minipage}
```

`quantity-product` The product symbol between the number and unit is set using the `quantity-product` option.

```
\qty{2.67}{\farad} \\
\qty[quantity-product = \ ]{2.67}{\farad} \\
\qty[quantity-product = ]{2.67}{\farad}
```

2.67 F
2.67 F
2.67F

`prefix-mode` `extract-mass-in-kilograms` The unit prefixes (`\kilo`, *etc.*) are normally given as letters. However, the package can convert these into numerical powers. This is controlled by the `prefix-mode` option, which takes the values `input`, `combine-exponent` and `extract-exponent`. The `input` setting leaves the prefixes unchanged. Using `combine-exponent` will add any exponent amount from the number to the first unit: this will modify any existing prefix. Finally, using `extract-exponent` will remove all prefixes and express them as an exponent. The treatment of kilograms in this case can be set using `extract-mass-in-kilograms`: when `true`, the *kilo* prefix is retained as part of the unit. This will mean that all grams are converted to kilograms.

```
\qty{1e3}{\metre\second} \\
\qty[prefix-mode = combine-exponent]{1e3}{\metre\second} \\
\qty{10}{\kilo\gram\squared\deci\second} \\
\qty[prefix-mode = extract-exponent]{10}{\kilo\gram\squared\deci\second}\\
\qty[prefix-mode = extract-exponent]{7.5}{\gram} \\
\sisetup{extract-mass-in-kilograms = false}
\qty{10}{\kilo\gram\squared\deci\second} \\
\qty[prefix-mode = extract-exponent]{10}{\kilo\gram\squared\deci\second} \\
\qty[prefix-mode = extract-exponent]{7.5}{\gram} \\
```

$1 \times 10^{3}\,\mathrm{m\,s}$
$1\,\mathrm{km\,s}$
$10\,\mathrm{kg^{2}\,ds}$
$10 \times 10^{-1}\,\mathrm{kg^{2}\,s}$
$7.5 \times 10^{-3}\,\mathrm{kg}$
$10\,\mathrm{kg^{2}\,ds}$
$10 \times 10^{5}\,\mathrm{g^{2}\,s}$
$7.5\,\mathrm{g}$

`separate-uncertainty-units` When a number has multiple parts (such as a separate uncertainty) then the unit must apply to all parts of the number. How this is shown is controlled using the `separate-uncertainty-units` options. The standard setting is `brackets`, which will place the entire numerical part in brackets and use a single unit symbol. Alternative options are `repeat` (print the unit for each part of the number) and `single` (print only one unit symbol: mathematically incorrect).

Table 21: Options for tabular material.

Option name	Type	Default
table-align-comparator	Switch	true
table-align-exponent	Switch	true
table-align-text-after	Switch	true
table-align-text-before	Switch	true
table-align-uncertainty	Switch	true
table-alignment	Meta	center
table-alignment-mode	Choice	marker
table-auto-round	Switch	false
table-column-width	Length	0pt
table-fixed-width	Switch	false
table-format	Special	2.2
table-model-setup	Literal	⟨*empty*⟩
table-number-alignment	Choice	center
table-text-alignment	Choice	center

```
\sisetup{uncertainty-mode = separate}
\qty{12.3(4)}{\kilo\gram} \\
\qty[separate-uncertainty-units = bracket]{12.3(4)}{\kilo\gram} \\
\qty[separate-uncertainty-units = repeat]{12.3(4)}{\kilo\gram}  \\
\qty[separate-uncertainty-units = single]{12.3(4)}{\kilo\gram}
```

$(12.3 \pm 0.4)\,\mathrm{kg}$
$(12.3 \pm 0.4)\,\mathrm{kg}$
$12.3\,\mathrm{kg} \pm 0.4\,\mathrm{kg}$
$12.3 \pm 0.4\,\mathrm{kg}$

4.12 Tabular material

Processing of material in tables obeys the same settings as described for the functions already described. However, there are some settings which apply only to the layout of tabular material (Table 21).

table-alignment-mode

The method used by siunitx to align numbers is selected using the table-alignment-mode option, which may be one of marker, format or ⟨*none*⟩. With the standard setting, marker, the package centers the decimal marker in a tabular column, potentially leaving white space at the shorter end of a number. The format mode uses information from the table-format key to construct a model: this is then used to define the space available to a number. For asymmetrical numbers, this method is strongly preferable. Finally, none disables alignment entirely: numbers are simply parsed.

table-number-alignment

When table-alignment-mode is set to format or none, the placement of the number 'block' within the cell as a whole is set by the table-number-alignment option, which may be one of left, center or right. (When table-alignment-mode is set to marker, the decimal marker is always centered in the cell.) The different alignment choices are illustrated in Table 22, which uses somewhat exaggerated column headings to show the relative position of the cell contents.

```
\begin{table}
  \caption{Aligning the \texttt{S} column.%
```

Table 22: Aligning the S column.

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

```
    \label{tab:S:align}}
  \centering
  \sisetup{table-format = 2.4, table-alignment-mode = format}
  \begin{tabular}{@{}
    S[table-alignment-mode = marker]
    S[table-number-alignment = center]
    S[table-number-alignment = left]
    S[table-number-alignment = right]
  @{}}
  \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}
```

When the alignment mode is set to `none`, number are simply collected and parsed without any further processing, as illustrated in Table 23.

```
\begin{table}
  \caption{Parsing without aligning in an \texttt{S} column.%
    \label{tab:S:parse}}
  \begin{tabular}
    {@{}
      S
      S[table-alignment-mode = none]
    @{}}
    \toprule
      {Decimal-centered} &
      {Simple centering} \\
    \midrule
      12.345   & 12.345   \\
      6,78     & 6,78     \\
      -88.8(9) & -88.8(9) \\
      4.5e3    & 4.5e3    \\
    \bottomrule
  \end{tabular}
\end{table}
```

`table-format` When the `table-alignment-mode` is set to `format`, siunitx uses the information set in `table-format` to construct a 'model' which defines the space to reserve for a number.

Table 23: Parsing without aligning in an S column.

Decimal-centered	Simple centering
12.345	12.345
6.78	6.78
−88.8(9)	−88.8(9)
4.5×10^3	4.5×10^3

The table-format key is interpreted in much the same way as a table cell. The numerical part should consist of a number showing how many figures to reserve in each part of the input, plus any comparators, signs, *etc.* A variety of examples are given in Table 24.

```
\begin{table}
  \caption{Reserving space in \texttt{S} columns.%
    \label{tab:S:format}}
  \sisetup{
    table-alignment-mode   = format,
    table-number-alignment = center,
  }
  \begin{tabular}{@{}
    S[table-format = 2.2]
    S[table-format = 2.2, table-number-alignment = right]
    S[table-format = 2.2(1)]
    S[table-format = 2.2(1), uncertainty-mode = separate]
    S[table-format = +2.2]
    S[table-format = 2.2e1]
   @{}}
  \toprule
      {Values}
    & {Values}
    & {Values}
    & {Values}
    & {Values}
    & {Values} \\
  \midrule
     2.3  &  2.3  &  2.3(5)  &  2.3(5)  &   2.3  &  2.3e8  \\
    34.23 & 34.23 & 34.23(4) & 34.23(4) &  34.23 & 34.23   \\
    56.78 & 56.78 & 56.78(3) & 56.78(3) & -56.78 & 56.78e3 \\
     3,76 &  3,76 &  3,76(2) &  3.76(2) & +-3.76 &      e6 \\
  \bottomrule
  \end{tabular}
\end{table}
```

It is important to note that any parts of a number *not* specified in the table format argument are set to be absent (the number of figures is set to zero). Setting the table-format option also resets table-alignment-mode to format.

table-model-setup

In some tables (particularly those using bold extended fonts), it may be necessary to insert additional information when creating the 'model'. This is handled using the table-model-setup key. An example of the use of this key is given in Section 10.13.

Space for material before and after the S column can be reserved by giving model text as part of the table-format key. This is then used to provide the necessary gap while maintaining alignment (Table 25).

Table 24: Reserving space in S columns.

Values	Values	Values	Values	Values	Values
2.3	2.3	2.3(5)	2.3 ± 0.5	2.3	2.3×10^{8}
34.23	34.23	34.23(4)	34.23 ± 0.04	34.23	34.23
56.78	56.78	56.78(3)	56.78 ± 0.03	-56.78	56.78×10^{3}
3.76	3.76	3.76(2)	3.76 ± 0.02	± 3.76	10^{6}

Table 25: Text before and after numbers.

Values
2.3456
34.2345[a]
56.7835
now 90.473

```
\begin{table}
  \caption{Text before and after numbers.%
    \label{tab:S:ends}}
  \sisetup{table-format = {now }2.4{\textsuperscript{\emph{a}}}}
  \begin{tabular}{@{}S@{}}
  \toprule
  {Values} \\
  \midrule
          2.3456  \\
         34.2345  \textsuperscript{\emph{a}}\\
         56.7835  \\
    now~ 90.473   \\
  \bottomrule
  \end{tabular}
\end{table}
```

table-align-exponent
table-align-uncertainty
table-align-comparator

When printing exponents in tables, there is a choice of aligning the exponent parts or having these close up to the mantissa. This is controlled by the `table-align-exponent` option (Table 26). Similarly, uncertainty parts which are printed separately from the mantissa can be aligned or closed up. This is set by the `table-align-uncertainty` option (Table 27). Finally, the same approach is available for the comparator with the `table-align-comparator` option (Table 28).

```
\begin{table}
  \caption{The \opt{table-align-exponent} option.%
    \label{tab:align:exp}}
  \sisetup{table-format = 1.3e2}
  \begin{tabular}{@{}SS[table-align-exponent = false]@{}}
    \toprule
      {Header} & {Header} \\
    \midrule
      1.2e3    & 1.2e3    \\
      1.234e56 & 1.234e56 \\
    \bottomrule
  \end{tabular}
```

Table 26: The table-align-exponent option.

Header	Header
1.2×10^{3}	1.2×10^{3}
1.234×10^{56}	1.234×10^{56}

Table 27: The table-align-uncertainty option.

Header	Header
1.2 ± 0.1	1.2 ± 0.3
1.234 ± 0.005	1.234 ± 0.005

```
\end{table}

\begin{table}
  \caption{The \opt{table-align-uncertainty} option.%
    \label{tab:align:uncert}}
  \sisetup{
    uncertainty-mode = separate,
    table-format = 1.3(1),
  }
  \begin{tabular}{@{}SS[table-align-uncertainty = false]@{}}
    \toprule
      {Header} & {Header} \\
    \midrule
      1.2(1)   & 1.2(3)    \\
      1.234(5) & 1.234(5) \\
    \bottomrule
  \end{tabular}
\end{table}

\begin{table}
  \caption{The \opt{table-align-comparator} option.%
    \label{tab:align:comp}}
  \sisetup{table-format = >2.2}
  \begin{tabular}{@{}SS[table-align-comparator = false]@{}}
    \toprule
      {Header} & {Header} \\
    \midrule
      >  1.2  & >  1.2  \\
      < 12.34 & < 12.34 \\
    \bottomrule
  \end{tabular}
\end{table}
```

table-align-text-before
table-align-text-after

Note markers are often given in tables after the numerical content. It may be desirable for these to close up to the numbers. Whether this takes place is controlled by the table-align-text-before and …-after option (Table 29).

```
\begin{table}
  \caption{Closing notes up to text.%
```

Table 28: The table-align-comparator option.

Header	Header
> 1.2	>1.2
<12.34	<12.34

```
  \label{tab:S:notes}}
\newrobustcmd\NoteMark[1]{%
  \textsuperscript{\emph{#1}}%
}
\sisetup{table-format = {\NoteMark{a}}2.4}
\begin{tabular}{@{}
  S
  S[table-align-text-before = false]
@{}}
\toprule
  {Values}             & {Values} \\
\midrule
                2.3456 &               2.3456 \\
  \NoteMark{a}  4.234  & \NoteMark{a}  4.234 \\
  \NoteMark{b}   .78   & \NoteMark{b}   .78  \\
  \NoteMark{d} 88      & \NoteMark{d} 88     \\
\bottomrule
\end{tabular}
\hspace{\fill}%
\sisetup{table-format = 2.4\NoteMark{a}}
\begin{tabular}{@{}
  S
  S[table-align-text-after = false]
@{}}
\toprule
  {Values}             & {Values} \\
\midrule
   2.3456              & 2.3456 \\
  34.234  \NoteMark{a} & 34.234 \NoteMark{a} \\
  56.78   \NoteMark{b} & 56.78  \NoteMark{b} \\
  90.4    \NoteMark{c} & 90.4   \NoteMark{c} \\
  88      \NoteMark{d} & 88     \NoteMark{d} \\
\bottomrule
\end{tabular}
\end{table}
```

table-auto-round

The contents of table cells can automatically be rounded or zero-filled to the number of decimal digits given for the decimal part of the table-format option. This mode is activated using the table-auto-round switch, as illustrated in Table 30.

```
\begin{table}
  \centering
  \caption{The \opt{table-auto-round} option.%
    \label{tab:S:auto}}
  \sisetup{table-format = 1.3}
  \begin{tabular}{@{}SS[table-auto-round]@{}}
    \toprule
```

Table 29: Closing notes up to text.

Values	Values
2.3456	2.3456
[a] 4.234	[a]4.234
[b] 0.78	[b]0.78
[d]88	[d]88

Values		Values
2.3456		2.3456
34.234	[a]	34.234[a]
56.78	[b]	56.78[b]
90.4	[c]	90.4[c]
88	[d]	88[d]

Table 30: The `table-auto-round` option.

Header	Header
1.2	1.200
1.2345	1.235

```
      {Header} & {Header} \\
    \midrule
      1.2    & 1.2    \\
      1.2345 & 1.2345 \\
    \bottomrule
  \end{tabular}
\end{table}
```

`parse-numbers` When the `parse-numbers` option is set to `false`, then the alignment code for tables takes a different approach. The output is always set in math mode, and alignment takes place at the first decimal marker. This is achieved by making it active in math mode. When reserving space for content only the integer and decimal values for the mantissa are considered (Table 31).

```
\begin{table}
  \caption{Aligning without parsing.%
    \label{tab:S:nonparsed}}
  \sisetup{
    parse-numbers = false,
    table-format  = 3.3
  }
  \centering
  \begin{tabular}{@{}
    S
    S[table-number-alignment = center]
    S[table-number-alignment = right]
    S[table-number-alignment = left]
    @{}}
  \toprule
      {Some values}
    & {Some values}
    & {Some values}
    & {Some values} \\
  \midrule
     2.35  &    2.35  &    2.35  &   2.35   \\
    34.234 &   34.234 &   34.234 &  34.234  \\
```

Table 31: Aligning without parsing.

Some values	Some values	Some values	Some values
2.35	2.35	2.35	2.35
34.234	34.234	34.234	34.234
56.783	56.783	56.783	56.783
3.762	3.762	3.762	3.762
$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$

```
    56.783 &   56.783 &   56.783 &  56.783  \\
     3,762 &    3,762 &    3,762 &   3.762  \\
  \sqrt{2} & \sqrt{2} & \sqrt{2} & \sqrt{2} \\
  \bottomrule
  \end{tabular}
\end{table}
```

drop-exponent

In cases where data cover a range of values, printing using a fixed exponent in a table may make presentation clearer. In these cases, dropping the exponent value from the table is useful. The general numerical options `drop-exponent` combined with `exponent-mode = fixed` can be used to achieve this (Table 32).

```
\begin{table}
  \caption{The \opt{drop-exponent} option.%
    \label{tab:exp:omit}}
  \begin{tabular}{@{}
    S[table-format = 1.1e1]
    S[
      drop-exponent  = true,
      exponent-mode  = fixed,
      fixed-exponent = 3,
      table-format   = 2.1,
    ]
  @{}}
    \toprule
      {Header} &
        \multicolumn{1}{c@{}}{Header / \num[print-unity-mantissa = false]{e3}} \\
    \midrule
      1.2e3 & 1.2e3 \\
       3e2  &   3e2 \\
     1.0e4  & 1.0e4 \\
    \bottomrule
  \end{tabular}
\end{table}
```

table-column-width
table-fixed-width

Usually, the width of the numerical column is allowed to vary depending on the content. However, there are cases where a strictly fixed width is desirable. For these cases, the `table-fixed-width` and `table-column-width` options are available. The `table-fixed-width` option activates fixed-width columns, whilst `table-column-width` defines the target width (Table 33). Setting `table-column-width` to a positive value automatically enables `table-fixed-width`.

```
\begin{table}
  \caption{Fixed-width columns.%
```

Table 32: The `drop-exponent` option.

Header	Header / 10^3
1.2×10^3	1.2
3×10^2	0.3
1.0×10^4	10

Table 33: Fixed-width columns.

Flexible	Fixed
1.23	1.23
45.6	45.6

```
    \label{tab:width:fixed}}
  \begin{tabular}
    {@{}
      S
      S[table-column-width = 2cm]
    @{}}
    \toprule
      {Flexible} &
      {Fixed}    \\
    \midrule
      1.23 & 1.23 \\
      45.6 & 45.6 \\
    \bottomrule
  \end{tabular}
\end{table}
```

The `table-column-width` option can also be used to achieve special effects. One example is centring a column of numbers under a wide heading, with the numbers themselves right-aligned (Table 34).

```
\begin{table}
  \centering
  \caption{Right-aligning under a heading.%
    \label{tab:width:special}}
  \settowidth{\mylength}{Long header}
  \sisetup{
    table-alignment-mode   = none      ,
    table-column-width     = \mylength ,
    table-number-alignment = right
  }
  \begin{tabular}{@{}S@{}}
  \toprule
  {Long header} \\
  \midrule
      12.33 \\
       2    \\
    1234    \\
   \bottomrule
```

Table 34: Right-aligning under a heading.

Long header
12.33
2
1234

Table 35: Aligning text in S columns.

Values	Values	Values
992.435	992.435	992.435
7734.2344	7734.2344	7734.2344
56.7834	56.7834	56.7834
3.7462	3.7462	3.7462

```
  \end{tabular}
\end{table}
```

table-text-alignment
table-alignment

Cell contents which are not part of a number can be protected using braces, as illustrated. Cells which contain no numerical data at all are aligned using the setting specified by the `table-text-alignment` option, which recognises the values `center`, `left`, `none` and `right` (Table 35). The setting `none` is intended for use with the package tabularray, which carries out its own alignment of textual values.

```
\begin{table}
  \caption{Aligning text in \texttt{S} columns.%
    \label{tab:S:text}}
  \sisetup{table-format = 4.4}
  \centering
  \begin{tabular}{@{}
    S
    S[table-text-alignment = left]
    S[table-text-alignment = right]
  @{}}
  \toprule
      {Values}
    & {Values}
    & {Values} \\
  \midrule
    992.435  &  992.435  &  992.435  \\
   7734.2344 & 7734.2344 & 7734.2344 \\
     56.7834 &   56.7834 &   56.7834 \\
      3,7462 &    3,7462 &    3,7462 \\
  \bottomrule
  \end{tabular}
\end{table}
```

table-alignment

The table alignment options `table-number-alignment` and `table-text-alignment` can be set to the same value using the `table-alignment` option. This will set all three alignment options to the same value (one of `center`, `right` or `left`).

4.13 Locale options

`locale` siunitx allows the user to switch between the typographic conventions of different (geographical) areas by using locales. Currently, the package is supplied with configurations for locales `UK`, `US`, `DE` (Germany), `IT` (Italian) `PL` (Poland), `FR` (French), `SI` (Slovene) and `ZA` (South Africa). The `locale` option is used to switch to a particular locale.

1.234 m	`\qty{1.234}{\metre}\\`
6,789 m	`\qty[locale = DE]{6.789}{\metre}`

4.14 Preamble-only options

`list-input-separator` `product-input-separator` The separator between *input* elements of lists (`\numlist`, `\qtylist`) and products (`\numproduct`, `\qtyproduct`) can be adjusted using the options `list-input-separator` and `product-input-separator`, respectively. These are load-time only options as the separator here is part of the 'document interface' rather than part of the individual number(s). The supplied value should be a *single* marker.

`table-column-type` The letter(s) used to create table columns can be adjusted using the `table-column-type` option. The standard setting is `S`, but one or more letters may be used: these must be single tokens. For example, provided the `numprint` package has not been loaded, the letter `n` could be used as this would suggest a numerical column.

5 Significant new features updates

Here, a short reminder for users of new features added since the release of version 3 of the package is given.

5.1 Version 3.1

- Extend support for complex numbers to cover polar form including new options to control output
- Parse and format multiple uncertainties; new options added to allow descriptions of each uncertainty
- Customisation of text sub/superscript output
- Selectable group size when dividing digits
- Finer controls for 'per mode' output
- Support for negative zero values and controlled output for zero values
- Allow negative values to round to zero
- Addition unit abbreviations for capacitance, power and magnetic flux
- Portuguese locale

5.2 Version 3.2

- 'Threshold' approach to producing exponent notation: similar to a calculator
- Finer control of printing plus signs using new options to split mantissa and exponent
- Support for updated prefixes introduced by BIPM
- Revised alignment of uncertainties in tabulars

5.3 Version 3.3

- Support for languages which need a begin as well as end phrase in ranges, *e.g.* Italian, was added; this includes appropriate supporting options
- Support for asymmetric uncertainties (tolerances)
- Rounding extended to allow truncation (rounding down) for both main values and when using the uncertainty part
- Finer control of the appearance of brackets in ambiguous output introduced by adding options for different contexts
- Control of list and product input and output extended
- Finer control of table alignment setup for bold values
- New unit abbreviations for magnetic fields

5.4 Version 3.4

- Support uncertainties in seconds part of arc angles
- Control creation of minute and second components on conversion of integer angles to arc format
- Ability to simplify 'asymmetric' uncertainties with equal components in both directions
- Combination of all control for angle unit formatting into `\degree`, `\arcminute` and `\arcsecond` unit macros, with `number-angle-product` deprecated
- Support for more varied printing of complex numbers phases

6 Upgrading from version 2

The package has been largely re-written internally between versions 2 and 3. A significant number of key–value settings have new, more descriptive, names. Where possible, older names are mapped to newer ones internally: you will be warned in the log if this is the case.

It is possible to use the LaTeX 2_ε kernel mechanism to load the last version 2 release for documents that cannot be successfully processed using version 3. This can be achieved using

```
\usepackage{siunitx}[=2021-04-09]
```

or

```
\usepackage{siunitx}[=v2]
```

This approach will work with older systems which still have version 2 installed, meaning that you can reliably use it to work between systems with different versions of siunitx.

\SI
\SIlist
\SIrange
\si

In version 3, the document commands have been revised to be more descriptive. As such, the commands `\SI`, `\SIlist`, `\SIrange` and `\si` remain available but are not recommended for use in new documents. Use the new `\qty`… commands instead: they are clearer and in some cases very slightly faster.

Some changes have been made to the semantics of commands or options. Most notably

- prefixes cannot now be given without units;
- prefixes can only be interconverted with numbers as part of a quantity, not as standalone units.

See Section 10.16 for how to work with the new approach if you want to print prefix information.

The font control system has been completely re-written for version 3. The method used is entirely different from version 2. Emulation is therefore not provided for all outcomes: if you need non-standard font settings, you will need to adjust your source. See Section 4.2 for more details on the options available in this area.

The input approach for version 3 is slightly more structured and restricted than for version 2. As well as the updated names for document commands, this means that

- Products of numbers must now be given using the dedicated `\numproduct` and `\qtyproduct` commands;
- Quotients of numbers are only supported as literals;
- Complex values need to be given using the dedicated command `\complexnum`.

The option `round-integer-to-decimal = false` has been removed, and whilst there is not a direct replacement, users are likely to find that `round-pad = false` offers the desired outcome.

A new approach has been taken to providing non-Latin symbols for use in units: these are now handled directly where needed, for example in the definition of the `\micro` prefix.

Translation of fixed strings is now carried out using the translations package. If you have manually set up translations in version 2 using translator, you will need to load it manually.

The letter used for a numerical tabular column can now be selected by the user: the letter `S` has been retained as the standard interface. The unit column (`s`) has been removed from this release. It can be emulated using the collcell package, for example

```
\usepackage{collcell}
\newcolumntype{s}{>{\collectcell\unit}c<{\endcollectcell}}
```

or

```
\usepackage{collcell}
\newcolumntype{s}{>{\collectcell\si}c<{\endcollectcell}}
```

If you are using `table-column-width` to have fixed-width columns, you also now need `table-fixed-width` to set this option active.

Direct support for loading a local configuration file, `siunitx.cfg`, has been removed. However, the approach described in Section 10.17 may be used to achieve the same effect with the additions more clearly shown in document sources.

The command `\SendSettingsToPgf` is deprecated, and should be replaced simply by setting the appropriate `\pgfkeys` in parallel to `\sisetup`.

7 Unit changes made by BIPM

In addition to the changes in the siunitx package described in Section 6, there are changes in the units defined by the BIPM in the 9th Edition of the SI Brochure which are reflected here. There are two major areas of change.

The first is in respect of units accepted for use with SI units. In the 8th Edition of the SI Brochure, the following units were listed as accepted for use in specialist fields

- ångström (`\angstrom`)
- bar (`\bar`)
- barn (`\barn`)
- knot (`\knot`)
- millimetre of mercury (`\mmHg`)
- nautical mile (`\nauticalmile`)

These are no longer listed in the 9th Edition, and so are deprecated as pre-defined units by siunitx. These units will issue a warning on first use, and users should add their own definitions to the start of their sources to avoid this.

Secondly, the move to a new definition of base units means that the table of units determined experimentally has been removed from the SI Brochure. This covers the following units defined by siunitx in previous releases

- `\bohr`
- `\clight`
- `\electronmass`
- `\elementarycharge`
- `\hartree`
- `\planckbar`

These are also deprecated in siunitx and users should provide their own definitions.

In addition to these two major blocks, the unit `\atomicmassunit` has similar deprecated status: this was listed as with experimentally-determined units in the 8th Edition of the SI Brochure but is equivalent to the dalton, a unit which remains accepted.

8 Localisation

The translations package provides a structured framework for localisation of words and phrases. In particular, it offers the `\GetTranslation` macro, which will provide appropriate translations based on the current babel or polyglossia language setting.

If translations is available, siunitx will load it and alter the standard settings for the `list-final-separator` and `range-phrase` options to read:

```
\sisetup{
  list-final-separator = { \GetTranslation{and} },
  list-pair-separator  = { \GetTranslation{and} },
  range-phrase         = { \GetTranslation{to (numerical range)} },
}
```

The setting for `range-open-phrase` is also adjusted to support a setting for `range-open-phrase`: this requires some conditionals so is more complex.

If the current language is known to the translations package then the result will be localised text. The preamble for this manual loads English, French, German, Italian, Polish, Spanish, Catalan, Portuguese and Brazilian as options, and also loads the babel package:

```
% In English by default
\numlist{1;2;3} \\
\numrange{1}{10} \\
\selectlanguage{french}%
\numlist{1;2;3} \\
\numrange{1}{10} \\
\selectlanguage{german}%
\numlist{1;2;3} \\
\numrange{1}{10} \\
\selectlanguage{italian}%
\numlist{1;2;3} \\
\numrange{1}{10} \\
\selectlanguage{polish}%
\numlist{1;2;3} \\
\numrange{1}{10} \\
\selectlanguage{spanish}%
\numlist{1;2;3} \\
\numrange{1}{10} \\
\selectlanguage{catalan}%
\numlist{1;2;3} \\
\numrange{1}{10} \\
\selectlanguage{portuguese}%
\numlist{1;2;3} \\
\numrange{1}{10} \\
\selectlanguage{brazilian}%
\numlist{1;2;3} \\
\numrange{1}{10}
```

1, 2 and 3
1 to 10
1, 2 et 3
1 à 10
1, 2 und 3
1 bis 10
1, 2 e 3
da 1 a 10
1, 2 i 3
1 do 10
1, 2 y 3
1 a 10
1, 2 i 3
1 a 10
1, 2 e 3
1 a 10
1, 2 e 3
1 a 10

9 Compatibility with other packages

In general, siunitx should be usable with other packages without interference.

When the physics package is loaded before siunitx, the command \qty is not defined. Users may use the version 2 command \SI, which can be used as a drop-in replacement for \qty. Alternatively, if the siunitx definition is preferred, you may use

```
\AtBeginDocument{\RenewCommandCopy\qty\SI}
```

and use the longer name \quantity to access the functionality of the physics package.

10 Hints for using siunitx

10.1 Problematic font encodings

The standard settings in siunitx assume that 'sensible' input and font encoding values prevail. The input encoding is assumed to be UTF-8 in all cases. With pdfLaTeX, the font encoding should be T1, whereas for XeLaTeX and LuaLaTeX, TU (Unicode font encoding) is expected.

Some packages, for example newtxtext or stix2, either force T1 or do not anticipate TU correctly with XeLaTeX and LuaLaTeX. In these cases, the symbols used by siunitx may be incorrect. If correcting the font encoding is not possible, you will need to re-declare the relevant units using symbol definitions which account for this non-standard setup.

10.2 Adjusting \litre and \liter

As detailed earlier, the unit macros \litre and \liter are both available for litres. With the standard settings, \liter is defined as

```
\DeclareSIUnit\liter{\litre}
```

meaning that \litre is the 'canonical' unit. This follows the same relationship as exists between \metre and \meter.

In contrast to metres, however, there is more likelihood of users wishing to adjust the appearance of litres: both 'l' and 'L' are commonly used. The recommended approach to adjustment is to re-declare the \litre macro, as \liter will follow automatically.

```
\DeclareSIUnit\litre{l}
```

10.3 Ensuring text or math output

The macros \ensuremath and \text should be used to ensure that a particular item is always printed in the desired mode. Some mathematical output does not work well in \mathrm (the font setting used by siunitx for printing units). The easiest way to solve this is to use the construction \text{\ensuremath{...}}, which will print the material in the standard mathematics font without affecting the rest of the output. In some cases, simply forcing \mathnormal will suffice, but this is less reliable with non-Latin characters.

10.4 Including a literal hyphen inside \text

In most cases, a - character inside \text will represent a minus. The package will therefore replace it with \textminus. However, this may be problematic if you want a hyphen, or if you are using - in a piece of TeX syntax. You can prevent this by using a second set of braces

Table 36: Values as macros in S columns.

Some Values
12 348.812 34
12 348.812 34
12348.81234
12348.81234
1234 8.8 1234

```
\DeclareSIUnit\electronvolt{\text{{e\kern -0.1em V}}}
```

or by defining a protected command that will yield a hyphen on typesetting

```
\usepackage{etoolbox}
\newrobustcmd*\hyphenminus{-}
\DeclareSIUnit\electronvolt{\text{e\kern \hyphenminus0.1em V}}
```

10.5 Expanding content in tables

When processing tables, siunitx will expand anything stored inside a macro, unless it is long or protected. LaTeX 2_ε robust commands are also detected and are not expanded (Table 36). Values which would otherwise be expanded can be protected by wrapping them in a set of braces. As TeX itself will expand the first token in a table cell before siunitx can act on it, using the ε-TeX protected mechanism is the recommended course of action to prevent expansion of macros in table cells. (As is shown in the table, TeX's expansion of LaTeX 2_ε robust commands can lead to unexpected results.)

```
\begin{table}
  \caption{Values as macros in \texttt{S} columns.%
    \label{tab:xmpl:macro}}
  \newcommand*\myvaluea{1234}
  \newcommand\myvalueb{1234}
  \DeclareRobustCommand*\myvaluec{1234}
  \protected\def\myvalued{1234}
  \begin{tabular}{@{}S@{}}
  \toprule
    {Some Values} \\
  \midrule
     \myvaluea  8.8 \myvaluea  \\ % Both expanded
     \myvalueb  8.8 \myvalueb  \\ % First expanded by TeX
                                  % to numbers
     \myvaluec  8.8 \myvaluec  \\ % First expanded by TeX
                                  % but not to numbers!
     \myvalued  8.8 \myvalued  \\ % Neither expanded
    {\myvaluea\ 8.8 \myvaluea} \\ % Neither expanded
  \bottomrule
  \end{tabular}
\end{table}
```

It is possible to use calculated values in tables. For this to work, the calculation must take place before attempting to parse the number (the parser cannot 'know' all of the possible values inside an expression). This is most conveniently handled using the xfp

Table 37: Calculated values.

Value	Doubled
66.7012	133.4024
66.0212	132.0424
64.9026	129.8052

package, which is distributed as part of the required support for siunitx. The general approach is illustrated in Table 37

```
\begin{table}
  \caption{Calculated values.%
    \label{tab:xmpl:calc}}
  \newcommand{\valuea}{66.7012}
  \newcommand{\valueb}{66.0212}
  \newcommand{\valuec}{64.9026}
  \begin{tabular}{
    @{}
    S[table-format = 2.4]
    S[table-format = 3.4]
    @{}
  }
    \toprule
      {Value} & {Doubled} \\
    \midrule
      \valuea & \fpeval{\valuea * 2} \\
      \valueb & \fpeval{\valueb * 2} \\
      \valuec & \fpeval{\valuec * 2} \\
    \bottomrule
  \end{tabular}
\end{table}
```

A more sophisticated approach is to generate the rows themselves from a database: this is illustrated in Section 10.6.

10.6 Using siunitx with datatool

As illustrated in Table 37, siunitx can be used to typeset data stored using datatool. For quickly displaying the contents of tables, datatool offers the `\DTLshowtable` macro. This will only work with `S` columns if number parsing is turned off (Table 38).

```
\DTLnewdb{moredata}
\DTLnewrow{moredata}\DTLnewdbentry{moredata}{value}{ 6.7012}
\DTLnewrow{moredata}\DTLnewdbentry{moredata}{value}{66.0212}
\DTLnewrow{moredata}\DTLnewdbentry{moredata}{value}{64.902 }
\begin{table}
  \caption{Displaying a \textsf{datatool} table.%
    \label{tab:xmpl:datatool}}
  \sisetup{parse-numbers= false, table-format = 2.4}
  \renewcommand*\dtlrealalign{S}
  \DTLdisplaydb{moredata}
\end{table}
```

Table 38: Displaying a datatool table.

value
6.7012
66.0212
64.902

Table 39: Calculated values using datatool.

Value	Doubled
66.7012	133.4024
66.0212	132.0424
64.9026	129.8052

The datatool package may also be used to create on-the-fly tables using calculations. For example, the demonstration in Table 37 may be achieved without needing to write out each row, as shown in Table 39. An extra column is used to contain the calculations in this case.

```
\begin{table}
  \caption{Calculated values using \pkg{datatool}.%
    \label{tab:xmpl:datatool-calc}}
  \DTLnewdb{data}
  \DTLnewrow{data}\DTLnewdbentry{data}{value}{66.7012}
  \DTLnewrow{data}\DTLnewdbentry{data}{value}{66.0212}
  \DTLnewrow{data}\DTLnewdbentry{data}{value}{64.9026}
  \begin{tabular}{
    @{}
    S[table-format = 2.4]
    S[table-format = 3.4]
    @{}l
    @{}
  }
    \toprule
      {Value} & {Doubled} &
    \DTLforeach{data}{\myvalue=value}{%
      \DTLiffirstrow {\\ \midrule}{\\}%
      \myvalue &            % First column
      \fpeval{\myvalue * 2} % second column
      & }\\
    \bottomrule
  \end{tabular}
\end{table}
```

10.7 Using units in section headings and bookmarks

The siunitx code is designed to work correctly with functions in headings. They will print correctly in headings and in the table of contents. As illustrated here, the standard behavior is to ignore font changes. When the hyperref package is loaded, the functions automatically 'degrade gracefully' to produce useful information in PDF bookmarks. If

you want more control over the bookmark text, use the \texorpdfstring function from hyperref, for example:

```
\section{Some text
  \texorpdfstring
    {\unit{\joule\per\mole\per\kelvin}}
    {J mol-1 K-1}%
  }
```

You may find it useful to load hyperref with the unicode option, as this will allow Ω to appear in bookmarks. Without the option, the encoding used by hyperref does not support this symbol.

10.8 A left-aligned column visually centered under a heading

When you have a column of non-related numbers, the usual advice is to make these left-aligned and then center the resulting column under the heading. With the dcolumn package, that would be done with something like D{x}{}{5.0}. This is something of an abuse of the nature of a number, but can also be done using siunitx (Table 40).

```
\begin{table}
  \caption{Formatting unrelated numbers.%
    \label{tbl:xmpl:unrel}}
  \centering
  \begin{tabular}
    {
      @{}
      S[
        table-format = 5.0,
        parse-numbers = false,
        input-decimal-markers = x
      ]
      @{}
    }
    \toprule
    \multicolumn{1}{@{}c@{}}{Header} \\
    \midrule
    120   \\
    12.3  \\
    12340 \\
    12.02 \\
    123   \\
    1     \\
    \bottomrule
  \end{tabular}
\end{table}
```

10.9 Regression tables

In some subject areas, it is common to present regression values or similar, which feature an uncertainty value in parenthesis on the line below the main value. As these are separate cells, they cannot be entered using siunitx in one value. There are a couple of

Table 40: Formatting unrelated numbers.

Header
120
12.3
12340
12.02
123
1

ways of formatting them using the package, depending on whether the values also need to be parsed.

Where parsing is not required, the most straight-forward method is available: provide a model format allowing space for an extra 'digit' at each end, which will then allow for the parenthesis. If a sign is applied to the number, it may not be necessary to add a 'digit' for the leading bracket. If parsing is also required, this approach cannot be employed. Instead, the parsing needs to be adjusted such that (and) are not treated as part of the number, and `table-align-text-before` is set to `false` such that these will be placed next to the numerical part. These methods are illustrated in Table 41.

```
\begin{table}
  \caption{Regression tables%
    \label{tab:regression}
  }
  \begin{tabular}
    {
      @{}
      S[table-format = 2.4, parse-numbers = false]
      S[table-format = +1.4, parse-numbers = false]
      S[
        input-open-uncertainty  = ,
        input-close-uncertainty = ,
        minimum-decimal-digits  = 3, % (
        table-format            = +1.3),
        table-align-text-before = false
        ]
      @{}
    }
    \toprule
    {Header} &{Header} & {Header} \\
    \midrule
    1.234   & -1.234  & -1.23   \\
    (0.053) & (0.053) & (0.053) \\
    \bottomrule
  \end{tabular}
\end{table}
```

10.10 Maximising performance

Both the number and unit parsers require significant effort in terms of TEX programming. For input that does not require such processing, the maximum performance for siunitx

Table 41: Regression tables

Header	Header	Header
1.234	−1.234	−1.230
(0.053)	(0.053)	(0.053)

can therefore be obtained by turning off both systems:

```
\qty{7.3}{\Hz} \\
\qty[parse-units = false]{7.3}{\Hz} \\
\qty[
  parse-numbers = false,
  parse-units   = false
]{7.3}{\Hz}
```

7.3 Hz
7.3 Hz
7.3 Hz

For tables, any settings that can be given before the table are only parsed once, whereas given in the optional argument to `S` they are read in every cell. As such, you should favour

```
\begin{table}
  \sisetup{...}
  \begin{tabular}{S}
  ...
```

for common settings.

10.11 Special considerations for the `\kWh` unit

The standard settings provide the `\kWh` unit set up with no spacing between the 'kW' and the 'h' unit to give 'kWh'. However, this only applies when the unit is given on its own: combinations will follow the normal rules

```
\unit{\kWh} \\
\unit{\kWh\per\metre}
```

kWh
kW h m^{-1}

This is because the unit `\kWh` is defined so that it can still be varied by altering `\kilo`, `\watt` and `\hour`, and so that the prefix can still be turned into a number. However, some users may prefer to have a non-flexible macro which never adds a space. This can be achieved by redefining `\kWh` with `\DeclareSIUnit`, by added an alternative definition

```
\DeclareSIUnit\kWh{kWh}
\DeclareSIUnit\KWH{kWh}
```

or of course by using literal unit input.

```
\unit{\KWH\per\metre}\\
\unit{kWh.m^{-1}}
```

kWh m^{-1}
kWh m^{-1}

Another point to notice is that the `\per` macro applies to the next unit, and not an entire unit combination. Thus in

```
\unit{\candela\per\kWh}
```

cd kW^{-1} h

\per applies to the watts but not to the hours. In this case, the units need to be written out in full or the sticky-per option should be used.

```
\unit{\candela\per\kilo\watt\per\hour} \\
\unit[sticky-per]{\candela\per\kWh}
```

$\mathrm{cd\,kW^{-1}\,h^{-1}}$
$\mathrm{cd\,kW^{-1}\,h^{-1}}$

10.12 Creating a column with numbers and units

Usually, numbers in a table should be given with the units in the column heading. However, there are cases where a series of data are best presented in a table but have different units. There are two ways to do this (Table 42). The first is to place the units in the first column of the table, which makes sense if there are several related items in the table. The second method is to generate two columns, one for numbers and a second for units, and then to format these to give the visual effect of a single column. The later effect is most appropriate when only one set of numbers are presented in a table. This method requires cell content is collected, easiest to do using the collcell package.

```
\begin{table}
  \caption{Tables where numbers have different units.%
    \label{tab:xmpl:mixed}}
  \hspace{\fill}%
  \begin{tabular}
    {
      @{}
      >{$}l<{$}
      S[table-format = 3.3(1)]
      S[table-format = 3.3(1)]
      @{}
    }
    \toprule
      & {One} & {Two} \\
    \midrule
    a / \unit{\pm}         &  123.4(2) &   567.8(4) \\
    \beta / \unit{\degree} & 90.34(4)  & 104.45(5)  \\
    \mu / \unit{\per\mm}   &  0.532    &   0.894    \\
    \bottomrule
  \end{tabular}
  \hspace{\fill}%
  \begin{tabular}
    {
      @{}
      S[table-format=1.3]@{\,}
      >{\collectcell\unit}l<{\endcollectcell}
      @{}
    }
    \toprule
    \multicolumn{2}{@{}c}{Heading} \\
    \midrule
    1.234 & \metre   \\
    0.835 & \candela \\
    4.23  & \joule\per\mole \\
```

Table 42: Tables where numbers have different units.

	One	Two
a/pm	123.4(2)	567.8(4)
β/°	90.34(4)	104.45(5)
μ/mm^{-1}	0.532	0.894

Heading
1.234 m
0.835 cd
4.23 J mol^{-1}

```
    \bottomrule
  \end{tabular}
  \hspace{\fill}%
\end{table}
```

10.13 Tables with heading rows

A common format for tables is to make the heading row visually distinct using a background color and bold text. If numbers appear in such a heading row within an S column then getting the appearance right can be challenging. The best approach is to make the `\bfseries` macro 'robust' (as demonstrated in Section 10.5), then to use this macro to make the heading cells bold. This approach is illustrated in Table 43, along with the use of `\rowcolor` to provide a background color. Some typical LaTeX font arrangements use a wider (extended) width for the bold font compared with the normal font. This will disturb the alignment if the `table-format` is not symmetrical. For these cases, the `table-model-setup` key may be used to apply the wider font to the 'model' used in ensuring alignment.

```
\begin{table}
  \caption{Header row in a table.%
    \label{tab:xmpl:headers}}
  \robustify\bfseries
  \centering
  \sisetup{text-series-to-math}
  \begin{tabular}
    {
      @{}
      S[table-format = 3.3]
      S[table-format = 3.6, table-model-setup = \bfseries]
      @{}
    }
    \rowcolor[gray]{0.9}
    \bfseries 123.456 & \bfseries 123.456789 \\
     23.45  &  23.45  \\
    123.4   & 123.4   \\
      3.456 &   3.456 \\
  \end{tabular}
\end{table}
```

10.14 Associating a locale with a babel language

It is possible to instruct the babel package to switch to a particular siunitx locale when changing language. This can be done using the babel `\extras⟨language⟩` system. For

Table 43: Header row in a table.

123.456	123.456 789
23.45	23.45
123.4	123.4
3.456	3.456

example, to associate the DE locale with the german babel language, the appropriate code would be

```
\addto\extrasgerman{\sisetup{locale = DE}}
```

10.15 Symbolic 'digits'

In some cases you may want to use 'digits' which do not fall within the usual set 0123456789. This can be done by setting the input-digits option, but bearing in mind that this will affect (prevent) for example rounding.

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

```
\sisetup{input-digits = 0123456789\pi}%
\num{4\pi e-7}
```

Each extra entry should be a single token, and should either have a definition which is safe in both math and text mode, or should only be used when the output mode is known. It may be necessary to make some tokens robust using etoolbox for this to work, for example

```
\robustify\dots
\sisetup{input-digits = 0123456789\dots}%
\qty{0,4066\dots}{\metre\squared}
```

$0.406\,6\ldots\ \mathrm{m}^2$

10.16 Demonstrating prefixes

As siunitx contains data about the numerical values of unit prefixes, you may wish to print this in an automated way. Prefixes cannot be given on their own, but it is possible to create a 'do nothing' unit.

```
\DeclareSIUnit\noop{\relax}
```

which can then be used to show just the prefix symbol.

Y

```
\unit{\yotta\noop}
```

To show just the numerical value of a prefix, you will need to use \qty and appropriate settings.

```
\qty[prefix-mode = extract-exponent, print-unity-mantissa = false]%
  {1}{\yotta\noop}
```

10^{24}

This may be conveniently wrapped up inside a document command, for example

```
\NewDocumentCommand\prefixvalue{m}{%
  \qty[prefix-mode=extract-exponent,print-unity-mantissa=false]{1}{#1\noop}
}
```

10.17 Creating a set of pre-defined units

There are many units which sit outside of those defined in the (current) SI Brochure which are of use to many people. Most obvious are those which have been detailed in previous editions of the Brochure, as described in Section 7, but there are many others.

It is often convenient to have a pre-defined set of useful units available without needing to copy the full set of definitions into each source file. At the same time, it is important that such sources do show that they are using units not defined by the core part of siunitx. The most straight-forward way to achieve this is to create a separate file, for example `siunitx-local-units.tex`, and place it in your local TEX tree (usually `~/texmf/tex/latex/` on Linux, `~/Library/texmf/tex/latex/` on macOS or `C:\Users\<name>\texmf\tex\latex` on Windows). This can then be loaded in the preamble using

```
% Load useful 'local' units
\input{siunitx-local-units}
```

10.18 Overloading the standard interfaces

The interfaces in siunitx deliberately split up different types of numerical input (ranges, products, etc.) for semantic reasons. This also means that there is less chance of confusion in parsing input. However, some users would prefer to overload the `\num` and `\qty` commands such that they accept multiple different types of input. There is a code-level interface that could be used for this, but most end-users will likely want to avoid this. A possible approach that covers most of the common cases is.

```
\ExplSyntaxOn
\cs_gset_eq:NN \ifstrpresent \tl_if_in:nnTF
\ExplSyntaxOff
\NewCommandCopy\oldnum\num
\NewCommandCopy\oldnumrange\numrange
\RenewDocumentCommand\num{O{}m}{%
  \ifstrpresent{#2}{x}
    {\numproduct[#1]{#2}}
    {%
      \ifstrpresent{#2}{;}
        {\numlist[#1]{#2}}
        {%
          \ifstrpresent{#2}{:}
            {\numrange[#1]{#2}}
            {\oldnum[#1]{#2}}%
        }%
    }%
}
\RenewDocumentCommand\numrange{O{}>{\SplitList{:}}m}
  {\oldnumrange[#1]#2}
```

for the `\num` command and

```
\ExplSyntaxOn
\cs_gset_eq:NN \ifstrpresent \tl_if_in:nnTF
\ExplSyntaxOff
\NewCommandCopy\oldqty\qty
\NewCommandCopy\oldqtyrange\qtyrange
```

```
\RenewDocumentCommand\qty{O{}mm}{%
  \ifstrpresent{#2}{x}
   {\qtyproduct[#1]{#2}{#3}}
   {%
     \ifstrpresent{#2}{;}
        {\qtylist[#1]{#2}{#3}}
        {%
          \ifstrpresent{#2}{:}
            {\qtyrange[#1]{#2}{#3}}
            {\oldqty[#1]{#2}{#3}}%
        }%
    }%
}
\RenewDocumentCommand\qtyrange{O{}>{\SplitList{:}}mm}
  {\oldqtyrange[#1]#2{#3}}
```

for `\qty`.

11 Using (SI) units

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

At the same time, physical quantities with units are mathematical entities, and as such way that they are typeset is important. In mathematics, changes of type (such as using bold, italic, sans serif typeface and so on) convey information. This means that rules exist not only for the type of units to be used under the SI system, but also the way they should appear in print. Advice on best practice has been made available by the *National Institute of Standards and Technology* (NIST) [2].

As befits an agreed international standard, the full rules are detailed. It is not appropriate to reproduce these in totality here; instead, a useful summary of the key points is provided. The full details are available from the *Bureau International des Poids et Mesures* [1].

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

11.1 Units

There are seven base SI units, listed in Table 1.[4] The base units have been chosen such that all physical quantities can be expressed using an appropriative combination of these units, needing no others and with no redundancy.

All other units within the SI system are regarded as 'derived' from the seven base units. At the most basic, all other SI units can be expressed as combinations of the base units. However, many units (listed in Table 2) have a special name and symbol. Most

[4]Some base units need others defined first; there is therefore a required order of definition.

of these units are simple combinations of one or more base units (raised to powers as appropriate).

A series of SI prefixes for decimal multiples and sub-multiples are provided, and can be used as modifiers for any SI unit (either base or derived units) with the exception of the kilogram. The prefixes are listed in Table 4. No space should be used between a prefix and the unit, and only a single prefix should be used. Even the degree Celsius can be given a prefix, for example 1 m°C.

It is important to note that the kilogram is the only SI unit with a prefix as part of its name and symbol. Only single prefix may be used, and so in the case of the kilogram prefix names are used with the unit name 'gram' and the prefix symbols are used with the unit symbol g. For example $1 \times 10^{-6}\,\mathrm{kg} = 1 \times 10^{-3}\,\mathrm{g} = 1\,\mathrm{mg}$.

The application of SI units is meant to provide a single set of units which ensure consistency and clarity across all areas. However, other units are common is many areas, and are not without merit. The units provided by siunitx by default do not include any of these; only units which are part of the SI set or are accepted for use with SI units are defined. However, several other sets of units can be loaded as optional modules. The binary prefixes and units (Table 6) are the most obvious example. These are *not* part of the SI specifications, but the prefix names are derived from those in Table 4.

Other units are normally to be avoided where possible. SI units should, in the main, be preferred due to the advantages of clear definition and self-consistency this brings. However, there will probably always be a place for specialist or non-standard units. This is particularly true of units derived from basic physical constants.

There are also many areas where non-standard units are used so commonly that to do otherwise is difficult or impossible. For example, most synthetic chemists measure the pressure inside vacuum apparatus in mmHg, partly because the most common gauge for the task still uses a column of mercury metal. For these reasons, siunitx allows definition of such units.

11.2 Mathematical meaning

As explained earlier, a quantity combination is a single mathematical entity. This has implications for how both the number and the unit should be printed. Firstly, the two parts should not be separated: a quantity is a product of the number and the unit. With the exception of the symbols for plane angles (°, ′ and ″), the BIPM specifies either a space or half-height (centered) dot should be used [1].

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

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

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

```
Units should \textbf{not be bold: \qty{54}{\farad}}\\
\textbf{But perhaps in a running block,\\
it might look better:
  \qty[text-series-to-math]{54}{\farad}}
```
Units should **not be bold:** 54 F
But perhaps in a running block,
it might look better: 54 F

Symbols for units formed from other units by multiplication are indicated by means of either a half-height (that is, centered) dot or a (thin) space.

```
$\unit{\metre\second} = \text{metre second}$ \\
$\unit{\milli\second} = \text{millisecond}$ \\
\sisetup{inter-unit-product = \ensuremath { { } \cdot { } } }
$\unit{\metre\second} = \text{metre second}$ \\
$\unit{\milli\second} = \text{millisecond}$
```
$\mathrm{m\,s}$ = metre second
ms = millisecond
$\mathrm{m} \cdot \mathrm{s}$ = metre second
ms = millisecond

There are some circumstances under which it is common practice to omit any spaces. The classic example is kWh, where 'kWh' does not add any useful information. If using such a unit repeatedly, users of siunitx are advised to create a custom unit to ensure consistency. It is important to note that while this is common practice, it is *not* allowed by the BIPM [1].

Symbols for units formed from other units by division are indicated by means of a virgule (oblique stroke, slash, /), a horizontal line, or negative exponents.[5] However, to avoid ambiguity, the virgule must not be repeated on the same line unless parentheses are used. This is ensured when using named unit macros in siunitx, which will 'trap' repeated division and format it correctly. In complicated cases, negative exponents are to be preferred over other formats.

```
\unit{\joule\per\mole\per\kelvin} \\
\unit[per-mode = fraction]{\joule\per\mole\per\kelvin} \\
\unit[per-mode = symbol]{\joule\per\mole\per\kelvin}
```
$\mathrm{J\,mol^{-1}\,K^{-1}}$
$\frac{\mathrm{J}}{\mathrm{mol\,K}}$
J/(mol K)

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

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

1 m to 5 m long. `\qtyrange{1}{5}{\metre} long.`

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

Table 44: An example of table labelling.

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

11.3 Graphs and tables

In graphs and tables, repetition of the units following each entry or axis mark is confusing and repetitive. It is therefore best to place the unit in the label part of the information. Placing the unit in square brackets is common but mathematically poor.[6] Much better is to show division of all quantities by the unit, which leaves the entries as unitless ratios. This is illustrated in Table 44 and Figure 1.

```
\begin{table}
  \caption{An example of table labelling.%
    \label{tab:xmpl:unitless}}
  \sisetup{
    table-number-alignment = center,
    table-format = 1.4
  }
  \begin{tabular}{@{}cS@{}}
    \toprule
      Entry & {Length/\unit{\metre}} \\
    \midrule
      1 & 1.1234 \\
      2 & 1.1425 \\
      3 & 1.7578 \\
      4 & 1.9560 \\
    \bottomrule
  \end{tabular}
\end{table}

\begin{figure}
  \begin{tikzpicture}
    \begin{axis}[
      xlabel = $t/\unit{\second}$,
      xmax   = 6,
      xmin   = 0,
      ylabel = $d/\unit{\metre}$,
      ymin   = 0
    ]
      \addplot[smooth,mark=*]
        plot coordinates {
          (0,0)
          (1,5)
          (2,8)
          (3,9)
```

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

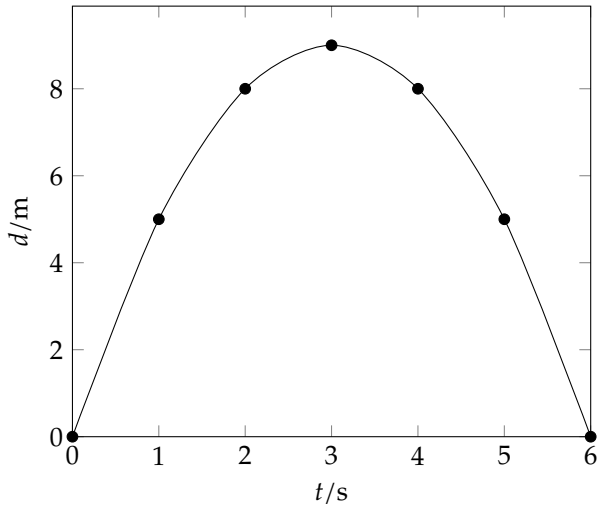

Figure 1: An example of graph labelling.

```
          (4,8)
          (5,5)
          (6,0)
        };
    \end{axis}
  \end{tikzpicture}
  \caption{An example of graph labelling.%
    \label{fig:xmpl:unitless}}
\end{figure}
```

In most cases, adding exponent values in the body of a table is less desirable than adding a fixed exponent to column headers. An example is shown in Table 45. The use of `\multicolumn` is needed here due to the '<'; without `\multicolumn`, the titles are followed by 'kg'!

```
\begin{table}
  \caption{Bad and good columns.%
    \label{tab:good}}
  \sisetup{table-number-alignment = center}
  \begin{tabular}{
    @{}
    c
    S[table-format = 1.3e1]
    @{\,\unit{\kilogram}}
    S[table-format = 2.2]
    @{}
  }
    \toprule
      Entry & \multicolumn{1}{c}{Mass} &
        {Mass/\qty[print-unity-mantissa = false]{e3}{\kilogram}} \\
    \midrule
      1 & 4.56e3  & 4.56 \\
      2 & 2.40e3  & 2.40 \\
      3 & 1.345e4 & 13.45 \\
```

Table 45: Bad and good columns.

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

```
    4 & 4.5e2   & 0.45 \\
  \bottomrule
 \end{tabular}
\end{table}
```

12 Installation

For most users, there will be no need to explicitly install siunitx: it is available from the package management system in current TEX Live and MiKTEX systems.

For manual installation, the package is available from CTAN. As well as the raw source files, CTAN hold the package as a pre-extracted zip file, `siunitx.tds.zip`. The later is most convenient for most users: simply unzip this in your local `texmf` directory.

The package requires expl3 support as provided in the l3kernel and bundle, which is required by the LATEX kernel. As such, you are very unlikely to need to update this manually.

13 Thanks

Many users have provided feedback, bug reports and ideas for new features for siunitx: thanks to all of them. Particular thanks to Stefan Pinnow, who has taken the lead role as beta tester for siunitx, finding incorrect output, bad documentation and the odd spelling mistake in the documentation. Thanks also to Enrico Gregorio for encouraging me to complete a fully expl3-compliant version of the package. Thanks also to Danie Els and Marcel Heldoorn for the SIstyle and SIunits packages, respectively, which provided the starting point for the development of siunitx.

14 Making suggestions and reporting bugs

Feedback on siunitx is always welcome, either to make suggestions or to report problems. When sending feedback, it is always useful if a small example file is included, showing the bug being reported or illustrating the desired output. It is helpful if a 'reference rendering' is included, showing what the output should look like. A typical example file might read

```
\listfiles
% Use the article class unless the problem is class-dependent
\documentclass{article}
\usepackage{siunitx}
```

```
% Other packages loaded as required
\begin{document}
Reference output: $1.23\,\mathrm{m}$

\textsf{siunitx} output: \qty{1.23}{\metre}
\end{document}
```

As illustrated, it is usually best to use the article class and to only load packages which are needed to show the issue. It is also useful to include a copy of the log file generate by LaTeX when reporting a bug (as the versions of packages can be important to solving the issue).

Feedback can be sent in a range of ways. The development code and issue tracker are hosted on GitHub: `https://github.com/josephwright/siunitx/`. Issues opened there are visible to other users and makes sure that they cannot be forgotten.

References

[1] *The International System of Units* (SI), `https://www.bipm.org/en/measurement-units/`.

[2] *International System of Units from* NIST, `http://physics.nist.gov/cuu/Units/index.html`.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z