The LUA-PHYSICAL library

Version 0.1

Thomas Jenni

March 16, 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 of physics 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

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. LuaLaTeX does make it possible to integrate physical calculations directly into the LuaLaTeX file. The package has been in use since 2016. Since then many bugs have been found and crushed. Nevertheless it still could be possible, that some were not found. 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 calculation 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.

```latex
1    \usepackage{lua-physical}
2    \usepackage{siunitx}
3
4    % configure siunitx
5    \sisetup{
6      output-decimal-marker = {.},
7      per-mode = symbol,
8      separate-uncertainty = false,
9      add-decimal-zero = true,
10     exponent-product = \cdot,
11     round-mode = off
12   }
13
14   % load lua-physical package
15   \begin{luacode*}
16     physical = require("physical")
17
18     _N = physical.Number
19     _N.omitUncertainty = true
20   \end{luacode*}
21
22   % print physical quantities
23   \newcommand{\q}[1]{%
24     \directlua{tex.print(physical.Quantity.tosiunitx(#1,"scientific-
           notation=fixed,exponent-to-prefix=false"))}%
25   }
```

Listing 1: basic preamble

2.1 Dependencies

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

3 Calculate with physical quantities

Given the basic preamble, units can be used in lua code directly. The Result of the calculation can be printed to TeX by using the macro `\q{}`.

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

A car travels $10\,\mathrm{m}$ in $2\,\mathrm{s}$. Calculate its velocity.

$$v = \frac{s}{t} = \frac{10\,\mathrm{m}}{2\,\mathrm{s}} = 5\,\mathrm{m/s}$$

The physical quantities `s` for displacement, `t` for time, do have the units meter `_m` and second `_s`. Once these variables are declared, the calculation of the velocity is straight forward. In the LaTeX code the quantities are printend in the problem statement and in the solution.
Another example is the following physics problem.

```
1  \begin{luacode}
2    p = 1.013 * _bar
3    n = 1 * _mol
4    T = 30 * _degC
5
6    V = ( n * _R * T:to(_K,true) / p ):to(_L)
7  \end{luacode}
8
9  An ideal gas ($\q{n}$) has a pressure of $\q{p}$ and a temperature
          of $\q{T}$. Calculate the volume of the gas.
10 $$
11   V=\frac{ \q{n} \cdot \q{_R} \cdot \q{T:to(_K,true)} }{ \q{p} }
12   = \uuline{\q{V}}
13 $$
```

An ideal gas ($1 \, \mathrm{mol}$) has a pressure of $1013 \, \mathrm{hPa}$ and a temperature of $0 \, \mathrm{°C}$. Calculate the volume of the gas.

$$V = \frac{1 \, \mathrm{mol} \cdot 8.31 \, \mathrm{J/(mol \, K)} \cdot 273.15 \, \mathrm{K}}{1013 \, \mathrm{hPa}} = \underline{\underline{22.4195 \, \mathrm{L}}}$$

4 Supported Units

4.1 Base Units

Quantity	Unit	Symbol	Dim.	Definition
number	–	1	1	The dimensionless number one.
length	meter	_m	L	The distance light travels in vacuum during $1/299\,792\,458$ second.
mass	kilogram	_kg	M	The mass of the international protoype of the kilogram.
time	second	_s	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	_A	I	The constant current which, if maintained in two straight parallel conductors of infinte length, of negligible circular crosss-section, and placed $1\,\mathrm{m}$ apart in vacuum, would produce between these conductors a force equal to $2 \cdot 10^{-7}\,\mathrm{N/m}$.
thermodynamic temperature	kelvin	_K	Θ	Is the fraction $1/273.16$ of the thermodynamic temperature of the triple point of water.
amount of substance	mole	_mol	N	Amount of substance that contains as many particles as there are atoms in $0.012\,\mathrm{kg}$ of carbon-12.
luminous intensity	candela	_cd	J	The luminous intensity, in a given direction, of a source that emits monochromatic radiation of frequenc $540 \cdot 10^{12}\,\mathrm{Hz}$ and has a radiant intensity in that direction of $(1/683)\,\mathrm{W/sr}$
information	bit	_bit	B	The smallest amount of information.
currency	euro	_EUR	L	The value of the currency Euro.

Table 1: Base units of the International System of Units (SI) expanded with the base units of information and currency.

4.2 Derived Units

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

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

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

Quantity	Unit	Symbol	Dim.	Definition
	percent %	_percent	1	1e-2*_1
	permille ‰	_permille	1	1e-3*_1
	parts-per-million	_ppm	1	1e-6*_1
	parts-per-billion	_ppb	1	1e-9*_1
	parts-per-trillion	_ppt	1	1e-12*_1
	parts-per-quadrillion	_ppq	1	1e-15*_1
	decibel	_dB	1	_1
plane angle	degree	_deg	1	(Pi/180)*_rad
	arc minute	_arcmin	1	_deg/60
	arc second	_arcsec	1	_arcmin/60
	gradian	_gon	1	(Pi/200)*_rad
	turn	_tr	1	2*Pi*_rad
solid angle	spat	_sp	1	4*Pi*_sr
length	astronomical unit	_au	L	149597870700*_m
	lightyear	_ly	L	_c*_a
	parsec	_pc	L	(648000/Pi)*_au
	angstrom	_angstrom	L	1e-10*_m
	fermi	_fermi	L	1e-15*_m
area	are	_ar	L^2	1e2*_m^2
	hectare	_hectare	L^2	1e4*_m^2
	barn	_barn	L^2	1e-28*_m^2
volume	liter	_L	L^3	0.001*_m^3
	metric teaspoon	_tsp	L^3	0.005*_L
	metric tablespoon	_Tbsp	L^3	3*_tsp
time	minute	_min	T	_60*_s
	hour	_h	T	_60*_min
	day	_d	T	_24*_h
	week	_wk	T	_7*_d
	year	_a	T	365.25*_d
	svedberg	_svedberg	T	1e-13*_s
mass	tonne	_t	M	1000*_kg

Table 3: Units outside of the International System of Units (SI)

Quantity	Unit	Symbol	Dim.	Definition
length	inch	`_in`	L	`0.0254*_m`
	thou	`_th`	L	`0.001*_in`
	pica	`_pica`	L	`_in/6`
	point	`_pt`	L	`_in/72`
	hand	`_hh`	L	`4*_in`
	foot	`_ft`	L	`12*_in`
	yard	`_yd`	L	`3*_ft`
	rod	`_rd`	L	`5.5*_yd`
	chain	`_ch`	L	`4*_rd`
	furlong	`_fur`	L	`10*_ch`
	mile	`_mi`	L	`8*_fur`
	league	`_lea`	L	`3*_mi`
	nautical mile	`_nmi`	L	`1852 * _m`
	nautical league	`_nlea`	L	`3*_nmi`
	cable	`_cbl`	L	`_nmi/10`
	fathom	`_ftm`	L	`6*_ft`
velocity	knot	`_kn`	$L^1\,T^{-1}$	`_nmi/_h`
area	acre	`_ac`	L^2	`43560*_ft^2`
volume	gallon	`_gal`	L^3	`4.54609*_L`
	quart	`_qt`	L^3	`_gal/4`
	pint	`_pint`	L^3	`_qt/2`
	cup	`_cup`	L^3	`_pint/2`
	gill	`_gi`	L^3	`_pint/4`
	fluid ounce	`_fl_oz`	L^3	`_gi/5`
	fluid dram	`_fl_dr`	L^3	`_fl_oz/8`

Table 4: Imperial units

Quantity	Unit	Symbol	Dim.	Definition
mass	grain	_gr	M	64.79891*_mg
	pound	_lb	M	7000*_gr
	ounce	_oz	M	_lb/16
	dram	_dr	M	_lb/256
	stone	_st	M	14*_lb
	quarter	_qtr	M	2*_st
	hundredweight	_cwt	M	4*_qtr
	long ton	_ton	M	20*_cwt

Table 5: Imperial units

Quantity	Unit	Symbol	Dim.	Definition
length	U.S. survey inch	_in_US	L	_m/39.37
	U.S. survey hand	_hh_US	L	4*_in_US
	U.S. survey foot	_ft_US	L	3*_hh_US
	U.S. survey link	_li_US	L	0.66*_ft_US
	U.S. survey yard	_yd_US	L	3*_ft_US
	U.S. survey rod	_rd_US	L	5.5*_yd_US
	U.S. survey chain	_ch_US	L	4*_rd_US
	U.S. survey furlong	_fur_US	L	10*_ch_US
	U.S. survey mile	_mi_US	L	8*_fur_US
	U.S. survey league	_lea_US	L	3*_mi_US
	U.S. survey fathom	_ftm_US	L	72*_in_US
	U.S. survey cable	_cbl_US	L	120*_ftm_US

Table 6: U. S. customary units

Quantity	Unit	Symbol	Dim.	Definition
currency	Afghan afghani	_AFN	C	0.012*_EUR
	Albanian lek	_ALL	C	0.008*_EUR
	Armenian Dram	_AMD	C	0.0018*_EUR
	Angolan Kwanza	_AOA	C	0.0028*_EUR
	Argentine Peso	_ARS	C	0.021*_EUR
	U.S. dollar	_USD	C	0.89*_EUR
	Japanese yen	_JPY	C	0.008*_EUR
	British pound	_GBP	C	1.17*_EUR
	Australian dollar	_AUD	C	0.63*_EUR
	Canadian dollar	_CAD	C	0.66*_EUR
	Swiss franc	_CHF	C	0.88*_EUR
	Chinese yuan	_CNY	C	0.13*_EUR
	Swedish krona	_SEK	C	0.094*_EUR
	New Zealand dollar	_NZD	C	0.60*_EUR

Table 7: 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.

```
1  local D = physical.Dimension
2  local U = physical.Unit
3  local N = physical.Number
4  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

Parameters

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

return : `Q`
    The created `Q` instance

Note
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

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

Q.defineBase(symbol,name,dimension)

This function is used to declare the base units. Units are represented as `Q` instances.

Parameters

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

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

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

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

Note

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 with it. The argument `dimension` allows any dimension to be associated to base quantities. By default, the SI convention is used.

Example

```
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 convertion 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 convertion 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
```

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

```
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 quantitiy.

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 quantitiy.

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

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

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

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

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

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

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

```
1   x = 0.5 * _1
2   print( x:atanh() )
3   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 two check if two quantities can be added or subtraced 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 ist 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)
```

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.

Parameters

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.

Parameters

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.

Parameters

return : String
   The siunitx representation of the unit term.