

# The LUA-PHYSICAL library

Version 0.1

Thomas Jenni

February 19, 2019

## Abstract

`lua-physical` is a pure Lua library which provides functions and object for doing computation with physical quantities. This package provides a standard set of units of the SI and the imperial system. It is possible to give a number a measurement uncertainty.

is also integrated and is calculated by gaussian error propagation. The package includes some

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Basic usage</b>	<b>2</b>
<b>3</b>	<b>Supported Units</b>	<b>4</b>
<b>4</b>	<b>Lua Documentation</b>	<b>9</b>
4.1	physical.Dimension . . . . .	9
4.2	physical.Quantity . . . . .	9

## 1 Introduction

The author of this package is a teacher at the *Kantonsschule Zug, Switzerland*, a high-school. The main use of this package is to write physics problem sets and integrate the calculation directly into the luatex-file. The package is now in use for more than two years and a lot of bugs have been found and crushed. Nevertheless it could be possible that some bugs are still there, living uncovered. Therefore I recommend not to use this library productively in industry or science. If one does so, it's the responsibility of the user to check results for plausability. If the user finds some bugs, please report them on [github.com](https://github.com) or directly to the author.

E-Mail: `thomas.jenni(at)ksz.ch`

## 2 Basic usage

Since this package is pure lua library one has to require it explicitly by calling `require("physical")`. For printing results the `siunitx` package is used. It's recommended to define a shortcut like `\q` or `\qty` to convert the lua quantity object to a siunitx expression. An example preamble is shown in the following.

```
1  \usepackage{siunitx}
2
3  % configure siunitx
4  \sisetup{
5    output-decimal-marker = {.,},
6    per-mode = symbol,
7    separate-uncertainty = false,
8    add-decimal-zero = true,
9    exponent-product = \cdot,
10   round-mode = off
11 }
12
13 % load lua-physical
14 \begin{luacode*}
15   physical = require("physical")
16 \end{luacode*}
17
18 % shortcut for printing physical quantities
19 \newcommand{\q}[1]{%
20   \directlua{tex.print(physical.Quantity.tosiunitx(#1,"scientific-
21     notation=fixed,exponent-to-prefix=false"))}%
22 }
```

Listing 1: basic preamble

Given the preamble one can use now units in lua code and insert results in the latex code.

```

1  \begin{luacode}
2    s = 10 * _m
3    t = 2 * _s
4    v = s/t
5  \end{luacode}
6
7  A car travels  $\text{\q{s}}$  in  $\text{\q{t}}$ . calculate its velocity.
8  $$
9    v=\frac{s}{t} = \frac{\text{\q{s}}}{\text{\q{t}}} = \text{\q{v}} = \text{\q{v:to(_km/_h)}}
10  $$

```

Listing 2: basic usage

A car travels 10 m in 2 s. Calculate its velocity.

$$v = \frac{s}{t} = \frac{10 \text{ m}}{2 \text{ s}} = 5 \text{ m/s} = 18 \text{ km/h}$$

### 3 Supported Units

There are a few units with dimension 1. The unit Bel is only available with prefix decibel, because `_B` is the unit byte.

Unit	Symbol	Definition
number	<code>_1</code>	The number one.
percent %	<code>_percent</code>	$1e-2*_1$
permille ‰	<code>_permille</code>	$1e-3*_1$
parts-per-million	<code>_ppm</code>	$1e-6*_1$
parts-per-billion	<code>_ppb</code>	$1e-9*_1$
parts-per-trillion	<code>_ppt</code>	$1e-12*_1$
parts-per-quadrillion	<code>_ppq</code>	$1e-15*_1$
decibel	<code>_dB</code>	<code>_1</code>

Table 1: Dimensionless units

Quantity	Unit	Symbol	Dim.	Definition
length	meter	<b>_m</b>	L	The distance light travels in vacuum during 1/299 792 458 second.
mass	kilogram	<b>_kg</b>	M	The mass of the international prototype of the kilogram.
time	second	<b>_s</b>	T	Is 9 192 631 770 times the period of the radiation from the transition between the two hyperfine levels of the ground state of caesium-133.
electric current	ampere	<b>_A</b>	I	The constant current which, if maintained in two straight parallel conductors of infinite length, of negligible circular cross-section, and placed 1 m apart in vacuum, would produce between these conductors a force equal to $2 \cdot 10^{-7}$ N/m.
thermodynamic temperature	kelvin	<b>_K</b>	$\Theta$	Is the fraction 1/273.16 of the thermodynamic temperature of the triple point of water.
amount of substance	mole	<b>_mol</b>	N	Amount of substance that contains as many particles as there are atoms in 0.012 kg of carbon-12.
luminous intensity	candela	<b>_cd</b>	J	the luminous intensity, in a given direction, of a source that emits monochromatic radiation of frequency $540 \cdot 10^{12}$ Hz and has a radiant intensity in that direction of (1/683) W/sr

Table 2: Base units of the International System of Units (SI)

Quantity	Unit	Symbol	Dimension	Definition
plane angle	radian	<code>_rad</code>	1	<code>_1</code>
solid angle	steradian	<code>_sr</code>	1	<code>_rad^2</code>
frequency	hertz	<code>_Hz</code>	$T^{-1}$	<code>1/_s</code>
force	newton	<code>_N</code>	$M L T^{-2}$	<code>_kg*_m/_s^2</code>
pressure	pascal	<code>_Pa</code>	$M L^{-1} T^{-2}$	<code>_N/_m^2</code>
energy	joule	<code>_J</code>	$M L^2 T^{-2}$	<code>_N*_m</code>
power	watt	<code>_W</code>	$M L^2 T^{-3}$	<code>_J/_s</code>
electric charge	coulomb	<code>_C</code>	$T I$	<code>_A*_s</code>
electric potential difference	volt	<code>_V</code>	$M L^2 T^{-3} I^{-1}$	<code>_J/_C</code>
capacitance	farad	<code>_F</code>	$L^{-2} M^{-1} T^4 I^2$	<code>_C/_V</code>
electric resistance	ohm	<code>_Ohm</code>	$L^2 M T^{-3} I^{-2}$	<code>_V/_A</code>
electric conductance	siemens	<code>_S</code>	$L^{-2} M^{-1} T^3 I^2$	<code>_A/_V</code>
magnetic flux	weber	<code>_Wb</code>	$L^2 M T^{-2} I^{-1}$	<code>_V*_s</code>
magnetic flux density	tesla	<code>_T</code>	$M T^{-2} I^{-1}$	<code>_V*_s</code>
inductance	henry	<code>_H</code>	$L^2 M T^{-2} I^{-2}$	<code>_Wb/_A</code>
Celsius temperature	degree Celsius	<code>_degC</code>	$\Theta$	<code>_K</code>
luminous flux	lumen	<code>_lm</code>	J	<code>_cd*_sr</code>
illuminance	lux	<code>_lux</code>	$L^{-2} J$	<code>_lm/_m^2</code>
activity	becquerel	<code>_Bq</code>	$T^{-1}$	<code>1/_s</code>
absorbed dose	gray	<code>_Gy</code>	$L^2 T^{-2}$	<code>_J/_kg</code>
dose equivalent	sievert	<code>_Sv</code>	$L^2 T^{-2}$	<code>_J/_kg</code>
catalytic activity	katal	<code>_kat</code>	$T^{-1} N$	<code>_mol/_s</code>

Table 3: Derived units of the International System of Units (SI)

Quantity	Unit	Symbol	Dim.	Definition
plane angle	degree	<code>_deg</code>	1	$(\text{Pi}/180)*\_rad$
	arc minute	<code>_arcmin</code>	1	<code>_deg/60</code>
	arc second	<code>_arcsec</code>	1	<code>_arcmin/60</code>
	gradian	<code>_gon</code>	1	$(\text{Pi}/200)*\_rad$
	turn	<code>_tr</code>	1	$2*\text{Pi}*\_rad$
solid angle	spat	<code>_sp</code>	1	$4*\text{Pi}*\_sr$
length	astronomical unit	<code>_au</code>	L	$149597870700*_m$
	lightyear	<code>_ly</code>	L	<code>_c*_a</code>
	parsec	<code>_pc</code>	L	$(648000/\text{Pi})*\_au$
	angstrom	<code>_angstrom</code>	L	$1e-10*_m$
	fermi	<code>_fermi</code>	L	$1e-15*_m$
area	are	<code>_ar</code>	$L^2$	$1e2*_m^2$
	hectare	<code>_hectare</code>	$L^2$	$1e4*_m^2$
	barn	<code>_barn</code>	$L^2$	$1e-28*_m^2$
volume	liter	<code>_L</code>	$L^3$	$0.001*_m^3$
	metric teaspoon	<code>_tsp</code>	$L^3$	$0.005*_L$
	metric tablespoon	<code>_Tbsp</code>	$L^3$	$3*_tsp$
time	minute	<code>_min</code>	T	<code>_60*_s</code>
	hour	<code>_h</code>	T	<code>_60*_min</code>
	day	<code>_d</code>	T	<code>_24*_h</code>
	week	<code>_wk</code>	T	<code>_7*_d</code>
	year	<code>_a</code>	T	$365.25*_d$
	svedberg	<code>_svedberg</code>	T	$1e-13*_s$
mass	tonne	<code>_t</code>	M	$1000*_kg$

Table 4: Units outside of the International System of Units (SI)

Quantity	Unit	Symbol	Dim.	Definition
length	inch	<code>_in</code>	L	$0.0254*_m$
	thou	<code>_th</code>	L	$0.001*_in$
	pica	<code>_pica</code>	L	$_in/6$
	point	<code>_pt</code>	L	$_in/72$
	hand	<code>_hh</code>	L	$4*_in$
	foot	<code>_ft</code>	L	$12*_in$
	yard	<code>_yd</code>	L	$3*_ft$
	rod	<code>_rd</code>	L	$5.5*_yd$
	chain	<code>_ch</code>	L	$4*_rd$
	furlong	<code>_fur</code>	L	$10*_ch$
	mile	<code>_mi</code>	L	$8*_fur$
	league	<code>_lea</code>	L	$3*_mi$

Table 5: Imperial units



## 4 Lua Documentation

In the following chapter, these shortcuts will be used.

```
1 local D = physical.Dimension
2 local Q = physical.Quantity
```

### 4.1 physical.Dimension

All physical quantities do have a physical dimension. For example the quantity *Area* has the dimension  $L^2$  (length 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 if two quantities can be added or subtracted and if they are equal.

#### **Dimension.new(q=nil)**

Constructor of the **Dimension** class.

##### **Parameters**

**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 is given, a dimension *zero* is created.

**return** : **Dimension**

The created **Quantity** object

##### **Notes**

—

##### **Examples**

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

### 4.2 physical.Quantity

#### **Quantity.new(q=nil)**

Constructor of the **Quantity** class.

##### **Parameters**

**q** : **Quantity** or **number**, optional  
Optional argument for copying a **Quantity** or to create a **number** quantity

**return** : **Quantity**  
The created **Quantity** object

### Notes

It takes an optional **quantity** or **number** for the argument **q**. If the argument **q** is given, the new quantity is a copy of it. If no argument is given, a quantity **\_1** is created.

### Examples

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

## **Quantity.defineBase(symbol,name,dimension)**

A unit system has some special units, called base units. From those base units all other units are derived. This function is used to declare the base units. Since in this library units are the same thing as quantities, one has to define base quantities (units).

### Parameters

**symbol** : **string**  
symbol of the base quantity

**name** : **string**  
name of the base quantity

**dimension** : **Dimension**  
object which represents the base **Dimension** of the base quantity

**return** : The created **physical.Quantity** object

### Notes

The function creates a global variable, an underscore concatenated with the **symbol** argument, e. g. **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. The argument **dimension** allows any dimension to be associated to base quantities.

### Examples

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

### **Quantity.define(symbol, name, q, tobase=nil, frombase=nil)**

Creates a new derived unit from an expression of other units. For affine quantities like the temperature in celcius, one can give conversion functions to and from base units.

#### **Parameters**

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

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

**q** : physical.Quantity  
Definition of the unit

**tobase** : function, optional  
to convert a quantity to base units

**frombase** : function, optional  
to convert a quantity from the base units

**return** : Quantity  
The defined quantity

#### **Examples**

```
1 Q.define("L", "liter", _dm^3)
2 Q.define("Pa", "pascal", _N/_m^2)
3 Q.define("C", "coulomb", _A*_s)
4
5 Q.define(
6   "degC",
7   "celsius",
8   _K,
9   function(q)
10     q.value = q.value + 273.15
11     return q
12   end,
13   function(q)
14     q.value = q.value - 273.15
15     return q
16   end
17 )
```

### **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

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

### **Quantity.addPrefix(prefixes, units)**

Create several units with prefixes from a given unit.

**prefixes** : string, list of unit symbols  
**units** : Quantity, list of quantities

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

### **Quantity.to(self,q,usefunction=false)**

Converts the quantity self to the unit of the quantity q. If the boolean **usefunction** is true, the conversion function is used for conversion.

**self** : Quantity  
**q** : Quantity  
**usefunction** : Bool

```
1 s = 1.9 * _km
2 print( s:to(_m) )
3 1900 * _m
4
5 T = 10 * _degC
6 print( T:to(_K) )
7 10 * _K
8 print( T:to(_K,true) )
9 283.15 * _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
```

```
1 s = 1.9 * _km
2
3 print( s:tosiunitx() )
4 \SI{1.9}{\kilo\meter}
5
6 print( s:tosiunitx(nil,1) )
7 \num{1.9}
8
9 print( s:tosiunitx(nil,2) )
10 \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
```

```
1 s_1 = 1.9 * _m
2 s_2 = 2.0 * _m
3 print( s_1:isclose(s_2,0.1) )
4 true
5 print( s_1:isclose(s_2,0.01) )
6 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
```

```
1 s_1 = 15 * _m
2 s_2 = 5 * _m
3 print(s_1:min(s_2))
4 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

```
1 s_1 = 15 * _m
2 s_2 = 5 * _m
3 print(s_1:max(s_2))
4 15 * _m
```

### **Quantity.abs(q)**

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

**q** : Quantity, Number, argument

```
1 U = -5 * _V
2 print(U)
3 -5 * _V
4 print(U:abs())
5 5 * _V
```

### **Quantity.sqrt(q)**

Returns the square root of the given quantity.

**q** : Quantity, Number argument

```
1 A = 25 * _m^2
2 s = sqrt(A)
3 print(s)
4 5 * _m
```

### **Quantity.log(q, base)**

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

**q** : Quantity, Number dimensionless argument  
**base** : Quantity, Number dimensionless argument

```
1 I = 1 * _W/_m^2
2 I_0 = 1e-12 * _W/_m^2
3 print(10 * (I/I_0):log(10) * _dB )
4 120 * _dB
```

### **Quantity.exp(q)**

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

**q** : Quantity, Number dimensionless argument

```
1 x = 2 * _1
2 print( x:exp() )
3 7.3890560989307
```

### **Quantity.sin(q)**

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

**q** : Quantity, Number dimensionless argument

```
1 alpha = 30 * _deg
2 print( alpha:sin() )
3 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

```
1 alpha = 60 * _deg
2 print( alpha:cos() )
3 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

```
1 alpha = 45 * _deg
2 print( alpha:tan() )
3 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

```
1 x = 0.5 * _1
2 print( x:asin():to(_deg) )
3 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

```
1 x = 0.5 * _1
2 print( x:acos():to(_deg) )
3 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

```
1 x = 1 * _1
2 print( x:atan():to(_deg) )
3 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

```
1  x = 1 * _1
2  print( x:sinh() )
3  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

```
1  x = 1 * _1
2  print( x:cosh() )
3  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

```
1  x = 1 * _1
2  print( x:tanh() )
3  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

```
1  x = 1 * _1
2  print( x:asinh() )
3  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.

```
1  x = 2 * _1
2  print( x:acosh() )
3  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

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

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

```
1  x = 0.5 * _1
2  print( x:atanh() )
3  0.54930614433405
```