The LUA-PHYSICAL library

Version 0.1

Thomas Jenni

May 1, 2019

Abstract

`lua-physical` is a pure Lua library which provides functions and object for doing computation with physical quantities. It has been written to simplify the creation problem sets. The package provides units of the SI and the imperial system. Furthermore an almost complete set of international currencies are supported, however without online exchange rates. In order to display the numbers with measurement uncertainties, the package is able to perform gaussian error propagation.

Contents

1 Introduction **3**

2 Loading **3**
2.1 Dependencies 4

3 Usage **4**
3.1 Unit conversion 4
3.1.1 Temperature Conversion 5
3.1.2 Uncertainty 6

4 Supported Units **8**
4.1 Base Units 8
4.2 Constants 10
4.3 Coherent derived units in the SI 11
4.4 Non-SI units accepted for use with the SI 12
4.5 Nominal Astronomical Units 12
4.6 Other Non-SI units 13
4.7 Imperial Units 15
4.8 U.S. customary units 17
4.9 International Currencies 19

5 Lua Documentation **20**
5.1 physical.Quantity 20
5.2 physical.Dimension 29
5.3 physical.Unit 29
5.4 physical.Number 30

Bibliography **33**

Index **34**

1 Introduction

The author of this package is a teacher at the high school *Kantonsschule Zug* in Switzerland. The main use of this package is to write physics problem sets. LuaLATEXdoes make it possible to integrate physical calculations directly. The package has been in use since 2016. Many bugs have been found and fixed. Nevertheless it still is possible, that some were not found yet. Therefore the author recommends not to use this package industry or science. If one does so, it's the responsability of the user to check results for plausability. If the user finds some bugs, they can be reported at github.com or directly to the author (`thomas.jenni (at) ksz.ch`).

2 Loading

This package is a pure Lua library. Therefore one has to require it explicitly by calling `require("physical")`. For printing results, the `siunitx` package can be used. It's recommended to define a macro like `\q` to convert the lua quantity object to a `siunitx` expression.
The following Latex preambel loads the `lua-physical` package and creates a macro `\q` for printing physical quantities.

Listing 1: basic preamble

```
\usepackage{lua-physical}
\usepackage{siunitx}

% configure siunitx
\sisetup{
  output-decimal-marker = {.},
  per-mode = symbol,
  separate-uncertainty = true,
  add-decimal-zero = true,
  exponent-product = \cdot,
  round-mode = off
}

% load the lua-physical package
\begin{luacode*}
  physical = require("physical")
  N = physical.Number
\end{luacode*}

% print a physical quantity
\newcommand{\q}[1]{%
  \directlua{tex.print(physical.Quantity.tosiunitx(#1,"scientific-
        notation=fixed,exponent-to-prefix=false"))}%
}
```

2.1 Dependencies

This is a standalone package. If a pretty print to LuaLaTeX is needed, the package `siunitx` should be installed.

3 Usage

Given the basic preamble, units can be used in lua code directly. By convention, all units have an underscore in front of them, i.e. meter is `_m`, second is `_s`. All available units are listed in chapter 4. The Result of the calculation can be printed to LuaLaTeX by using the macro `\q{}`.

Listing 2: The velocity of a car.

```
\begin{luacode}
  s = 10 * _m
  t = 2 * _s
  v = s/t
\end{luacode}

A car travels $\q{s}$ in $\q{t}$. calculate its velocity.
$$
  v=\frac{s}{t} = \frac{\q{s}}{\q{t}} = \q{v}
$$
```

> A car travels 10 m in 2 s. Calculate its velocity.
>
> $$v = \frac{s}{t} = \frac{10\,\mathrm{m}}{2\,\mathrm{s}} = 5\,\mathrm{m/s}$$

In the above listing 2, the variable `s` stands for displacement and has the unit meter `_m`. The variable `t` stands for time and is given in seconds `_s`. If mathematical operations are done on them, new physical quantities are created. In the problem above, the velocity `v` is calculated by dividing `s` by `t`. The instance `v` has the derived unit m/s. By using the macro `\q{}` all quantities can be printed to the LuaLaTeX code directly.

3.1 Unit conversion

Very often, the result of a calculation has to be converted to another unit. In the following listing 3 the task is to calculate the volume of a cuboid with lengths given in three different units. If the volume is calculated by multipling all three lengths, the unit of the result is cm mm m. If the unit cm^3 is preferred, it has to be converted explicitly. The conversion function is called `to()` and is available on all physical quantitiy instances. At first this looks a bit cumbersome. The reason of this behaviour is, that the software is not able to guess the unit of the result. In many cases, like in the example here, it's not clear what unit the result sould have. Therefore the user has always to give the target unit explicitly.

Listing 3: The volume of a cuboid.

```
\begin{luacode}
  a = 12 * _cm
  b = 150 * _mm
  c = 1.5 * _m

  V = a*b*c
\end{luacode}

Find the volume of a rectangular cuboid with lengths $\q{a}$,
$\q{b}$ and $\q{c}$.
$$
  V= a \cdot b \cdot c
  = \q{a} \cdot \q{b} \cdot \q{c}
  = \q{V}
  = \uuline{\q{V:to(_dm^3)}}
$$
```

Find the volume of a rectangular cuboid with lengths 12 cm, 150 mm and 1.5 m.

$$V = a \cdot b \cdot c = 12\,\mathrm{cm} \cdot 150\,\mathrm{mm} \cdot 1.5\,\mathrm{m} = 2700\,\mathrm{cm\,mm\,m} = \underline{\underline{27\,\mathrm{dm}^3}}$$

3.1.1 Temperature Conversion

Most of the units transform linearly to base units. Exceptions are the unit degree Celsius `_degC` and degree Fahrenheit `_degF`. These units are ambigous and can be interpreted as temperature differences or as an absolute temperatures. In the latter case, the conversion to base units is not a linear, but an affine transformation. This is because degree Celsius and degree Fahrenheit scales have their zero points at different temperatures compared to the unit Kelvin.
By default `_degC` and `_degF` units are standing for temperature differences. If one wants to have it converted absolutely, it has to be done adding / subtracting the quantities `_degC_0` and `_degF_0`.
In the following problem, listing 4 , the task is to convert temperatures given in the unit degree Celsius and degree Fahrenheit to Kelvin.

Listing 4: Temperature conversion.

```
\begin{luacode}
  theta_1 = 110 * _degC
  T_1 = ( theta_1 + _degC_0 ):to(_K)
  theta_1 = T_1:to(_degC) - _degC_0

  theta_2 = 212 * _degF
  T_2 = ( theta_2 + _degF_0 ):to(_K)
  theta_2 = T_2:to(_degF) - _degF_0
\end{luacode}

```

```
\begin{align*}
  \vartheta_1 &= \q{theta_1} \\
  T_1 &= \q{T_1} \\
  %
  \vartheta_2 &= \q{theta_2} \\
  T_2 &= \q{T_2} \\
\end{align*}
```

$$
\begin{aligned}
\vartheta_1 &= 110\,^{\circ}\mathrm{C} \\
T_1 &= 383.15\,\mathrm{K} \\
\vartheta_2 &= 212\,^{\circ}\mathrm{F} \\
T_2 &= 373.15\,\mathrm{K}
\end{aligned}
$$

3.1.2 Uncertainty

The package supports uncertainty propagation. To create a number with an uncertainty, an instance of `physical.Number` has to be created, see listing 5. It has to be remembered, that `N` is a alias for `physical.Number`. The first argument of the constructor `N(mean, uncertainty)` is the mean value and the second one the uncertainty of the measurement. If the proposed preamble 1 is used, the uncertainty is by default seperated from the mean value by a plus-minus sign.
For the uncertainty propagation the gaussian formula

$$\Delta f = \sqrt{\left(\frac{\partial f}{x_1} \cdot \Delta x_1\right)^2 + \cdots + \left(\frac{\partial f}{x_n} \cdot \Delta x_2\right)^2}$$

is used. This formula is a good estimation for the uncertainty Δf, if the quantities $x_1, \ldots, x_n$ the function f depends on, have no correlation. Further, the function f has to change linear, if quantities x_i are changed in the range of their uncertainties.

Listing 5: Uncertainty in area calculation.

```
\begin{luacode}
  a = N(2,0.1) * _m
  b = N(3,0.1) * _m

  A = (a*b):to(_m^2)
\end{luacode}

Calculate the area of a rectangle with lengths $\q{a}$ and $\q{b}$.
$$
  A = a \cdot b
  = \q{a} \cdot \q{b}
  = \uuline{\q{A}}
$$
```

Calculate the area of a rectangle with lengths $(2.00 \pm 0.10)\,\mathrm{m}$ and $(3.00 \pm 0.10)\,\mathrm{m}$.

$$A = a \cdot b = (2.00 \pm 0.10)\,\mathrm{m} \cdot (3.00 \pm 0.10)\,\mathrm{m} = \underline{\underline{(6.0 \pm 0.4)\,\mathrm{m}^2}}$$

Instead of printing always the uncertainties, one can use the uncertainty calculation to provide significant digits.
In the following problem, listing 6 , the task is to calculate the volume of an ideal gas. Given are pressure `p` in `_bar`, amount of substance `n` in `_mol` and temperature `T` in degree celsius `_degC`. In order to do the calculation, one has to convert `T`, which is given as an absolute temperature in degree celsius to the base unit Kelvin first. By setting `N.omitUncertainty = true`, all uncertainties are not printed.

Listing 6: Volume of an ideal gas.

```
\begin{luacode}
  N.omitUncertainty = true
  p = N(1.013,0.0001) * _bar
  n = N(1,0.01) * _mol
  T = N(30,0.1) * _degC

  V = ( n * _R * (T + _degC_0):to(_K) / p ):to(_L)
\end{luacode}

An ideal gas ($\q{n}$) has a pressure of $\q{p}$ and a temperature of $\q{T}$. Calculate the volume of the gas.
$$
  V=\frac{ \q{n} \cdot \q{_R} \cdot \q{(T + _degC_0):to(_K)} }{ \q{p} }
  = \q{V}
  = \uuline{\q{V}}
$$
```

An ideal gas ($1.0\,\mathrm{mol}$) has a pressure of $1.013\,\mathrm{bar}$ and a temperature of $30\,^\circ\mathrm{C}$. Calculate the volume of the gas.

$$V = \frac{1.0\,\mathrm{mol} \cdot 8.31\,\mathrm{J/(mol\,K)} \cdot 303\,\mathrm{K}}{1.013\,\mathrm{bar}} = \underline{\underline{25\,\mathrm{L}}}$$

4 Supported Units

All supported units are listed in this chapter. Subchapter 4.1 lists the seven base units of the International System of Units (SI). In subchapter 4.2 mathematical and physical constants are defined. The subchapter 4.3 contains all coherent derived units from the SI system and 4.4 those which are accepted to use with the SI.
The subchapter 4.5 lists nominal astronomical units, which are proposed by [3]. Subchapter 4.6 lists units, which are common but outside of the SI system. The subchapters 4.7 and 4.8 are dedicated to imperial and U.S. customary units. The last subchapter 4.9 containts international currencies.

4.1 Base Units

Quantity	Unit	Symbol	Dim.	Definition
number	–	`_1`	1	The dimensionless number one.
time	second	`_s`	T	The SI unit of time. It is defined by taking the fixed numerical value of the caesium frequency $\Delta\nu_{Cs}$, the unperturbed ground-state hyperfine transition frequency of the caesium 133 atom, to be 9 192 631 770 when expressed in the unit 1/s.
length	meter	`_m`	L	The SI unit of length. It is defined by taking the fixed numercial value of the speed of light in vacuum c to be 299 792 458 when expressed in the unit of 1 m/s.
mass	kilogram	`_kg`	M	The SI unit of mass. It is defined by taking the fixed numerical value of the Planck constant h to be $6.626\,070\,15 \cdot 10^{-34}$ when expressed in $\mathrm{m}^2\,\mathrm{kg/s}$.
electric current	ampere	`_A`	I	The SI unit of electric current. It is defined by taking the fixed numerical value of the elementary charge e to be $1.602\,176\,634 \cdot 10^{-19}$ when expressed in A s.

Quantity	Unit	Symbol	Dim.	Definition
thermodynamic temperature	kelvin	`_K`	Θ	The SI unit of the thermodynamic temperature. It is defineed by taking the fixed numerical value of the Boltzmann constant k_B to be $1.380\,649 \cdot 10^{-23}$ when expressed in $1\,\mathrm{kg}\,\mathrm{m}^2/(\mathrm{s}^2\,\mathrm{K})$
amount of substance	mole	`_mol`	N	The SI unit of amount of substance. One mole contains exactly $6.022\,140\,76 \cdot 10^{23}$ elementary entities. This number is the fixed numerical value of the Avogadro constant N_A when expressed in 1/mol.
luminous intensity	candela	`_cd`	J	The SI unit of luminous intensity in a given direction. It is defined by taking the fixed numerical value of the luminous efficacy of monochromatic radiation of frequency $5.4 \cdot 10^{14}\,\mathrm{Hz}$, K_{cd}, to be 683 when expressed in the unit $\mathrm{cd}\,\mathrm{sr}\,\mathrm{s}^3/(\mathrm{kg}\,\mathrm{m}^2)$.
information	bit	`_bit`	B	The smallest amount of information.
currency	euro	`_EUR`	C	The value of the currency Euro.

4.2 Constants

All physical constants are taken from the NIST webpage [1].

Name	Symbol	Definition
pi	`_Pi`	`3.14159265358979323846264338327950288419716 * _1`
eulersnumber	`_E`	`2.71828182845904523536028747135266249775724 * _1`
speedoflight	`_c`	`299792458 * _m/_s`
gravitationalconstant	`_Gc`	`N(6.67408e-11,3.1e-15) * _m^3/(_kg*_s^2)`
planckconstant	`_h_P`	`6.62607015e-34 * _J*_s`
reducedplanckconstant	`_h_Pbar`	`_h_P/(2*_Pi)`
elementarycharge	`_e`	`1.602176634e-19 * _C`
vacuumpermeability	`_u_0`	`4e-7*Pi * _N/_A^2`
vacuumpermitivity	`_e_0`	`1/(_u_0*_c^2)`
atomicmassunit	`_u`	`N(1.66053904e-27, 2e-35) * _kg`
electronmass	`_m_e`	`N(9.10938356e-31, 1.1e-38) * _kg`
protonmass	`_m_p`	`N(1.672621898e-27, 2.1e-35) * _kg`
neutronmass	`_m_n`	`N(1.674927471e-27, 2.1e-35) * _kg`
bohrmagneton	`_u_B`	`_e*_h_Pbar/(2*_m_e)`
nuclearmagneton	`_u_N`	`_e*_h_Pbar/(2*_m_p)`
electronmagneticmoment	`_u_e`	`N(-928.4764620e-26,5.7e-32) * _J/_T`
protonmagneticmoment	`_u_p`	`N(1.4106067873e-26,9.7e-35) * _J/_T`
neutronmagneticmoment	`_u_n`	`N(-0.96623650e-26,2.3e-26) * _J/_T`
finestructureconstant	`_alpha`	`_u_0*_e^2*_c/(2*_h_P)`
rydbergconstant	`_Ry`	`_alpha^2*_m_e*_c/(2*_h_P)`
avogadronumber	`_N_A`	`6.02214076e23/_mol`
boltzmannconstant	`_k_B`	`1.380649e-23 * _J/_K`
molargasconstant	`_R`	`N(8.3144598, 4.8e-6) * _J/(_K*_mol)`
stefanboltzmannconstant	`_sigma`	`_Pi^2*_k_B^4/(60*_h_Pbar^3*_c^2)`
standardgravity	`_g_0`	`9.80665 * _m/_s^2`

4.3 Coherent derived units in the SI

All units in this section are coherent derived units from the SI base units with special names, [2, 118].

Quantity	Unit	Symbol	Definition
Plane Angle[1]	radian	`_rad`	`_1`
Solid Angle[2]	steradian	`_sr`	`_rad^2`
Frequency	hertz	`_Hz`	`1/_s`
Force	newton	`_N`	`_kg*_m/_s^2`
Pressure	pascal	`_Pa`	`_N/_m^2`
Energy	joule	`_J`	`_N*_m`
Power	watt	`_W`	`_J/_s`
Electric Charge	coulomb	`_C`	`_A*_s`
Electric Potential	volt	`_V`	`_J/_C`
Electric Capacitance	farad	`_F`	`_C/_V`
Electric Resistance	ohm	`_Ohm`	`_V/_A`
Electric Conductance[3]	siemens	`_S`	`_A/_V`
Magnetic Flux	weber	`_Wb`	`_V*_s`
Magnetic Flux Density	tesla	`_T`	`_Wb/_m^2`
Inductance	henry	`_H`	`_Wb/_A`
Celsius Temperature[4]	celsius	`_degC`	`_K`
Luminous Flux	lumen	`_lm`	`_cd*_sr`
Illuminance	lux	`_lx`	`_lm/_m^2`
Activity	becquerel	`_Bq`	`1/_s`
Absorbed Dose	gray	`_Gy`	`_J/_kg`
Dose Equivalent	sievert	`_Sv`	`_J/_kg`
Catalytic Activity	katal	`_kat`	`_mol/_s`

[1] In the SI system, the quantity Plane Angle has the dimension of a number.
[2] In the SI system, the quantity Solid Angle has the dimension of a number.
[3] The unit `_PS` stands for peta siemens and is in conflict with the german version of the unit horsepower (Pferdestärke). Since the latter is more common than peta siemens, `_PS` is defined as the german version of horsepower.
[4] The unit `_degC` is by default interpreted as a temperature difference.

4.4 Non-SI units accepted for use with the SI

There are a few units with dimension 1.

Quantity	Unit	Symbol	Definition
Time	minute	`_min`	`60 * _s`
	hour	`_h`	`60 * _min`
	day	`_d`	`24 * _h`
Plane Angle	degree	`_deg`	`(_Pi/180) * _rad`
	arcminute	`_arcmin`	`_deg/60`
	arcsecond	`_arcsec`	`_arcmin/60`
Area	hectare	`_hectare`	`1e4 * _m^2`
Volume	liter	`_L`	`1e-3 * _m^3`
Mass	tonne	`_t`	`1e3 * _kg`

4.5 Nominal Astronomical Units

The nominal values of solar, terrestrial and jovial quantities are taken from IAU Resolution B3 [3].

Quantity	Unit	Symbol	Definition
Length	nomsolradius	`_R_S_nom`	`6.957e8 * _m`
Irradiance	nomsolirradiance	`_S_S_nom`	`1361 * _W/_m^2`
Radiant Flux	nomsolluminosity	`_L_S_nom`	`3.828e26 * _W`
Temperature	nomsolefftemperature	`_T_S_nom`	`5772 * _K`
Mass Parameter	nomsolmassparam	`_GM_S_nom`	`1.3271244e20 * _m^3*_s^-2`
Length	nomterreqradius	`_Re_E_nom`	`6.3781e6 * _m`
Length	nomterrpolradius	`_Rp_E_nom`	`6.3568e6 * _m`
Mass Parameter	nomterrmassparam	`_GM_E_nom`	`3.986004e14 * _m^3*_s^-2`
Length	nomjovianeqradius	`_Re_J_nom`	`7.1492e7 * _m`
Length	nomjovianpolradius	`_Rp_J_nom`	`6.6854e7 * _m`
Mass Parameter	nomjovianmassparam	`_GM_J_nom`	`1.2668653e17 * _m^3*_s^-2`

4.6 Other Non-SI units

The unit Bel is only available with prefix decibel, because `_B` is the unit byte.

Quantity	Unit	Symbol	Definition
Length	angstrom	`_angstrom`	`1e-10 * _m`
	fermi	`_fermi`	`1e-15 * _m`
	astronomicalunit	`_au`	`149597870700 * _m`
	lightsecond	`_ls`	`_c*_s`
	lightyear	`_ly`	`_c*_a`
	parsec	`_pc`	`(648000/_Pi) * _au`
Area	barn	`_barn`	`1e-28 * _m^2`
	are	`_are`	`1e2 * _m^2`
Volume	metricteaspoon	`_tsp`	`5e-3 * _L`
	metrictablespoon	`_Tbsp`	`3 * _tsp`
Time	svedberg	`_svedberg`	`1e-13 * _s`
	week	`_wk`	`7 * _d`
	year	`_a`	`365.25 * _d`
Plane Angle	gradian	`_gon`	`(Pi/200) * _rad`
	turn	`_tr`	`2*Pi * _rad`
Solid Angle	spat	`_sp`	`4*Pi * _sr`
Force	kilopond	`_kp`	`_kg*_g_0`
Pressure	bar	`_bar`	`1e5 * _Pa`
	standardatmosphere	`_atm`	`101325 * _Pa`
	technicalatmosphere	`_at`	`_kp/_cm^2`
	millimeterofmercury	`_mmHg`	`133.322387415 * _Pa`
	torr	`_Torr`	`(101325/760) * _Pa`

Quantity	Unit	Symbol	Definition
Energy	thermochemicalcalorie	`_cal`	`4.184 * _J`
	internationalcalorie	`_cal_IT`	`4.1868 * _J`
	gramoftnt	`_g_TNT`	`1e3 * _cal`
	tonoftnt	`_t_TNT`	`1e9 * _cal`
	electronvolt	`_eV`	`_e*_V`
	wattsecond	`_Ws`	`_W*_s`
	watthour	`_Wh`	`_W*_h`
Power	voltampere	`_VA`	`_V*_A`
Electric Charge	amperesecond	`_As`	`_A*_s`
	amperehour	`_Ah`	`_A*_h`
Information	nibble	`_nibble`	`4 * _bit`
	byte	`_B`	`8 * _bit`
Information Transfer Rate	bitpersecond	`_bps`	`_bit/_s`
Number	percent	`_percent`	`1e-2 * _1`
	permille	`_permille`	`1e-3 * _1`
	partspermillion	`_ppm`	`1e-6 * _1`
	partsperbillion	`_ppb`	`1e-9 * _1`
	partspertrillion	`_ppt`	`1e-12 * _1`
	partsperquadrillion	`_ppq`	`1e-15 * _1`
	decibel	`_dB`	`_1`
Power	metrichorsepower	`_PS`	`75 * _g_0*_kg*_m/_s`
Activity	curie	`_Ci`	`3.7e10 * _Bq`
Absorbed Dose	rad	`_Rad`	`1e-2 * _Gy`
Dose Equivalent	rem	`_rem`	`1e-2 * _Sv`
Viscosity	poiseuille	`_Pl`	`_Pa*_s`

4.7 Imperial Units

Quantity	Unit	Symbol	Definition
Length	inch	`_in`	`2.54e-2 * _m`
	thou	`_th`	`1e-3 * _in`
[1]	point	`_pt`	`_in/72`
	pica	`_pica`	`12 * _pt`
	hand	`_hh`	`4 * _in`
	foot	`_ft`	`12 * _in`
	yard	`_yd`	`3 * _ft`
	rod	`_rd`	`5.5 * _yd`
	chain	`_ch`	`4 * _rd`
	furlong	`_fur`	`10 * _ch`
	mile	`_mi`	`8 * _fur`
	league	`_lea`	`3*_mi`
	nauticalmile	`_nmi`	`1852 * _m`
	nauticalleague	`_nlea`	`3 * _nmi`
	cable	`_cbl`	`0.1 * _nmi`
	fathom	`_ftm`	`6 * _ft`
Velocity	knot	`_kn`	`_nmi/_h`
Area	acre	`_ac`	`10 * _ch^2`
Volume	gallon	`_gal`	`4.54609*_L`
	quart	`_qt`	`_gal/4`
	pint	`_pint`	`_qt/2`
	cup	`_cup`	`_pint/2`
	gill	`_gi`	`_pint/4`
	fluidounce	`_fl_oz`	`_gi/5`
	fluiddram	`_fl_dr`	`_fl_oz/8`

[1] The desktop publishing point or PostScript point is 1/72 of an international inch.

Quantity	Unit	Symbol	Definition
Mass	grain	`_gr`	`64.79891*_mg`
	pound	`_lb`	`7000*_gr`
	ounce	`_oz`	`_lb/16`
	dram	`_dr`	`_lb/256`
	stone	`_st`	`14*_lb`
	quarter	`_qtr`	`2*_st`
	hundredweight	`_cwt`	`4*_qtr`
	longton	`_ton`	`20*_cwt`

Table 1: Imperial units

4.8 U.S. customary units

In the U.S., the length units are bound to the meter differently than in the imperial system. The followin definitions are taken from `https://en.wikipedia.org/wiki/United_States_customary_units`.

Quantity	Unit	Symbol	Definition
Length	ussurveyinch	`_in_US`	`_m/39.37`
	ussurveyhand	`_hh_US`	`4*_in_US`
	ussurveyfoot	`_ft_US`	`3*_hh_US`
	ussurveylink	`_li_US`	`0.66*_ft_US`
	ussurveyyard	`_yd_US`	`3*_ft_US`
	ussurveyrod	`_rd_US`	`5.5*_yd_US`
	ussurveychain	`_ch_US`	`4*_rd_US`
	ussurveyfurlong	`_fur_US`	`10*_ch_US`
	ussurveymile	`_mi_US`	`8*_fur_US`
	ussurveyleague	`_lea_US`	`3*_mi_US`
	ussurveyfathom	`_ftm_US`	`72*_in_US`
	ussurveycable	`_cbl_US`	`120*_ftm_US`
Area	ussurveyacre	`_ac_US`	`_ch_US*_fur_US`
Volume	usgallon	`_gal_US`	`231*_in^3`
	usquart	`_qt_US`	`_gal_US/4`
	uspint	`_pint_US`	`_qt_US/2`
	uscup	`_cup_US`	`_pint_US/2`
	usgill	`_gi_US`	`_pint_US/4`
	usfluidounce	`_fl_oz_US`	`_gi_US/4`
	ustablespoon	`_Tbsp_US`	`_fl_oz_US/2`
	usteaspoon	`_tsp_US`	`_Tbsp_US/3`
	usfluiddram	`_fl_dr_US`	`_fl_oz_US/8`

Quantity	Unit	Symbol	Definition
Mass	usquarter	`_qtr_US`	`25*_lb`
Mass	ushundredweight	`_cwt_US`	`4*_qtr_US`
Mass	uston	`_ton_US`	`20*_cwt_US`

Table 2: U.S. customary units

4.9 International Currencies

Currency	Symbol	Definition
AfghanAfghani	`_AFN`	`0.012*_EUR`
AlbanianLek	`_ALL`	`0.008*_EUR`
ArmenianDram	`_AMD`	`0.0018*_EUR`
AngolanKwanza	`_AOA`	`0.0028*_EUR`
ArgentinePeso	`_ARS`	`0.021*_EUR`
AustralianDollar	`_AUD`	`0.63*_EUR`
AzerbaijaniManat	`_AZN`	`0.63*_EUR`
BosnianMark	`_BAM`	`0.51*_EUR`
BangladeshiTaka	`_BDT`	`0.011*_EUR`
BurundianFranc	`_BIF`	`0.00049*_EUR`
BolivianBoliviano	`_BOB`	`0.13*_EUR`
BrazilianReal	`_BRL`	`0.23*_EUR`
BotswanaPula	`_BWP`	`0.083*_EUR`
BelarusianRuble	`_BYN`	`0.42*_EUR`
CanadianDollar	`_CAD`	`0.66*_EUR`
USDollar	`_USD`	`0.89*_EUR`
JapaneseYen	`_JPY`	`0.008*_EUR`
PoundSterling	`_GBP`	`1.16*_EUR`
ChineseRenminbiYuan	`_CNY`	`0.13*_EUR`
SwedishKrona	`_SEK`	`0.094*_EUR`
NewZealandDollar	`_NZD`	`0.61*_EUR`

Table 3: International currency units based on exchange rates from 7.3.2019, 21:00 UTC.

5 Lua Documentation

In this chapter, the following shortcuts will be used.

```
local D = physical.Dimension
local U = physical.Unit
local N = physical.Number
local Q = physical.Quantity
```

The term `number` refers to a lua integer or a lua float number. By `string` a lua string is meant and by `bool` a lua boolean.

5.1 physical.Quantity

The quantity class is the main part of the library. Each physical Quantity and all units are represented by an instance of this class.

`Q.new(q=nil)`

Copy Constuctor

`q` : `Q` or `number`, optional
: Optional argument is either `Q`, a `number` or `nil`.

`return` : `Q`
: The created `Q` instance

As an argument it takes `Q`, `number` or `nil`. If `Q` is given, a copy of it is made and returned. If a `number` is given, the function creates a dimeensionless quantity with that value. In the case `nil` is given, the quantity `_1` is returned.

Example

```
myOne = Q()
myNumber = Q(42)
myLength = Q(73*_m)
```

`Q.defineBase(symbol,name,dimension)`

This function is used to declare base quantities from which all other quantities are derived from.

`symbol` : `string`
: The symbol of the base quantity.

`name` : `string`
: The name of the base quantity.

`dimension` : `D`
: An instance of the `D` class, which represents the dimension of the quantity.

`return` : `Q`
  The created `Q` instance.

The function creates a global variable of the created base quantity. The name consist of an underscore concatenated with the `symbol` argument, i.e. the symbol `m` becomes the global variable `_m`.

The `name` is used for example in the siunitx conversion function, e.g `meter` will be converted to `\meter`.

Each quantity has a dimension associated with it. The argument `dimension` allows any dimension to be associated to base quantities.

Example

```
Q.defineBase("m", "meter", L)
Q.defineBase("kg", "kilogram", M)
```

`Quantity.define(symbol, name, q)`

Creates a new derived quantity from an expression of other quantities. Affine quantities like the absolute temperature in celsius are not supported.

`symbol` : `string`
  Symbol of the base quantity

`name` : `string`
  The Name of the derived quantity.

`q` : `physical.Quantity`
  The definition of the derived quantity.

`return` : `Quantity`
  The created quantity.

The function creates a global variable of the created base quantity. The name consist of an underscore concatenated with the `symbol` argument, i.e. the symbol `N` becomes the global variable `_N`.

The `name` is used for example in the siunitx conversion function, e.g `newton` will be converted to `\newton`.

Example

```
Q.define("L", "liter", _dm^3)
Q.define("Pa", "pascal", _N/_m^2)
Q.define("C", "coulomb", _A*_s)

Q.define("degC", "celsius", _K)
```

`Quantity.defineAlias(symbol, q)`

Creates an alias of a quantity.

`symbol` : `string`
: The name of the alias.

`q` : `Quantity`
: The aliased quantity.

```
Q.defineAlias("degC_0", 273.15*_degC)
```

`Quantity.definePrefix(symbol,name,factor)`

Defines a new prefix.

`symbol` : `string`
: Symbol of the base quantity

`name` : `string`
: Name of the base quantity

`factor` : `number`
: The factor which corresponds to the prefix

```
Q.definePrefix("c", "centi", 1e-2)
Q.definePrefix("a", "atto", 1e-18)
```

`Quantity.addPrefix(prefixes, units)`

Create several units with prefixes from a given unit.

`prefixes` : `string`
: A list of unit symbols.

`units` : `Quantity`
: A list of quantities.

```
Q.addPrefix({"n","u","m","k","M","G"},{_m,_s,_A})
```

Quantity.to(self,q)

Converts the quantity self to the unit of the quantity `q`.

`self` : `Quantity`
A quantity.

`q` : `Quantity` The quantity to wich `self` will be converted to.

```
s = 1.9 * _km
print( s:to(_m) )
1900 * _m

T = 10 * _degC
print( T:to(_K) )
10 * _K
print( T:to(_K) )
10 * _K
```

Quantity.tosiunitx(self,param,mode)

Converts the quantity into a siunitx string.

`self` : `Quantity`

`param` : `string`

`mode` : `Number`, 0:\SI, 1:\num, 2:\si

```
s = 1.9 * _km

print( s:tosiunitx() )
\SI{1.9}{\kilo\meter}

print( s:tosiunitx(nil,1) )
\num{1.9}

print( s:tosiunitx(nil,2) )
\si{\kilo\meter}
```

Quantity.isclose(self,q,r)

Checks if this quantity is close to another one. The argument `r` is the maximal relative deviation.

`self` : `Quantity`

`q` : `Quantity,Number`

`r` : `Number`

```
s_1 = 1.9 * _m
s_2 = 2.0 * _m
print( s_1:isclose(s_2,0.1) )
true
print( s_1:isclose(s_2,0.01) )
false
```

`Quantity.min(q1, q2, ...)`

Returns the smallest quantity of several given ones. The function returns `q1` if the Quantities are equal.

`q1` : `Quantity,Number`, first argument

`q2` : `Quantity,Number`, second argument

```
s_1 = 15 * _m
s_2 = 5 * _m
print(s_1:min(s_2))
5 * _m
```

`Quantity.max(q1, q2, ...)`

Returns the biggest quantity of several given ones. The function returns `q1` if the Quantities are equal.

`q1` : `Quantity,Number`, first argument

`q2` : `Quantity,Number`, second argument

```
s_1 = 15 * _m
s_2 = 5 * _m
print(s_1:max(s_2))
15 * _m
```

`Quantity.abs(q)`

Returns the absolute value of the given quantity `q`.

`q` : `Quantity,Number`, argument

```
U = -5 * _V
print(U)
-5 * _V
print(U:abs())
5 * _V
```

`Quantity.sqrt(q)`

Returns the square root of the given quantity.

`q` : `Quantity`,`Number` argument

```
A = 25 * _m^2
s = sqrt(A)
print(s)

```

`Quantity.log(q, base)`

Returns the logarithm of the given quantitiy. If no base is given, the natural logarithm is calculated.

`q` : `Quantity`,`Number` dimensionless argument

`base` : `Quantity`,`Number` dimensionless argument

```
I = 1 * _W/_m^2
I_0 = 1e-12 * _W/_m^2
print(10 * (I/I_0):log(10) * _dB )
120 * _dB
```

`Quantity.exp(q)`

Returns the value of the exponential function of the given quantitiy.

`q` : `Quantity`,`Number` dimensionless argument

```
x = 2 * _1
print( x:exp() )
7.3890560989307
```

`Quantity.sin(q)`

Returns the value of the sinus function of the given quantitiy.

`q` : `Quantity`,`Number` dimensionless argument

```
alpha = 30 * _deg
print( alpha:sin() )
0.5
```

Quantity.cos(q)

Returns the value of the cosinus function of the given quantity. The quantity has to be dimensionless.

q : Quantity,Number dimensionless argument

```
alpha = 60 * _deg
print( alpha:cos() )
0.5
```

Quantity.tan(q)

Returns the value of the tangent function of the given quantity. The quantity has to be dimensionless.

q : Quantity,Number dimensionless argument

```
alpha = 45 * _deg
print( alpha:tan() )
1
```

Quantity.asin(q)

Returns the value of the arcus sinus function of the given quantity. The quantity has to be dimensionless.

q : Quantity,Number dimensionless argument

```
x = 0.5 * _1
print( x:asin():to(_deg) )
30 * _deg
```

Quantity.acos(q)

Returns the value of the arcus cosinus function of the given quantity. The quantity has to be dimensionless.

q : Quantity,Number dimensionless argument

```
x = 0.5 * _1
print( x:acos():to(_deg) )
60 * _deg
```

`Quantity.atan(q)`

Returns the value of the arcus tangent function of the given quantity. The quantity has to be dimensionless.

`q` : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:atan():to(_deg) )
45 * _deg
```

`Quantity.sinh(q)`

Returns the value of the hyperbolic sine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\sinh(x) = 0.5 \cdot e^x - 0.5/e^x \quad .$$

`q` : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:sinh() )
1.1752011936438
```

`Quantity.cosh(q)`

Returns the value of the hyperbolic cosine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\cosh(x) = 0.5 \cdot e^x + 0.5/e^x \quad .$$

`q` : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:cosh() )
1.5430806348152
```

Quantity.tanh(q)

Returns the value of the hyperbolic tangent function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad .$$

q : Quantity,Number dimensionless argument

```
x = 1 * _1
print( x:tanh() )
0.76159415595576
```

Quantity.asinh(q)

Returns the value of the inverse hyperbolic sine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\operatorname{asinh}(x) = \ln\left(x + \sqrt{x^2 + 1}\right) \quad .$$

q : Quantity,Number dimensionless argument

```
x = 1 * _1
print( x:asinh() )
0.88137358701954
```

Quantity.acosh(q)

Returns the value of the inverse hyperbolic cosine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\operatorname{acosh}(x) = \ln\left(x + \sqrt{x^2 - 1}\right) \quad , x > 1 \quad .$$

q : Quantity,Number dimensionless argument bigger than or equal to one.

```
x = 2 * _1
print( x:acosh() )
1.3169578969248
```

`Quantity.atanh(q)`

Returns the value of the inverse hyperbolic cosine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\mathrm{atanh}(x) = \ln\left(\frac{1+x}{1-x}\right) \quad , -1 < x < 1 \quad .$$

`q` : `Quantity,Number` dimensionless argument with magnitude smaller than one.

```
x = 0.5 * _1
print( x:atanh() )
0.54930614433405
```

5.2 physical.Dimension

All physical quantities do have a physical dimension. For example the quantity *Area* has the dimension L^2 (lenght to the power of two). In the SI-System there are seven base dimensions, from which all other dimensions are derived. Each dimension is represented by an n-tuple, where n is the number of base dimensions. Each physical quantity has an associated dimension object. It is used to check equality and if addition or substraction is allowed.

`Dimension.new(q=nil)`

Constructor of the `Dimension` class.

`q` : `Dimension` or `string`, optional
The name or symbol of the dimension. If `q` is a dimension, a copy of it is made. If no argument ist given, a dimension *zero* is created.

`return` : `Dimension`
The created `Quantity` object

Example

```
V_1 = D("Velocity")
L = D("L")
V_2 = D(L/T)
```

5.3 physical.Unit

The task of this class is keeping track of the unit term. The unit term is a fraction of units. The units in the enumerator and denominator can have an exponent.

`Unit.new(u=nil)`

Copy Constructor. It copies a given unit object. If nothing is given, an empty unit is created.

`u` : `Unit`
The unit object which will be copied.

`return` : `Unit`
The created `Unit` object

`Unit.new(symbol, name, prefixsymbol=nil, prefixname=nil)`

Constructor. A new `Unit` object with symbol is created. The prefixsymbol and prefixname are optional.

`symbol` : `String`
The symbol of the unit.

`name` : `String`
The name of the unit.

`prefixsymbol` : `String`
The optional symbol of the prefix.

`prefixname` : `String`
The optional name of the prefix.

`return` : `Unit`
The created `Unit` object

`Unit.tosiunitx(self)`

The unit term will be compiled into a string, which the LaTeX package siunitx can understand.

`return` : `String`
The siunitx representation of the unit term.

5.4 physical.Number

It does arithmetics with gaussian error propagation. A number instance has a mean value called `x` and an uncertainty called `dx`.

`Number.new(n=nil)`

This is the copy Constructor. It copies a given number object. If `n` is `nil`, an instance representing number zero with uncertainty zero is created.

`n` : `Number`
: The number object to be copied.

`return` : `Number`
: The created `Number` instance.

`Number.new(x, dx)`

This constructor, creates a new instance with mean value `x` and uncertainty `dx`.

`x` : `number`
: mean value

`dx` : `number`
: uncertainty value

`return` : `Number`
: The created `Number` instance.

Example

```
n = N(12,0.1)
print(n)
```

`Number.new(str)`

This constructor creates a new instance from a string. It can parse strings of the form `3.4`, `3.4e-3`, `5.4e-3 +/- 2.4e-6`, `5.45(7)e-23`.

Parameters / Return

`str` : `string`
: The number as a string.

`return` : `Number`
: The created `Number` object

Example

```
n_1 = N("12.3e-6")
print(n_1)

n_2 = N("12 +/- 0.1")
print(n_2)

n_3 = N("12.0(1)")
print(n_3)
```

`Number.mean(n)`

Returns the mean value

Parameters / Return

`return` : `number`
The mean value

`Number.uncertainty(n)`

Returns the uncertainty value

Parameters / Return

`return` : `number`
The uncertainty value

`Number.abs(n)`

Returns the absolute value of the number.

Parameters / Return

`return` : `number`
The absolute value

`Number.sqrt(n)`

Returns the square root of the number.

Parameters / Return

`return` : `number`
The square root

References

[1] Webpage https://physics.nist.gov/cuu/index.html, August 2019.

[2] Bureau International des Poids et Mesures. The international system of units (si), 2006.

[3] Prša et al. Nominal values for selected solar and planetary quantities: Iau 2015 resolution b3. The Astronomical Journal, 152:41, August 2016.

Index

acre `_ac`, 15
amperehour `_Ah`, 14
amperesecond `_As`, 14
angstrom `_angstrom`, 13
arcminute `_arcmin`, 12
arcsecond `_arcsec`, 12
are `_are`, 13
astronomicalunit `_au`, 13
atomicmassunit `_u`, 10
avogadronumber `_N_A`, 10

bar `_bar`, 13
barn `_barn`, 13
becquerel `_Bq`, 11
bitpersecond `_bps`, 14
bohrmagneton `_u_B`, 10
boltzmannconstant `_k_B`, 10
byte `_B`, 14

cable `_cbl`, 15
celsius `_degC`, 11
chain `_ch`, 15
coulomb `_C`, 11
cup `_cup`, 15
curie `_Ci`, 14

day `_d`, 12
decibel `_dB`, 14
degree `_deg`, 12
dram `_dr`, 16

electronmagneticmoment `_u_e`, 10
electronmass `_m_e`, 10
electronvolt `_eV`, 14
elementarycharge `_e`, 10
eulersnumber `_E`, 10

farad `_F`, 11
fathom `_ftm`, 15
fermi `_fermi`, 13
finestructureconstant `_alpha`, 10
fluiddram `_fl_dr`, 15
fluidounce `_fl_oz`, 15
foot `_ft`, 15
furlong `_fur`, 15

gallon `_gal`, 15
gill `_gi`, 15
gradian `_gon`, 13
grain `_gr`, 16
gramoftnt `_g_TNT`, 14
gravitationalconstant `_Gc`, 10
gray `_Gy`, 11

hand `_hh`, 15
hectare `_hectare`, 12
henry `_H`, 11
hertz `_Hz`, 11
hour `_h`, 12
hundredweight `_cwt`, 16

inch `_in`, 15
internationalcalorie `_cal_IT`, 14

joule `_J`, 11

katal `_kat`, 11
kilopond `_kp`, 13
knot `_kn`, 15

league `_lea`, 15
lightsecond `_ls`, 13
lightyear `_ly`, 13
liter `_L`, 12
longton `_ton`, 16
lumen `_lm`, 11
lux `_lx`, 11

metrichorsepower `_PS`, 14
metrictablespoon `_Tbsp`, 13
metricteaspoon `_tsp`, 13
mile `_mi`, 15
millimeterofmercury `_mmHg`, 13
minute `_min`, 12
molargasconstant `_R`, 10

nauticalleague `_nlea`, 15

nauticalmile _nmi, 15
neutronmagneticmoment _u_n, 10
neutronmass _m_n, 10
newton _N, 11
nibble _nibble, 14
nomjovianeqradius _Re_J_nom, 12
nomjovianmassparam _GM_J_nom, 12
nomjovianpolradius _Rp_J_nom, 12
nomsolefftemperature _T_S_nom, 12
nomsolirradiance _S_S_nom, 12
nomsolluminosity _L_S_nom, 12
nomsolmassparam _GM_S_nom, 12
nomsolradius _R_S_nom, 12
nomterreqradius _Re_E_nom, 12
nomterrmassparam _GM_E_nom, 12
nomterrpolradius _Rp_E_nom, 12
nuclearmagneton _u_N, 10

ohm _Ohm, 11
ounce _oz, 16

parsec _pc, 13
partsperbillion _ppb, 14
partspermillion _ppm, 14
partsperquadrillion _ppq, 14
partspertrillion _ppt, 14
pascal _Pa, 11
percent _percent, 14
permille _permille, 14
pi _Pi, 10
pica _pica, 15
pint _pint, 15
planckconstant _h_P, 10
point _pt, 15
poiseuille _Pl, 14
pound _lb, 16
protonmagneticmoment _u_p, 10
protonmass _m_p, 10

quart _qt, 15
quarter _qtr, 16

rad _Rad, 14
radian _rad, 11
reducedplanckconstant _h_Pbar, 10
rem _rem, 14
rod _rd, 15
rydbergconstant _Ry, 10

siemens _S, 11
sievert _Sv, 11
spat _sp, 13
speedoflight _c, 10
standardatmosphere _atm, 13
standardgravity _g_0, 10
stefanboltzmannconstant _sigma, 10
steradian _sr, 11
stone _st, 16
svedberg _svedberg, 13

technicalatmosphere _at, 13
tesla _T, 11
thermochemicalcalorie _cal, 14
thou _th, 15
tonne _t, 12
tonoftnt _t_TNT, 14
torr _Torr, 13
turn _tr, 13

uscup _cup_US, 17
usfluiddram _fl_dr_US, 17
usfluidounce _fl_oz_US, 17
usgallon _gal_US, 17
usgill _gi_US, 17
ushundredweight _cwt_US, 18
uspint _pint_US, 17
usquart _qt_US, 17
usquarter _qtr_US, 18
ussurveyacre _ac_US, 17
ussurveycable _cbl_US, 17
ussurveychain _ch_US, 17
ussurveyfathom _ftm_US, 17
ussurveyfoot _ft_US, 17
ussurveyfurlong _fur_US, 17
ussurveyhand _hh_US, 17
ussurveyinch _in_US, 17
ussurveyleague _lea_US, 17
ussurveylink _li_US, 17
ussurveymile _mi_US, 17
ussurveyrod _rd_US, 17
ussurveyyard _yd_US, 17

ustablespoon _Tbsp_US, 17
usteaspoon _tsp_US, 17
uston _ton_US, 18

vacuumpermeability _u_0, 10
vacuumpermitivity _e_0, 10
volt _V, 11
voltampere _VA, 14

watt _W, 11
watthour _Wh, 14
wattsecond _Ws, 14
weber _Wb, 11
week _wk, 13

yard _yd, 15
year _a, 13